Lecture Notes

Compression Technologies and Multimedia Data Formats

Ao.Univ.-Prof. Dr. Andreas Uhl

Department of Computer Sciences

University of Salzburg

Adress:

Andreas Uhl
Department of Computer Sciences
J.-Haringerstr.2
A-5020 Salzburg
Österreich
Tel.: ++43/(0)662/8044/6303
Fax: ++43/(0)662/8044/172
E-mail: uhl@cosy.sbg.ac.at

Inhaltsverzeichnis

Kapitel 1

Basics

1.1 What is Compression Technology ?

Compression denotes compact representation of data. In this lecture we exclusively cover compression of *digital* data. Examples for the kind of data you typically want to compress are e.g.

- text
- source-code
- arbitrary files
- images
- video
- audio data
- speech

Obviously these data are fairly different in terms of data volume, data structure, intended usage etc. For that reason it is plausible to develop specific compression technologies for different data and application types, respectively. The idea of a general purpose compression technique for all thinkable application scenarios has to be entirely abandoned. Instead, we find a large number of very different techniques with respect to target data types and target application environments (e.g. data transmission in networks like streaming, storage and long term archiving applications, interactive multimedia applications like gaming etc.). Given this vast amount of different techniques, there are different ways how to classify compression techniques:

- with respect to the type of data to be compressed
- with respect to the target application area
- with respect the the fundamental building blocks of the algorithms used

Terminology: when talking abount *compression*, we often mean “lossy compression” while “lossless compression” is often termed as *coding*. However, not all coding algorithm do actually lead to lossless compression, e.g. error correction codes. Like in every other field in computer science or engineering, the dominating language in compression technologies is English of course. There are hardly any comprehensive and up-to-date German books available, and there do NOT exist any German journals in the field. *Codec* denotes a complete system capable of encoding and decoding data which consists of an *Encoder* and a *Decoder*, *transcoding* is a conversion from one encoded digital representation into another one. A fundamental term in the area is compression rate (or compression ratio) which denotes the relation between the size of the original data before compression and the size of the compressed data. Compression ratio therefore rates the effectivity of a compression system in terms of data reduction capability. This must not be confused with other measures of compressed data size like bit per pixel (bpp) or bit per sample (bps).

1.2 Why do we still need compression ?

Compression Technology is employed to efficiently use storage space, to save on transmission capacity and transmission time, respectively. Basically, its all about saving resources and money. Despite of the overwhelming advances in the areas of storage media and transmission networks it is actually quite a surprise that still compression technology is required. One important reason is that also the resolution and amount of digital data has increased (e.g. HD-TV resolution, ever-increasing sensor sizes in consumer cameras), and that there are still application areas where resources are limited, e.g. wireless networks. Apart from the aim of simply reducing the amount of data, standards like MPEG-4, MPEG-7, and MPEG-21 offer additional functionalities.

During the last years three important trends have contributed to the fact that nowadays compression technology is as important as it has never been before – this development has already changed the way we work with multimedial data like text, speech, audio, images, and video which will lead to new products and applications:

- The availability of highly effective methods for compressing various types of data.
- The availability of fast and cheap hardware components to conduct compression on single-chip systems, microprocessors, DSPs and VLSI systems.
- Convergence of computer, communication, consumer electronics, publishing, and entertainment industries.

1.3 Why is it possible to compress data ?

Compression is enabled by statistical and other properties of most data types, however, data types exist which cannot be compressed, e.g. various kinds of noise or encrypted data. Compression-enabling properties are:

- Statistical redundancy: in non-compressed data, all symbols are represented with the same number of bits independent of their relative frequency (fixed length representation).

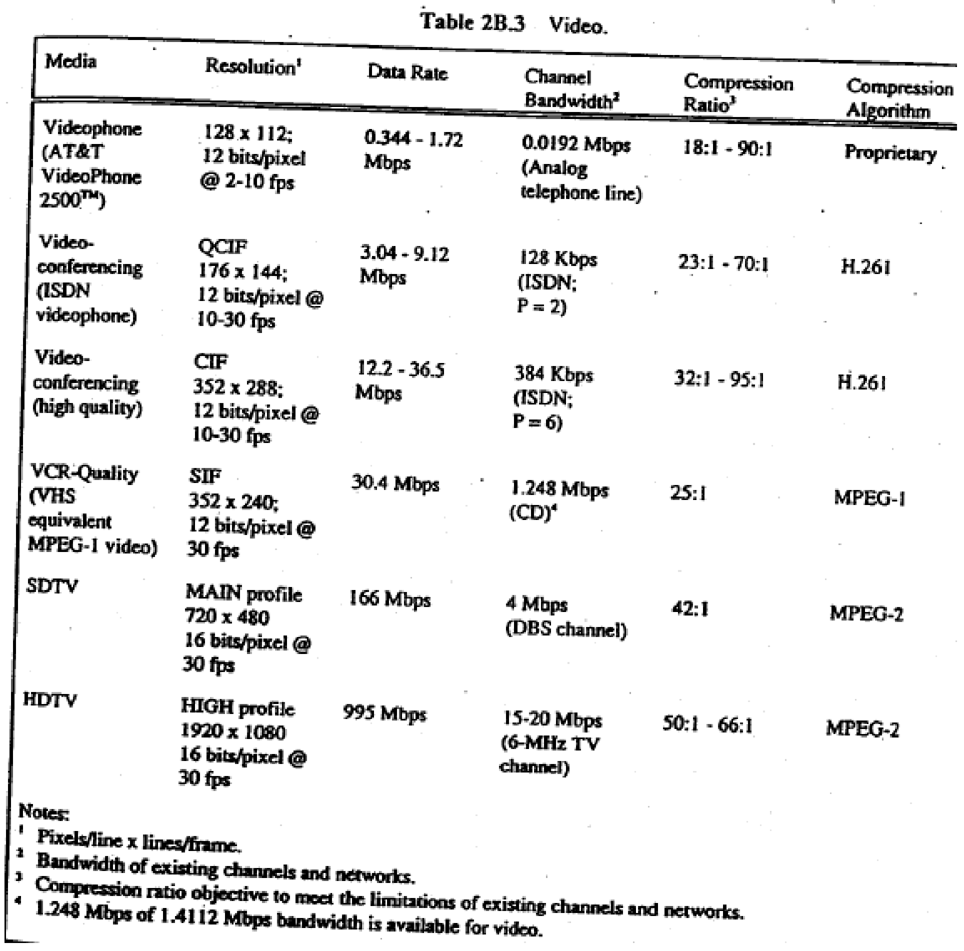


Table 2B.3 Video.

Media	Resolution[1]	Data Rate	Channel Bandwidth[2]	Compression Ratio[3]	Compression Algorithm
Videophone (AT&T VideoPhone 2500™)	128 x 112; 12 bits/pixel @ 2-10 fps	0.344 - 1.72 Mbps	0.0192 Mbps (Analog telephone line)	18:1 - 90:1	Proprietary
Video-conferencing (ISDN videophone)	QCIF 176 x 144; 12 bits/pixel @ 10-30 fps	3.04 - 9.12 Mbps	128 Kbps (ISDN; P = 2)	23:1 - 70:1	H.261
Video-conferencing (high quality)	CIF 352 x 288; 12 bits/pixel @ 10-30 fps	12.2 - 36.5 Mbps	384 Kbps (ISDN; P = 6)	32:1 - 95:1	H.261
VCR-Quality (VHS equivalent MPEG-1 video)	SIF 352 x 240; 12 bits/pixel @ 30 fps	30.4 Mbps	1.248 Mbps (CD)[4]	25:1	MPEG-1
SDTV	MAIN profile 720 x 480 16 bits/pixel @ 30 fps	166 Mbps	4 Mbps (DBS channel)	42:1	MPEG-2
HDTV	HIGH profile 1920 x 1080 16 bits/pixel @ 30 fps	995 Mbps	15-20 Mbps (6-MHz TV channel)	50:1 - 66:1	MPEG-2

Notes:
[1] Pixels/line x lines/frame.
[2] Bandwidth of existing channels and networks.
[3] Compression ratio objective to meet the limitations of existing channels and networks.
[4] 1.248 Mbps of 1.4112 Mbps bandwidth is available for video.

- Correlation: adjacent data samples tend to be equal or similar (e.g. think of images or video data). There are different types of correlation:
  - Spatial correlation
  - Spectral correlation
  - Temporal correlation

In addition, in many data types there is a significant amount of *irrelevancy* since the human brain is not able to process and/or perceive the entire amount of data. As a consequence, such data can be omitted without degrading perception. Furthermore, some data contain more abstract properties which are independent of time, location, and resolution and can be described very efficiently (e.g. fractal properties).

1.4 History of compression technologies

- 1st century B.C.: Stenography

- 19th century: Morse- and Braille alphabeths
- 50ies of the 20th century: compression technologies exploiting statistical redundancy are developed – bit-patterns with varying length are used to represent individual symbols according to their relative frequency.
- 70ies: dictionary algorithms are developed – symbol sequences are mapped to shorter indices using dictionaries.
- 70ies: with the ongiong digitization of telephone lines telecommunication companies got interested in procedures how to get more channels on a single wire.
- early 80ies: fax transmission over analog telephone lines.
- 80ies: first applications involving digital images appear on the market, the "digital revolution" starts with compressing audio data
- 90ies: video broadcasting, video on demand, etc.

1.5 Selection criteria for chosing a compression scheme

If it is evident that in an application compression technology is required it has to be decided which type of technology should be employed. Even if it is not evident at first sight, compression may lead to certain surprising benefits and can offer additional functionalities due to saved resources. When selecting a specific system, quite often we are not entirely free to chose due to standards or de-facto standards – due to the increasing develeopment of open systems and the eventual declining importance of standards (example: MPEG-4 standardization !) these criteria might gain even more importance in the future. Important selection criteria are for example:

- data dimensionality: 1-D, 2-D, 3-D, ...
- lossy or lossless compression: dependent of data type, required quality, compression rate
- quality: with target quality is required for my target application ?
- algorithm complexity, speed, delay: on- or off-line application, real-time requirements
- hardware or software solution: speed vs. price (video encoder boards are still costly)
- encoder / decoder symmetry: repeated encoding (e.g. video conferencing) vs. encoding only once but decoding often (image databases, DVD, VoD, ...)
- error robustness and error resilience: do we expect storage or transmission errors
- scalability: is it possible to decode in different qualities / resolutions without hassle ?
- progressive transmission: the more data we transmit, the better the quality gets.

- standard required: do we build a closed system which is not intended to interact with other systems (which can be desired due to market position, e.g. medical imaging) or do we want to exchange data across many systems
- multiple encoding / decoding: repeated application of lossy compression, e.g. in video editing
- adaptivity: do we expect data with highly varying properties or can we pre-adapt to specific properties (fingerprint compression)
- synchronisation issues – audio and video data should both be frame-based preferably
- transcoding: can the data be converted into other datatypes easisly (e.g. MJPEG $-->$ MPEG)

Kapitel 2

Fundamental Building Blocks

2.1 Lossless Compression

In lossless compression (as the name suggests) data are reconstructed after compression without errors, i.e. no information is lost. Typical application domains where you do not want to loose information is compression of text, files, fax. In case of image data, for medical imaging or the compression of maps in the context of land registry no information loss can be tolerated. A further reason to stick to lossless coding schemes instead of lossy ones is their lower computational demand. For all lossless compression techniques there is a well known trade-off: Compression Ratio – Coder Complexity – Coder Delay.

Lossless Compression typically is a process with three stages:

- The model: the data to be compressed is analysed with respect to its structure and the relative frequency of the occuring symbols.
- The encoder: produces a compressed bitstream / file using the information provided by the model.
- The adaptor: uses informations extracted from the data (usually during encoding) in order to adapt the model (more or less) contineously to the data.

The most fundamental idea in lossless compression is to employ codewords which are shorter (in terms of their binary representation) than their corresponing symbols in case the symbols do occur frequently. On the other hand, codewords are longer than the corresponding symbols in case the latter do not occur frequently (there's no free lunch !).

Each process which generates information can be thought of a source emitting a sequence of symbols chosen from a finite alphabeth (in case of computer-based processing this boils down to the binary alphabeth 0,1). We denote this "source of information" by S with the corresponding alphabeth $\{s_1, s_2, \ldots, s_n\}$ and their occurance probabilities $\{p(s_1), p(s_2), \ldots, p(s_n)\}$. To make things simpler (and more realistic as well), we consider these probabilities as being relative frequencies. We want to determine the average information a source emits. Following intuition, the occurence of a less frequent symbol should provide more information compared to the occurence of highly frequent symbols. We assume symbols to be independent and consider the information provided by such symbols as being the sum of information as given by the single independent symbols. We define

$$I(s_i) = \log \frac{1}{p(s_i)}$$

as being the information delivered by the occurence of the symbol s_i related to its probability $p(s_i)$. The basis of the logarithm determines the measure of the amount of information, e.g. in case of basis 2 we talk about bits.

By averaging over all symbols emitted by the source we obtain the average information per symbol, the *entropy*:

$$H(S) = \sum_{i=1}^{n} p(s_i) I(s_i) = -\sum_{i=1}^{n} p(s_i) \log_2 p(s_i) \quad \text{bits/symbol}$$

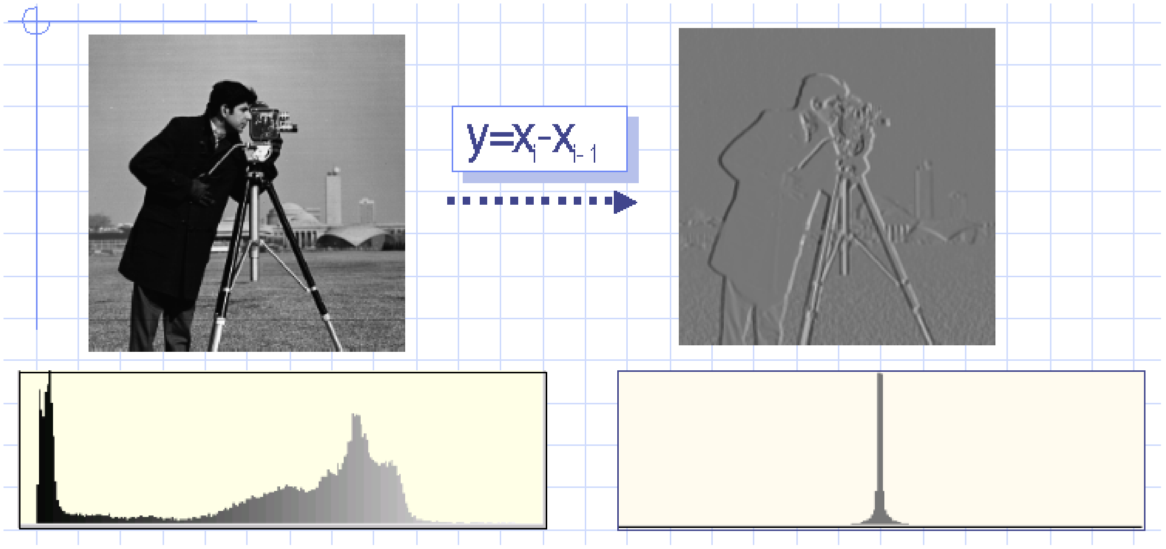
The most significant interpretation of entropy in the context of lossless compression is that entropy measures how many bits are required on average per symbol in order to represent the source, i.e. to conduct lossless compression. Entropy therefore gives a lower bound on the number of bits required to represent the source information. In the next section we will discuss several techniques approaching more or less closely this theoretical bound.

2.1.1 Influence of the Sampling Rate on Compression

If we compare data originating from an identical analog source where y_i is sampled ten times as densly as x_i, we notice that neighbouring values of y_i tend to be much more similar as compared to neighbouring values in x_i. If we assume that the analog signal exhibits continuity to some degree, the samples of y_i will be more contineous (i.e. more similar) as compared to the samples of x_i since these are taken from locations in the signal more dislocated as the samples of y_i. The higher degree of similarity in y_i has a positive effect on entropy and therefore on the achievable compression ratio.

2.1.2 Predictive Coding - Differential Coding

This technique is not really a compression scheme but a procedure to preprocess data in order to make them more suited for subsequent compression. Differential coding changes the statistics of the signal – based on the fact that neighbouring samples in digital data are often similar or highly correlated, differential coding does not compress the sequence of original data points $\{x_1, x_2, \ldots, x_N\}$ but the sequence of differences $y_i = x_i - x_{i-1}$. While x_i usually follow a uniform or normal distribution, y_i exhibit significant peaks around 0 (always assuming that neighbouring samples tend to be similar as it is the case in audio, image, and video data). To give an example, we consider an image with 999 pixels and 10 different symbols. In case a) we have $p(s_i) = 1/10$, $i = 1, \ldots, 10$, in case b) we have $p(s_1) = 91/100$ and $p(s_i) = 1/100$ for $i = 2, \ldots, 10$. Case a) corresponds to the "conventional" image with unifomly distributed symbols, while case b) is the differential image consisting of y_i, where s_1 is the zero symbol. In case a) we result in $H(S) = 3.32$, while in case b) entropy is only $H(S) = 0.72$. This small example immediately makes clear that differential coding is sensible. Actually, it is the basis of many compression schemes employing prediction.



2.1.3 Runlength Coding

The basic principle is to replace sequences of successive identical symbols by three elements: a single symbol, a counter, and an indicator which gives the interpretation of the other two elements.

As an example we discuss the tape drive of the IBM AS/400 (where obviously lots of blanks are encountered): strings of consecutive blanks (2-63 byte) are compressed into a codeword which contains the information "blank" and a counter. Strings of consecutive identical characters unequal blank (3-63 byte) are compressed into 2 byte: Control byte ("connsecutive char" and counter) and character byte. Strings of non-consecutive identical characters are expanded by a control byte ("non- consecutive identical").

Example: b Blank, * control byte; bbbbbbABCDEFbb33GHJKbMN333333 is compressed into **ABCDEF**33GHJKbMN*3

ATTENTION: if the data is not suited for runlength encoding we result in data expansion (caused by the expansion of non-consecuritve identical chars). Runlengh coding is extremely fast on the other hand and can compress quite well in case of suited data.

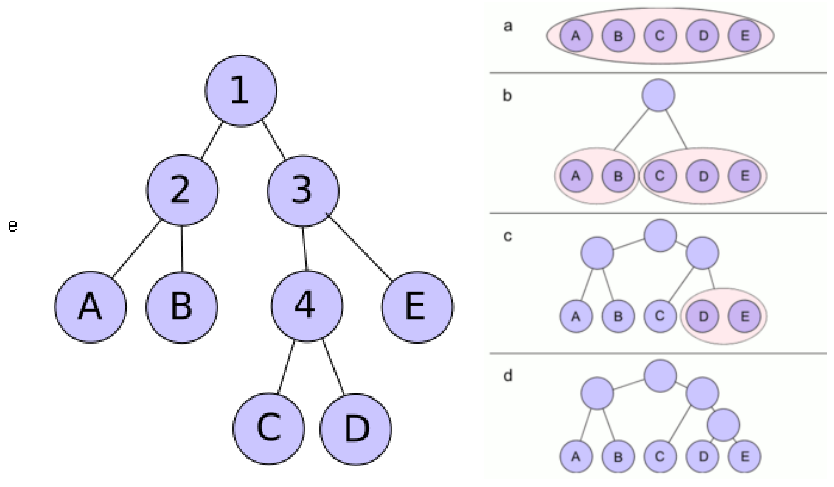
2.1.4 Variable length Coding and Shannon-Fano Coding

We denote the length of the codeword for the symbol s_i as $l(s_i)$ (in bits). The average code length is defined as: $L(S) = \sum_{i=1}^{n} p(s_i)l(s_i)$. In order to minimize this expression (which obviously is the aim of compression) we need to assign short codewords to symbols with high probability of occurence. It is intersting to notice the close relation to entropy: $H(S) = \sum_{i=1}^{n} p(s_i)I(s_i)$. Consequently it is evendent that $l(s_i) = I(s_i) = -\log_2 p(s_i)$ needs to be fulfilled in order to attain the actual entropy value. However, this is possible only in case $\log_2 p(s_i)$ is a natural number. Usually, this is not the case of course, so the value is rounded to the next integer (which is a hint to the sub-optimality of the entire procedure !). Shannon-Fano Coding (see below) exactly provides the corresponding solution using the idea described before. The term code efficiency is defined as $\frac{H(S)}{L(S)}$ – obviously values close to 1 are desirable !

Example: $S = \{s_1, s_2, s_3, s_4\}$ with $p(s_1) = 0.6$, $p(s_2) = 0.3$, $p(s_3) = 0.05$ and $p(s_4) = 0.05$. $H(S) = 1.4$ bits/symbol. When chosing a fixed length encoding like e.g. 00, 01, 10, 11 as codewords with average code length of 2 bits/symbol. When computing $l(s_i)$ we obtain the values 1,2, and 5 and result in the following variable length code: 0, 10, 110 und 111. s_3 and s_4 would require a 5 bit codeword but 3 bits are sufficient to generate a unique code. The corresponding average code length is 1.5 bits/symbol.

A *prefix code* (or prefix-free code) is a code system, typically a variable-length code, with the "prefix property": there is no valid code word in the system that is a prefix (start) of any other valid code word in the set. E.g., a code with code words 9, 59, 55 has the prefix property; a code consisting of 9, 5, 59, 55 does not, because 5 is a prefix of both 59 and 55. With a prefix code, a receiver can identify each word without requiring a special marker between words.

A way to generate a prefix code is to use a full binary tree to represent the code. The leaves represent the symbols to be coded while path traversing the tree from the root to the leave determines the bit-sequence of the resulting codeword. To actually determine a bitsequence, we need to define how to encode a branching: for example, a branching to the left can be encoded by 0 and a branching to the right by 1. The tree given in the figure leads to the following codewords: A – 00, B – 01, C – 100, D – 101, E – 1. The tree (and code) generation of the Shannon-Fano scheme works as follows (in the example, we assume the following frequency counts: A – 15, B – 7, C – 6, D – 6, E – 5):



1. For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol's relative frequency of occurrence is known.

2. Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.

3. Divide the list into two parts, with the total frequency counts of the left half being as close to the total of the right as possible.

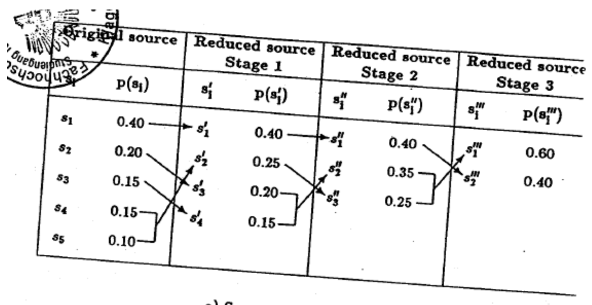
4. The left half of the list is assigned the binary digit 0, and the right half is assigned the digit 1. This means that the codes for the symbols in the first half will all start with 0, and the codes in the second half will all start with 1.

5. Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

A procedure often employed is to replace the alphabeth consisting of the s_i by an extension using tuples $s_i s_j$ with their corresponding relative frequencies. Using this strategy, a code length of 1.43 bits/symbol can be achieved, however, at a higher cost in terms of computation and memory requirements.

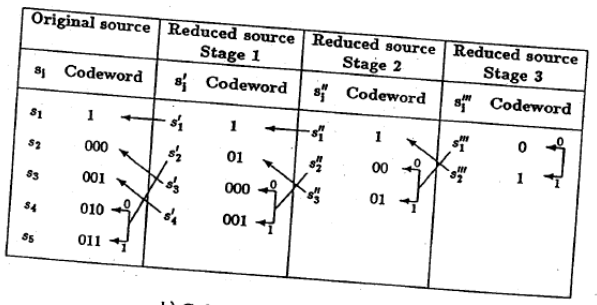
2.1.5 Huffman Coding

Huffman Coding is a popular technique using the idea of variable length coding combined with a constructive algorithm how to build the corresponding unique codewords. Suppose we have given a source S with an alphabeth of size n. The two least frequent symbols are combined into a new virtual symbol which is assigned the sum of occurence probabilites of the two original symbols. Subsequently, the new alphabeth of size $n-1$ is treated the same way, which goes on until only two symbols are left. If we know the codewords of the new symbols, the codewords for the two original symbols are obtained by adding a 0 and 1 bit to the right side of the new symbol. This procedure is applied recursively, i.e. starting with the two most frequent symbols which are assigned codewords 0 and 1, we successively add corresponding bits to the right until codewords for all original symbols have been generated. The example has entropy $H(S) = 2.15$, the gererated Huffman code have average code length of 2.2 bits/symbol, an ad-hoc generated code like shown before, like e.g. $\{0, 10, 110, 1110, 1111\}$ has average code length of 2.25 bits/symbol.



Original source		Reduced source Stage 1		Reduced source Stage 2		Reduced source Stage 3	
s_i	$p(s_i)$	s'_i	$p(s'_i)$	s''_i	$p(s''_i)$	s'''_i	$p(s'''_i)$
s_1	0.40	s'_1	0.40	s''_1	0.40	s'''_1	0.60
s_2	0.20	s'_2	0.25	s''_2	0.35	s'''_2	0.40
s_3	0.15	s'_3	0.20	s''_3	0.25		
s_4	0.15	s'_4	0.15				
s_5	0.10						

a) Source reduction process



Original source		Reduced source Stage 1		Reduced source Stage 2		Reduced source Stage 3	
s_i	Codeword	s'_i	Codeword	s''_i	Codeword	s'''_i	Codeword
s_1	1	s'_1	1	s''_1	1	s'''_1	0
s_2	000	s'_2	01	s''_2	00	s'''_2	1
s_3	001	s'_3	000	s''_3	01		
s_4	010	s'_4	001				
s_5	011						

b) Codeword construction process

Modification: in case many symols have small probabilities of occurence, a large number of long codewords is the result which is not efficient. Therefore, all such symbols are assinged into a dedicated class which is termed "ELSE" in the Huffman code and which is encoded separately. This idea is calles *modified Huffman code*.

Problems:

1. In case a source has a symbol with $p(s)$ close to 1 and many others with small probability of occurence we result in an average code length of 1 bit/symbol since the smallest length for a codeword is of course 1 bit. The entropy is of course much smaller (recall the coding of differences).

2. In case of changing statistics of the source one can easily obtain a data expansion.

3. Since a fixed codebook is of poor quality in many cases we have a two stage algorithm: building statistics (the model), generate the code.

4. Adaptivity is difficult to implement, since changes in the statistics affect the entire tree and not just a local part. We can either store corresponding Huffman tables (trees) in the data [which is inefficient in terms of compression] or compute them on the fly from decoded data [which is inefficient in terms of compression speed].

Huffman encoding is used in most older standards like JPEG, MPEG-1 to MPEG-4, however, in a non-adaptive manner which is possible due to the nature of the DCT transform.

2.1.6 Arithmetic Coding

In Huffman coding we have a correspondence between a single symbol and its codeword – which is the main reason for its suboptimality. Arithmetic coding uses a single codeword for an entire sequence of source symbols of length m. In this manner the restriction to integer-valued bits per symbol values can be avoided in an elegant way. A drawback however is that similar sequences of source symbols can lead to entirely different codewords.

The principal idea is as follows. Suppose we have a sequence of binary source symbols s_m with length m where $p(0) = p$ and $p(1) = 1 - p$. Let $I = [0, 1)$ be the semi-open unit interval and $\sum p(s_m) = 1$ when summing over all 2^m possible sequences of length m. Based on this initial construction we can build subintervals I_l with $l = 1, 2, \ldots, 2^m$ which are all in I and which can be assinged in unique manner to a sequence s_m where $|I_l| = p(s_m)$ and all I_j are disjoint. The following procedure constructs such intervals and is called “Elias Code” (this approach is meant for illustrative purposes only since it is inpractical for implementations).

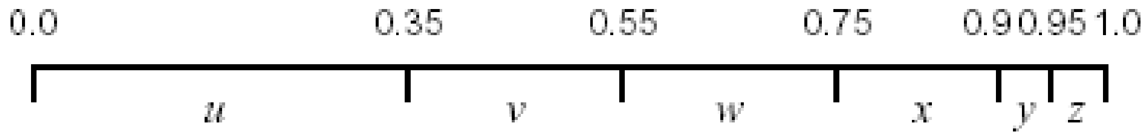
The interval I is partitioned into two subintervals [0,p) (in case 0 is the symbol to be coded) and [p,1) (in case 1 is to be coded). Based on the occurance of the next symbol (0 or 1) these intervals are further partitioned recursively into $[0, p^2)$ and $[p^2, p)$ and into $[p, 2p - p^2)$ and $[2p - p^2, 1)$, respectively. The interval $[p^2, p)$ for example is assigned to the sequence 01. After $j - 1$ source symbols the interval $[L^{j-1}, R^{j-1})$ has been assigned to the sequence s_{j-1}.

- 0 is next: $L^j = L^{j-1}$ and $R^j = L^{j-1} + p(R^{j-1} - L^{j-1})$.
- 1 is next: $L^j = L^{j-1} + p(R^{j-1} - L^{j-1})$ und $R^j = R^{j-1}$.

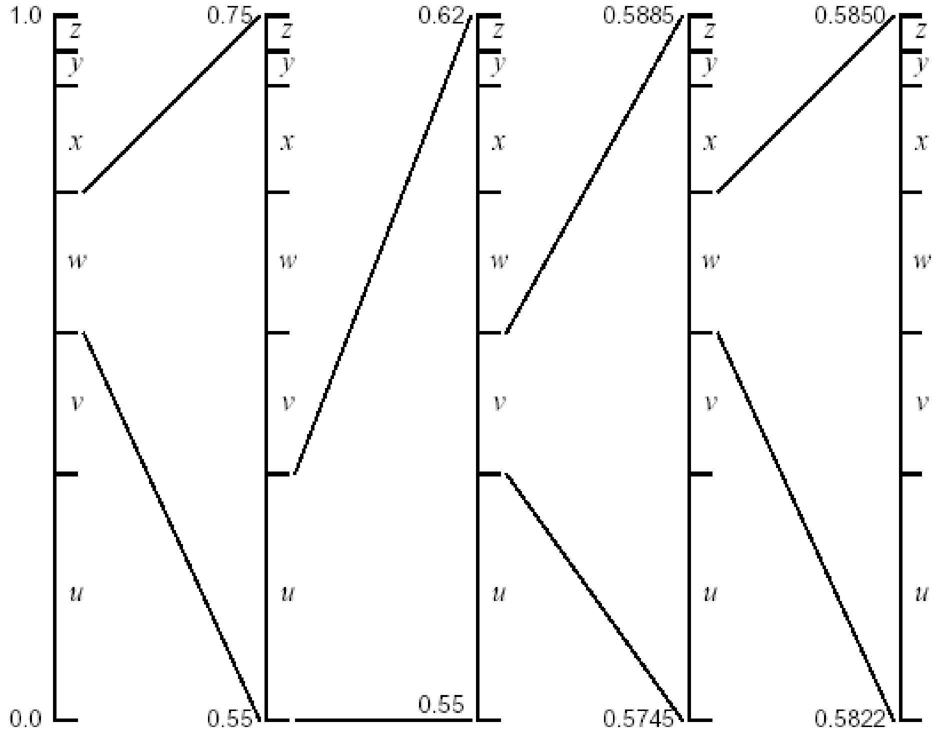
Based on this construction one can show that the length of each interval $[L^j, R^j)$ has the same length as $p(s_m)$ of the corresponding sequence. As soon as the interval has been constructed, the codeword for s_m is found easily: we represent L^j in binary manner and keep $-\log_2 p(s_m)$ bits (rounded to next integer) after the decimal point. This is sufficient to uniquely identify the interval and the corresponding symbol sequence. In actual implementations it is sufficient to encode an arbitrary point in the constructed interval (of course a point with minimal length in the binary representation is selected) and the number of encoded symbols. An important fact is that in small intervals, more binary positions are required to represent a point (since no points with small “binary length” are contained).

Actual implementations use a re-normalisation to $[0, 1)$ in each interval partitioning stage due to problems with rounding errors.

Example: Encoding the sequence $wuvw$ leads to the interval $[0.5822, 0.5850]$ the length of which is 0.0028 $= p(w)p(u)p(v)p(w)$. In binary representation the interval borders are 0.1001010111000_2 and 0.1001010100001_2,



repsectively. The shortest binary number in this interval is 0.100101011_2 which is the codeword for this sequence. When computing the entropy of the sequence, we get $-log_2 0.0028 = 8.48bit$ which is close to 9.



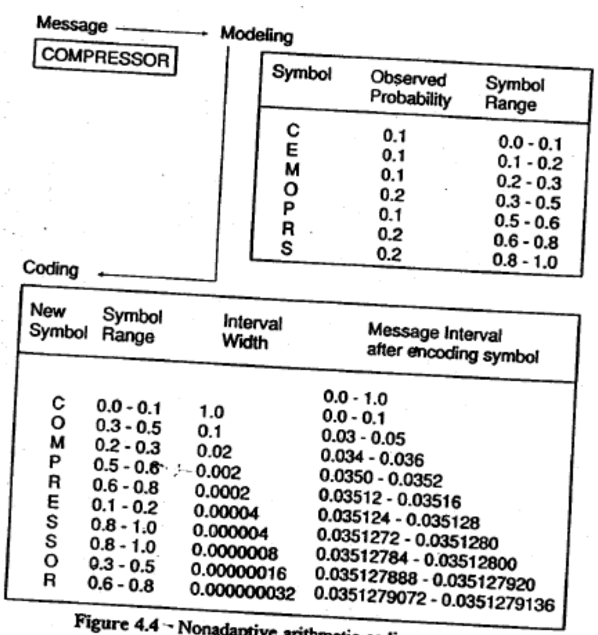
Advantages:

1. Adaptivity can be easily realised – encoder and decoder need to carry the statistical information in identical (i.e. synchronized) manner and can adapt the interval boundaries correspondingly. Either for each symbol, or for a fixed amount of symbols, or when a certain extent of statistical change occurs. In any case there is no need to store the statistical information.

2. Already if only a small amount of encoded data is available (i.e. the MSB of the position) we can start decoding, there is no need to wait for the receipt of the entire data to start decoding. With the first symbols we can determine in which of the two initial intervals the final interval will be located, upon receipt of more information also more symbols can be decoded.

3. The restriction to integer bits/symbol values is overcome.

Arithmetic coding is used in JBIG-1 as the only coding engine and in the more recent lossy standards like JPEG2000 and H.264 as lossless compression stage.


Message
Modeling
COMPRESSOR
Coding

Symbol	Observed Probability	Symbol Range
C	0.1	0.0 - 0.1
E	0.1	0.1 - 0.2
M	0.1	0.2 - 0.3
O	0.2	0.3 - 0.5
P	0.1	0.5 - 0.6
R	0.2	0.6 - 0.8
S	0.2	0.8 - 1.0

New Symbol	Symbol Range	Interval Width	Message Interval after encoding symbol
			0.0 - 1.0
C	0.0 - 0.1	1.0	0.0 - 0.1
O	0.3 - 0.5	0.1	0.03 - 0.05
M	0.2 - 0.3	0.02	0.034 - 0.036
P	0.5 - 0.6	0.002	0.0350 - 0.0352
R	0.6 - 0.8	0.0002	0.03512 - 0.03516
E	0.1 - 0.2	0.00004	0.035124 - 0.035128
S	0.8 - 1.0	0.000004	0.0351272 - 0.0351280
S	0.8 - 1.0	0.0000008	0.03512784 - 0.03512800
O	0.3 - 0.5	0.00000016	0.035127888 - 0.035127920
R	0.6 - 0.8	0.000000032	0.0351279072 - 0.0351279136

Figure 4.4 – Nonadaptive arithmetic coding example.

2.1.7 Dictionary Compression

Basic idea: the redundancy of repeated phrases in a text is exploited (recall: in Runlenth encoding, repeated identical symbols can be efficiently coded). Here we encode repeatedly occuring symbol strings in arbitrary files in an efficient manner. A frontend selects strings and information for compression and dekompression is stored in a dictionary. The encoder and the decoder need to have access to a copy of this dictionary of course. In this manner, entire strings are replaced by tokens (i.e. codewords). The basic algorithmic elements are lookup tables, and the corresponding operations are very fast.

LZ77 Dictionary Compression

The name originates from Lempel and Ziv (1977), the algorithm actually described here is a variant, LZSS (Storer/Szymanski) as of 1982. A typical feature is the so-called "sliding window" consisting of two parts: the history buffer (the dictionary) contains a block of data recently encoded (some 1000 symbols) – and the look-ahead buffer which contains strings to be encoded (10-20 symbols).

In important issue is the coice of the buffer sizes:

- History buffer small $->$ match is unlikely.
- History buffer large $->$ high computational requirements and poor compression rate caused by the large address space for the pointers into the buffer.

- Look-ahead buffer small $->$ no match for long strings is possible (waste of potential redundancy).
- Look-ahead buffer large $->$ time is wasted for the unlikely case of finding matches for long strings.

Example: *Text out* < – ——DATA COMPRESSION—— COMPRESSES DATA—— < – *Text in.* "DATA COMPRESSION" is the content of the history buffer, "COMPRESSES DATA" is the look-ahead buffer. The actual coding used subsequently is the QIC-122 standard for tape drives using a history buffer of 2048 bytes.

To start with, we encode each symbol in "DATA COMPRESSION" as 0<ASCII Char. Wert>. "COMPRESS" is a string of length 9 which has been found in the history buffer 12 symbols backwards: 1<offset=12><length=9 Subsequently we encode "E" and "S" again as 0<ASCII Char. Wert> and "DATA" as string of length 4 which has been found in the history buffer 28 positions backwards: 1<offset=28><length=4>.

We notice that

- the algorithm requires a start-up phase where the data in the dictionary has to be encoded explicitely, which means that it is hardly suited for compressing lots of small files.
- The algorithm is assymetric – searching the history buffer in an efficient manner is crucial. Intelligent data structures are required.
- Since data leaves the history buffer after some time, long term redundancy cannot be exploited.

LZ78 Dictionary Compression

The algorithm described here the LZW(elch) Coding variant as of 1984. This algorithm solves the problems arising with the explicit definition of the two buffers. Stings in the dictionary are not automatically reased and the lookahead buffer does not have a fixed size. The algorithm does not employ pointers to earlier complete strings but a procedure called "Phrasing". A text is divided into phrases, where a phrase is the longest string found so far plus an additional character.

All symbols are replaced by topkens (i.e. codewords) which point to entries in the dictionary. The first 256 disctionary entries are the common 8 bit ASCII code symbols. The dictionary is successively extended by new phrases generated from the text (i.e. data) to be coded.

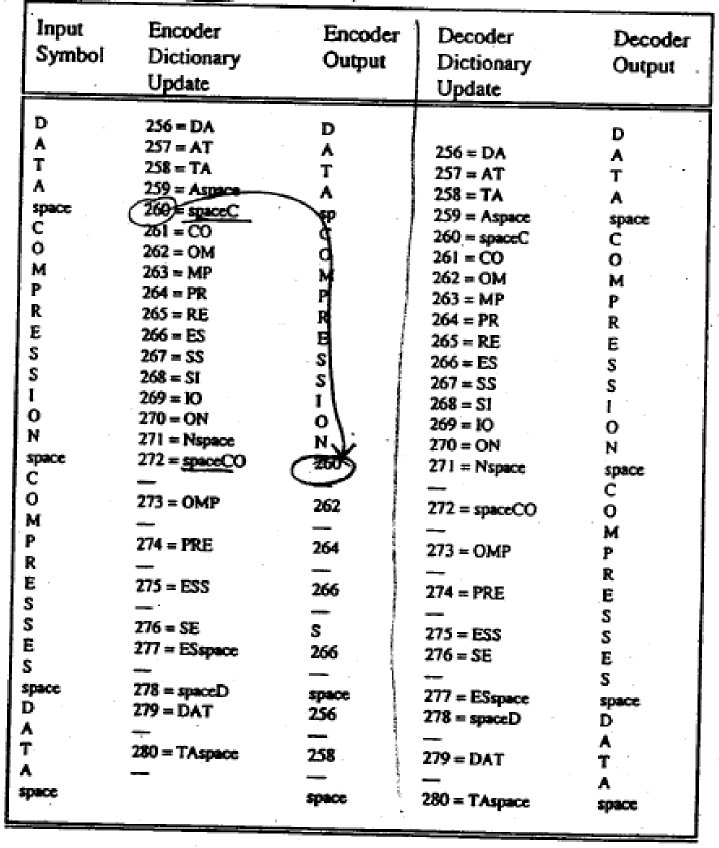
Critical issues:

- How to efficiently encode a variable number of strings with variable length ?
- How long can we grow the dictionary ? What are the strategies when the dictionary gets too large (start from scratch, keep the most common entries) ?

2.1.8 Which lossless algorithm is most suited for a given application ?

Testing with exemplary data is important apart from the facts we know theoretically with respect to speed, assymetry, compression capability etc. !

Table 4.3 LZW compression example.



Input Symbol	Encoder Dictionary Update	Encoder Output	Decoder Dictionary Update	Decoder Output
D	256 = DA	D		D
A	257 = AT	A	256 = DA	A
T	258 = TA	T	257 = AT	T
A	259 = Aspace	A	258 = TA	A
space	260 = spaceC	sp	259 = Aspace	space
C	261 = CO	C	260 = spaceC	C
O	262 = OM	O	261 = CO	O
M	263 = MP	M	262 = OM	M
P	264 = PR	P	263 = MP	P
R	265 = RE	R	264 = PR	R
E	266 = ES	E	265 = RE	E
S	267 = SS	S	266 = ES	S
S	268 = SI	S	267 = SS	S
I	269 = IO	I	268 = SI	I
O	270 = ON	O	269 = IO	O
N	271 = Nspace	N	270 = ON	N
space	272 = spaceCO	260	271 = Nspace	space
C	—	—	—	C
O	273 = OMP	262	272 = spaceCO	O
M	—	—	—	M
P	274 = PRE	264	273 = OMP	P
R	—	—	—	R
E	275 = ESS	266	274 = PRE	E
S	—	—	—	S
S	276 = SE	S	275 = ESS	S
E	277 = ESspace	266	276 = SE	E
S	—	—	—	S
space	278 = spaceD	space	277 = ESspace	space
D	279 = DAT	256	278 = spaceD	D
A	—	—	—	A
T	280 = TAspace	258	279 = DAT	T
A	—	—	—	A
space		space	280 = TAspace	space

2.2 Lossy Compression

In addition to the tradeoff between coding efficiency – coder complexity – coding delay the additional aspect of compression quality arises with the use of lossy methods. Quality is hard to assess since for many application human perception is the only relevant criterion. However, there are some scenarios where other factors need to be considered, e.g. when compressed data is used in matching procedures like in biometrics. Although several quality measures have been proposed over the last years, PSNR is still the most used measure to determine image and video quality after compression has been applied:

$$RMSE = \sqrt{1/N \sum_{i=1}^{N} (f(i) - f'(i))^2}$$

where $f'(i)$ is the reconstructed signal after decompression. Peak-signal-to-noise ratio (measured in deciBel dB) is defined as follows (where MAX is the maximal possible value in the original signal $f(i)$):

$$PSNR = 20 \log_{10}(\frac{MAX}{RMSE}).$$

The higher the value in terms of PSNR, the better is the quality of the reconstructed signal. With increasing compression ratio, PSNR decreases. While PSNR is not well correlated with moderate quality changes, it serves its purpose as a rough quality guideline. Many other numerical measures have been developed with better prediction of human perception (e.g. structured similarity index SSIM) but these have not been widely adopted One reason is that image compression can be easily optimized with respect to RMSE (rate / distortion optimisation), but not with respect to the "better" measures. In order to really take human perception into account, subjective testing involving a significant number of human subjects is necessary – the resulting subjective measure is called MOS ("mean opinion score"). When developing quality measures, a high correlation with MOS computed over large databased is desired. PSNR does not correlate well with MOS.

2.2.1 Quantisation

When changing the description of a signal from using a large alphabeth to using a smaller alphabeth of symbols we apply quantisation. Obviously, when reversing the process, a perfect reconstuction is no longer possible since two or more symbols of the larger alphabeth are mapped to a single symbol of the small alphabeth. Often, quantisation errors are the only actual errors caused by lossy compression (apart from rounding errors in transforms).

Example: suppose we have stored a grayscale image with 8bit/pixel (bpp) precision, which means that for each pixel, we may choose among 256 different grayscales. If we apply a quantisation to 4 bpp, only 16 grayscale can be used for a pixel. Of course, it is no longer possible to re-transform the 4 bpp into the original bit-depth without errors. The information is lost.

If single signal values are subject to quantisation one-by-one (a single value is quantised into its less accurate version), the procedure is denoted "scalar quantisation". If we subject n-tuples of signal values together to quantisation (a vector is replaced by a less accurate version), we call the scheme "vector quantisation". In any case, quantisation may be interpreted as a specific kind of approximation to the data.

If we devide the range of the original signal into n equally sized and disjoint intervals and map all symbols contained in an interval to one of the n symbols of the new alphabeth, the quantisation is called "uniform". Especially in the case of uniformly distributed probabilities of symbol occurrence uniform quantisation makes sense. Quantisation is most often applied to transform coefficients (e.g. after DCT or wavelet transform) – in this context, coefficients quantised to zero are desired to appear frequently. Therefore, "uniform quantisation with deadzone around 0" is introduced, we uniform quantisation is applied, except for the quantisation bin around zero, which has twice the size of the other bins (symetrically around zero).

In case the occurrence probabilities of the symbols in the original signal are not uniformly distributed and the distribution is known, it makes sense to apply finer quantisation (i.e. smaller bins) in the areas of the histogram where more data values are accumulated. In this case we talk about "non- uniform quantisation". Note that following this strategy we minimise PSNR (the error is minimized in areas where many values are found) but we do not care at all about perceptual issues.

If the distribution of the occurrence probabilities of the symbols is not known, the signal can be analysed and based on the results, the quantisation bins can be set adaptively – termed "adaptive quantisation". Since especially in the case of transform coding in many cases the probabilities of symbol occurrence is quite stable for typical signals, quatisation can be modeled in advance and does not have to be adapted to signal characteristics.

2.2.2 DPCM

Differential pulse code modulation (DPCM) is the general principle behind differential / predictive coding. Instead of considering the simple difference between adjacent signal values, also the difference between a signal value to be predicted and a linear-combination of surrounding values (i.e. the prediction) can be considered.

Example: $y_i = x_i - x'_i$, where $x'_i = \alpha x_{i-1} + \beta x_{i-2} + \gamma x_{i-3}$. The values α, β, γ are denoted "predictor coefficients", x'_i is the prediction for x_i. The example is a *linear* predictor of *order* 3. Instead of storing or processing the sequence x_i, the sequence of differences y_i is considered, which has a significantly lower entropy (and can therefore be better compressed).

The predictor coefficients can be fixed for classes of known signals (e.g. exploiting specific patterns like ridges in fingerprint images) or can be adapted for each single signal (in the latter case, these need to be stored). The optimal values for α, β, γ e.g. minimising entropy for y_i can be determined of course. Furthermore, for each distinct part of a signal dedicated coefficients can be used and optimsied ("local adaptive prediction"). By analogy, these coefficients need to be stored for each signal part impacting on compression ratio (obviously, there is a trade-off !).

The decision if DPCM is lossy or lossless is taken by the way how y_i is compressed. In case e.g. Huffman coding is applied to y_i we result in an overall lossless procedure. Is y_i subject to quantisation, the overall scheme is a lossy one.

A variant of DPCM frequently employed is to use a fixed set of predictor and quantiser combinations which are applied tp the data depending on local signal statistics. Lossless JPEG is such a scheme (without applying quantisation of course).

2.2.3 Transform Coding

Most standardised lossy media compression schemes are based on transform coding techniques. Typically, tranform based coding consists of three stages.

1. Transformation
2. Quantisation: a variety of techniques (as discussed before) can be applied to map the transform coefficients to a small alphabeth, the size of the latter being selected corresponding to the target bitrate. Most rate-control procedures are applied in this stage.
3. Lossless encoding of coefficients: a variety of techniques (as discussed before) is applied in order to encode the quantised transform coefficients close to the entropy bound. Typically, a combination of runlength (zero-runs !) and Huffman coding (for the older standards like JPEG, MPEG-1,2,4) or arithmetic coding (for the more recent standards like JPEG2000 H.264) is used.

The Transform

The aim of transformation is to change the data of a signal in a way that it is better suited for subsequent compression. In most cases, so-called "integral-transforms" are used, which are based (in their contineous form) on applying integral operators. When applying transforms, the concept of projecting the data onto orthogonal basis functions is used.

Motivation: Vectors in two-dimensional space can be represented by a set of othogonal basis-vectors ("orthogonal basis", orthogonal requires the inner product between vectors to be 0).

$$(x, y) = \alpha(1, 0) + \beta(0, 1).$$

$\{(1, 0), (0, 1)\}$ are orthogonal basis-vectors, α und β are called "coefficients" which indicate a weight for each basis-vector in order to be able to represent the vector (x, y). The property of orthogonality guarantees that a minimal number of basis-vectors suffices (obviously, this will be important for compression).

This concept can be transferred by analogy to represent functions or signals by means of basis elements.

$$f(x) = \sum_n < f(x), \psi_n(x) > \psi_n(x)$$

$\psi_n(x)$ are orthogonal basis-functions, $< f(x), \psi_n(x) >$ are transform coefficients which determine the weight to be applied to each basis-function, to represent a given signal. When performing the actual compression, the $< f(x), \psi_n(x) >$ are computed, quantised, and stored in encoded form. Since $\psi_n(x)$ are chosen to be an orthogonal function system, the corresonding number of coefficients to be stored is minimal.

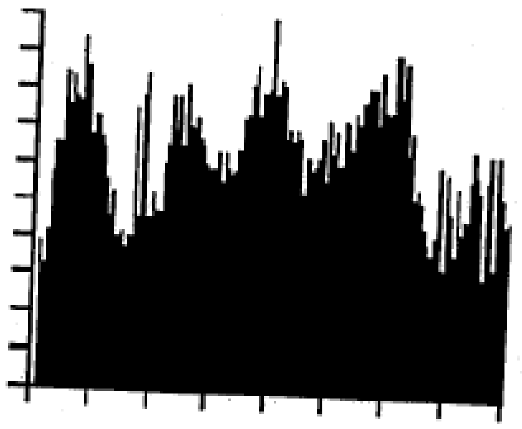
Example: in the case of the Fourier transform $\psi_n(x) = e^{-\pi inx} = \cos(nx) - i\sin(nx)$. In this special case we observe the frequency of periodic signals. A Fourier coefficient $< f(x), \psi_n(x) >$ reports the strength of the frequency component n in the signal which has been subjected to the transform. Obviously, not all functions can be represented well by stationary (i.e. frequency does not change over time) periodic basis functions. This is where the wavelet transform and other types enter the scene.

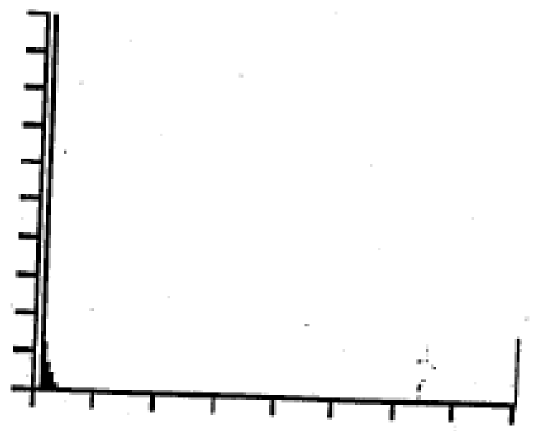
To conduct transform coding, we represent the signal $f(x)$ by $f(x) = \sum_n < f(x), \psi_n(x) > \psi_n(x)$ (i.e. the transform coefficients are computed). The transform coefficients $< f(x), \psi_n(x) >$ are quantised subsequently and finally compressed losslessly. In case $\psi_n(x)$ are well chosen, the transform coefficients are 0 after quantization for most basis functions and do not need to be stored. Consequently, in the reconstruction the error is small, since most coefficients were 0 anyway. Most of the signal information is concentrated in a small number of coefficients – we speak of energy compactation here. Therefore, the art of transform coding is to select an appropriate set of suited basis functions that works well for a wide class of signals.

A further way to interpret integral transforms is to view them as a rotation of the coordinate axes. The data is transformed into a coordinate system, which allows for a more efficient compression. As a motivation for this interpretation, we consider pairs $X = (x_1, x_2)$ which are gernerated from neighboring pixel values in an image. Plotting all pairs into a coordinate system most of the pairs are positioned around the main diagonal (since most neighboring pixels tend to be similar). The variance of x_j is defined as

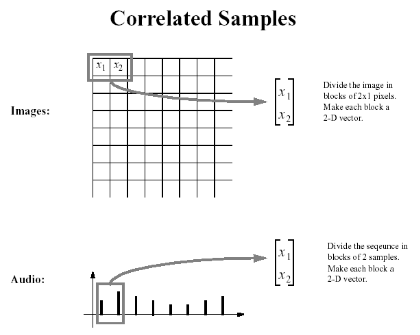
$$\sigma^2_{x_j} = 1/M \sum_{i=1}^{M} (x_{ji} - \mathbf{x_j})^2$$

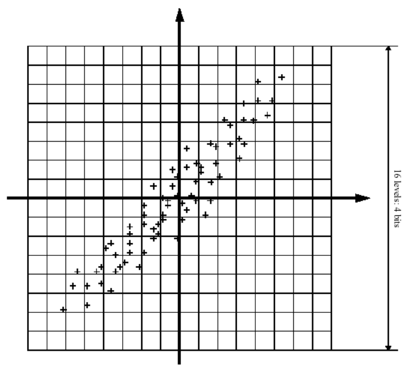
where M is the number of pairs and $\mathbf{x_j}$ is the mean of x_j computed over all pairs. A simple technique to apply compression is to replace one of the two components in each pair by the corresponding mean $\mathbf{x_j}$. The

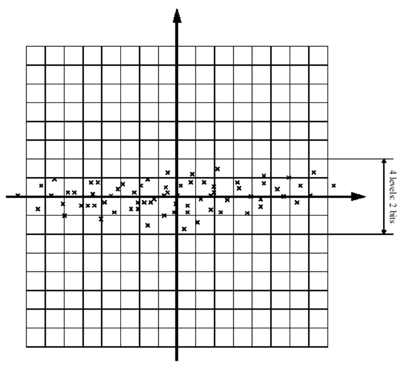




MSE of this compression approach is exactly $\sigma^2_{x_j}$, the variance. No matter which coordinate we replace be the mean, the variance of both cases is large and therefore, the error of this compression approach is large.


Correlated Samples
Images:
x_1 x_2
Divide the image in blocks of 2x1 pixels. Make each block a 2-D vector.
Audio:
Divide the seqeunce in blocks of 2 samples. Make each block a 2-D vector.





Let us consider the transformation of X to $Y = (y_1, y_2)$ by applying a rotation of the coordinate axes by 45 degrees (Y = AX where A is a rotation matrix). A fundamental property of these types of transforms is that they preserve the variance, i.e.

$$\sigma^2_{x_1} + \sigma^2_{x_2} = \sigma^2_{y_1} + \sigma^2_{y_2}$$

While variances in the original coordinate system have been almost equally for x_1 and x_2, in the new coordinate system variance is distribute quite unevenly, i.e. $\sigma^2_{y_1} >> \sigma^2_{y_2}$. Therefore, in this representation it is obvious to apply our compression scheme to y_2 (by replacing its components by the mean $\mathbf{y_2}$) since the variance and consequently the error will be small.

Examples of popular transforms (desired properties are: decorrelation of data, energy compactation, basis functions not dependent on data, fast algorithms):

1. Karhunen-Loeve transform: optimal transform since basis functions are computed depending on the data. A disadvantage is the high complexity $O(N^3)$ and the fact that the basis functions need to be stored and cannot be used universally. This transform is used to decorrelate "multiple-spectral band imagery" with a larger number of bands. Closely related to PCA !

2. Discrete Fourier transform (DFT): “the” classical transform, however, due to its complex output and periodicity not well suited for compression. Mostly employed in audio coding for signal analysis. $F(u) = 1/n \sum_{j=0}^{n-1} f(j) e^{\frac{2\pi iuj}{n}}$.

3. Discrete cosine transform (DCT): applied in image, video, and audio coding where it is applied to chunks of data and not to the entire signal. $F(u) = 1/n \sum_{j=0}^{n-1} f(j) \cos \frac{(2j+1)u\pi}{2n}$.

4. Wavelet and subband transform: uses basis functions with 2 parameters instead of the single frequency parameter of the DFT and DCT: a sort of scale and time / position. More details to come in the chapter on lossy image compression.

Quantisation

A fundamental issue for quantisation of transform coefficients is the strategy how coefficients are selected to be processed further:

1. According to pre-defined zones in the transform domain: e.g., high frquency coefficients are severely quantised and low frequency components are protected (HVS modelling).

2. According to magnitude: coefficients smaller than some threshold are ignored (disadvantage: location needs to be store somehow), or only L largest coefficients are kept in quantised form, etc.

In fact, in most schemes both ideas are combined (e.g. JPEG).

Entropy Coding

Most schemes use either a combination of Huffman and Runlegth Coding (mostly older standards like JPEG, MPEG-1,2,4) or arithmetic coding (more recent schemes like JPEG2000, H.264 / CABAC)

2.2.4 Codebook-based Compression

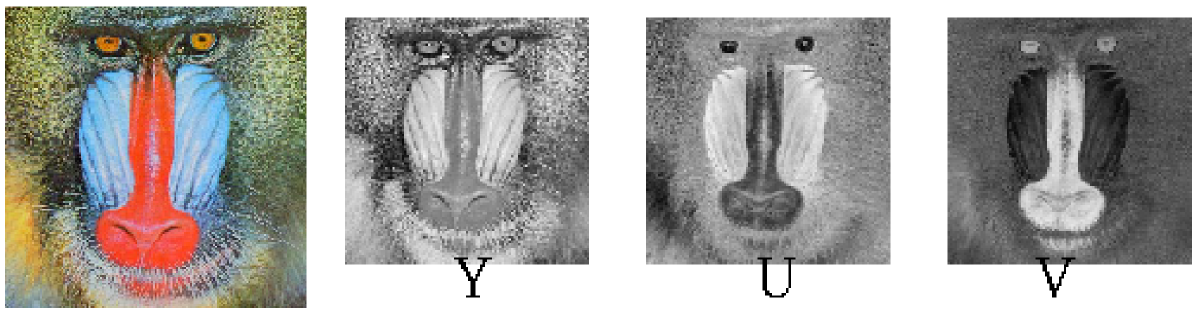
Similar to the lossless case, we employ a dictionary to represent parts of the signal by entries in the dictionary. I.e. select a part of the signal of size n and replace it by the entry in the dictionary with has the same size and is most similar to it. Of course, also here arises the question how to determine the measure of similarity or quality. Instead of the signal part we only store the index of the entry in the dictionary. Encoder and Decoder need to be synchronised with respect to the dictionary entries.

Important techniques of this class are vector quantisation (applied directly in the signal domain and not to transform coefficients) and fractal compression.

Kapitel 3

Image Compression

We consider images to be represented as two-dimensional arrays which contain intesity / luminance values as their respective array elements. In case of grayscale images having n bit of accuracy per pixel, we are able to encode 2^n different grayscales. A binary image therefore requires only 1 bpp (bpp = bit per pixel). Colour images are represented in different ways. The most classical way is the RGB representation where each colour channel (red, green, blue) is encoded in a separate array containing the respective colour intensity values. A typical colour image with 24 bpp consists therefore of 3 8bpp colour channels. The human eye is not equally sensitive to all three colours – we perceive green best, followed by red, blue is perceived worst. Based on this fact (and some other more technical reasons) an alternative colour scheme has been developed: YUV. Y is a luminance channel, U and V are the colour or chrominance channels. Y is obtained by a suited weighting of RGB (i.e. Y = 0.3 R + 0.5 G + 0.2 B), by analogy U and V are computed as difference signals – both ways to represent colour images can be converted into each other. An advantage of the YUV representation is the easy availability of a grayscale variant of the data and the immediate possibility of compress colour information more severely.



An entirely different way to represent colour are "palette" images: colour information is encoded by indices which are defined in the image header (i.e. only the number of colours actually present in the image needs to be encoded thus reducing the number of required indices and thus, storage space). On the other hand, in this representation close indices do not necessarily correspond to similar colours which can impact on prediction techniques for example. Therefore, it is imperative that colour representation and applied compression technique match.

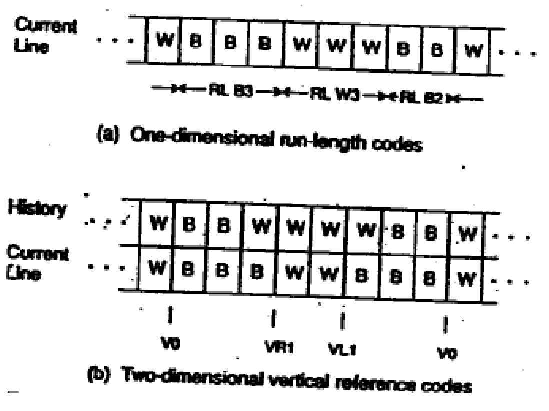
3.1 Lossless Image Compression

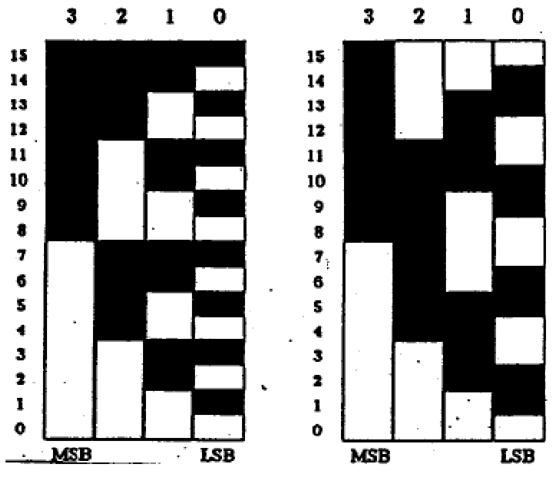
When discussing lossless compression in general it aleady got clear that the range of compression ratios that can be achieved in lossless coding is limited. Therefore, in image compression, lossless techniques are applied only in case this is required either by law (medical images in many countries) or by the application (e.g. the storage of geographic maps or texts as images). Limited computational resources can be another reason for the employment of lossless techniques.

3.1.1 Fax Compression

In this area the exist the common standards ITU-T T.4 and T.6 for group 3 and 4 fax machines (which are basically standards for lossless compression of binary images). Compression ratios achieved can range up to 15 and higher (however, it is important to note that this is conly the case for text documents, which are structured entirely different as compared to grayscale images after halftoning). All these standards employ runlength and Huffman coding. One-dimensional T.4 coding is only focused onto a single line of pixels – first, the runs are determined, subsequently the out output is encoded using a fixed (modified) Huffman code. Each line is terminated with an EOL symbol in order to allow resynchronisation after transmission errors.

In order to incrase compression efficiency, a two-dimensional version has been developed: A limited number of lines is encoded like described before, the remaining lines are encoded relative to the lines coded in one-dimensional manner. An additional bit after the EOL symbol indicates if the next line is encoded in 1-D or 2-D manner. Resynchronisation after transmission errors is only possible based on lines encoded in 1-D manner. ITU-T T.6 is meant for environments with guaranteed lossless transmission – there are no EOL symbols and encoding is only done in 2-D manner. This strategy maximises compression ratio at the price of the lost error robustness.

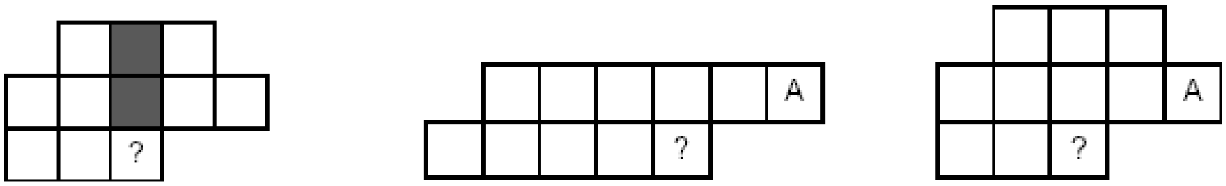




3.1.2 JBIG-1

Joint Binary Image Experts Group was supported by ITU-T and ISO. JBIG is a standard for compressing binary images, but not restricted to text documents (fax) – instead the focus also inlcuded halftoned images. ITU-T terminology is ITI-T T.82 recommendation. The coding engine is an arithmetic coder (IBM Q-Coder) which uses contexts spanning over three lines around the actual pixel position. The idea of using "context-based coding" is to consider the neighbourhood of a pixel to determine the actual probability of occurrance. In case a certain part of a letter has been already encoded, the remaining part can be compared with signal parts that have already been encoded and the pixel value to be expected can be predicted with high accuracy and probability (which leads to a short description as we already know. For example, a 0 surrounded by other 0s is much more likely to occur as compared to a 0 surrounded by 1s. Another example is shown in the figure – the pixel to be encoded has a high probalility of being black (could be the letter I or T or whatever).

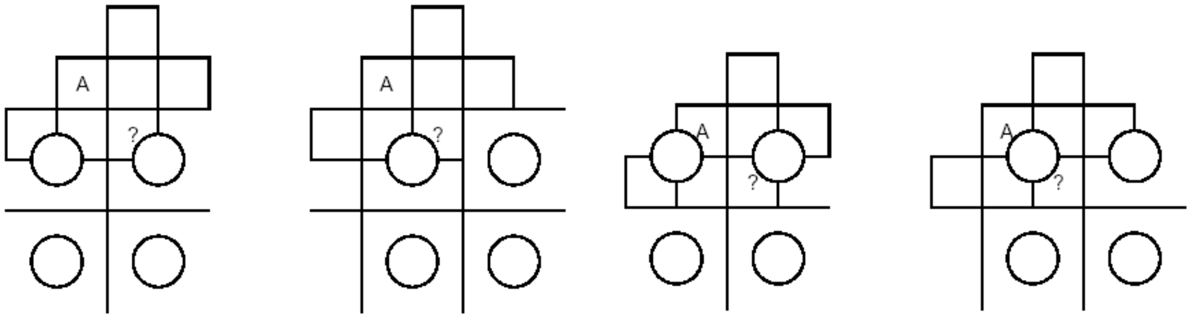
The pixel with the question-mark is to be encoded; the probalility of occurrence is estimated based on 10 already encoded pixels n the vincinity using the pattern as shown. This means that $2^{10} = 1024$ contexts need to be considered – in fact, the statistical for all of these contexts are contineously updated and the partitioning of the interval in arithmetic coding is done accordingly.



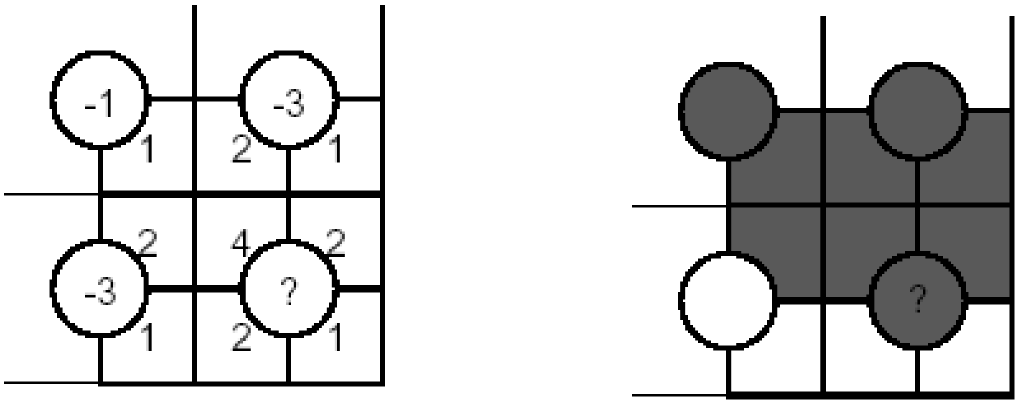
In fact two different context patterns can be used: the first spans across two lines ony (which allows more efficient software implementation), while the second uses a three line context (which gives about 5% better compression efficiency). The choice of the less popular first pattern is encoded with `LRLTWO` in the header. The pixel labeled "A" is an adaptive part of the pattern which can be freely selected in a large window around the pixel to be encoded – this can be beneficial especially in halftoned areas with repetitive patterns when the distance of A corresponds to the period of the pattern.

JBIG supports hierarchical encoding (see below) – in all but the lowest resolution also information of a lower resolution is included in the context of a pixel value to be encoded. In the figure, circles depict pixels of such a lower resolution while the squares are the pixels of the resolution we currently encode. The context contains 4 lower resolution pixels and 6 current resolution pixels which can have four relative positions as shown in the figure. This leads to $4 \times 2^{10} = 4096$ contexts.

In order to facilitate hierarchical encoding, starting from the highest resolution, a set of images with successively lower resolution is generated. This could be done by simple "downsampling by two" in each direction: every other pixel is deleted in horizontal and vertical direction, respectively. This is a bad idea in general: for example, in case horizontal or vertical lines are represented single pixel wide, they can be erased by this operation. Moreover, in case halftoning achieves a medium grayscale with alternating black and white pixels, downsampling could eventually erase only black pixel causing a significant increase in average luminance. Therefore, a linear recurvive IIR (infinite impulse response) filter is used which employs a 3×3 window in the current resolution and three additional pixels of the lower resolution (aleady generated)



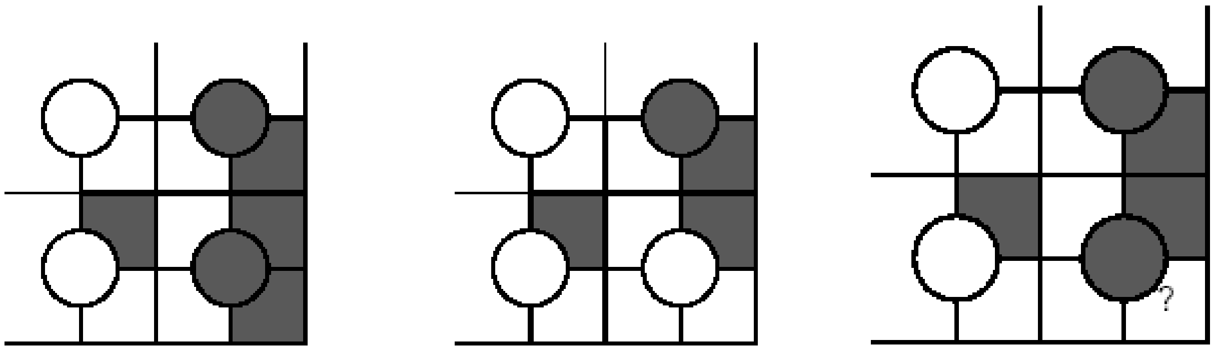
to generate the next pixel. The twelve pixel values are multiplied by the depicted values in the filter and the products are summed up. In case the result is 5 or larger, the pixel is set to 1. However, it still may happen that the filter erases edges (as shown in the figure). This is one of several (over 500) exceptions to the filtering procedure which is explicitely coded. Since all $2^{12} = 4096$ filter responses are held in a table anyway, this does not cause additional complications.



As explained, for all but the lowest resolution the information of the next lower resolution is used for encoding the current resolution. In certain situations the value of the pixel to be encoded can be deduced from the context *without* actually coding the information – this is based on knowledge about the resolution reduction algorithm. This technique is called "deterministic prediction" and the according symbols are inserted again after arithmetic encoding.

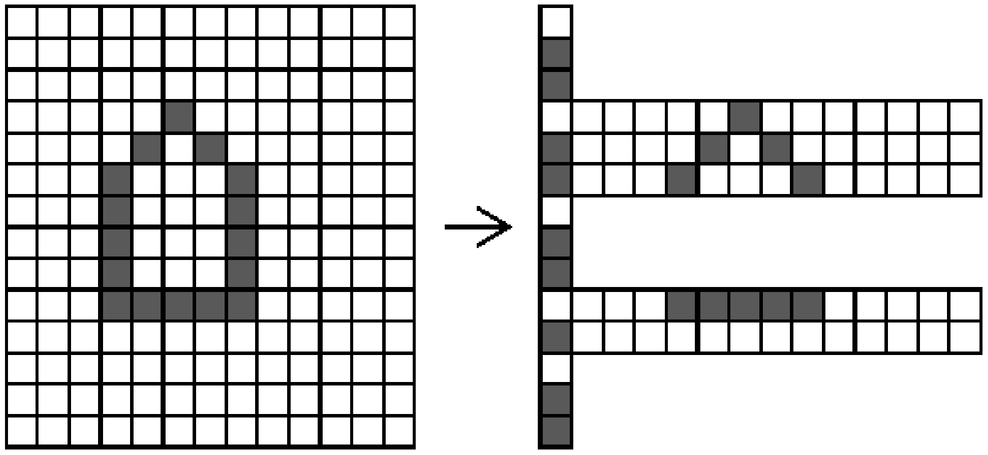
Example: In case resolution reduction is performed by skipping all even-numbered lines and columns, in the high resolution image all pixels with even-numbered line- and column-number share the same value as their counterparts in the lower resolution. Therefore, in the higher resolution 25% of the data would not need to be encoded. However, the actual technique is much more complicated, still the basic idea may be applied. In the figure is shown that the flipping of the bit in the lower-right pixel changes the value of the pixel in the lower resolution. Therefore it is clear that the pixel depicted with "?" needs to be a 1.

The actual procedure consists of a 6912 entries table with contains all context situations (contexts with 8, 9, 11, or 12 neighbours) which allow to deterministically predict a pixel and the corresponding prediction values. In case the standard resolution reduction scheme is used, the table does not need to be encoded, in case a different scheme is employed, the table is communicated in the header.

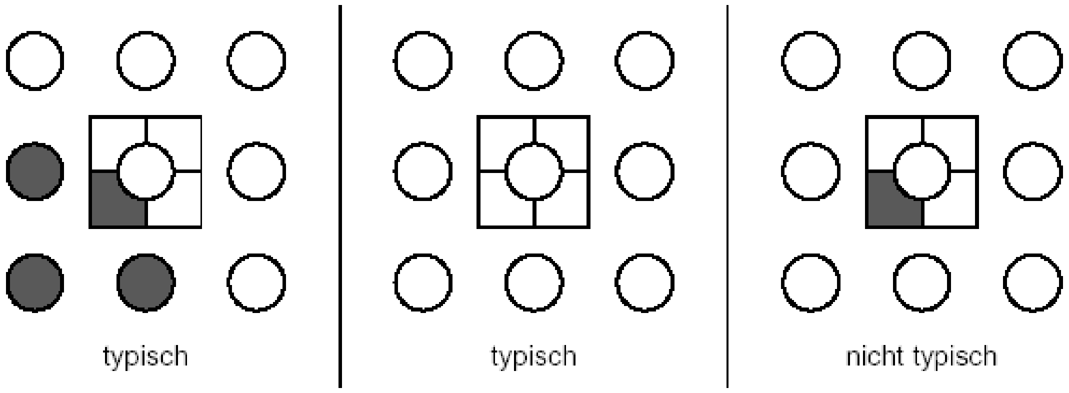


An additional mechanism termed “typical prediction” is applied even before deterministic prediction and removes pixels which can be encoded more efficiently as by feeding them into the arithmetic encoder. The strategies used differ whether they are employed in the lowerst resolution or in the higher resolutions.

In the lowest resolution, typical prediction is used to encode successive identical lines only once. To facilitate this so-called “pseudo-pixels” are inserted at the start of each line, if the content is identical to the preceeding line (which is a typical line in this terminology) this value is set to 0 and the data does not have to be encoded. In fact, the change between typical and non-typical lines is encoded instead of the actual value.



In the higher resolutions, again the relation to the next lower resolution is exploited. A pixel is termed typical, if the fact that the pixel itself and its 8 neighbours share the same colour implies that the corresponding four pixels at the higher resolution also share the same colour. The figure shows two typical pixels and a non-typical pixel. An entire line is termed typical if it consists of typical pixels only. Again a pseudo-pixel is introduced: if the line at lower resolution for a pair of lines at higher resolution is typical (i.e. their values can be deduced from the lower resolution), this is indicated by this value. Based on this information (and the values of the lower resolution), the pixel value can be reconstructed.



Taken these ideas together results in a higher compression ratio as compared to ITU-T T.4 and T.6 – an average 5.2 and 48 versus 1.5 and 33 for halftoned material and documents, respectively. On the other hand, computational demand increases by a factor of 2-3 as compared to ITU-T T.4 and T.6.

3.1.3 JBIG-2

JBIG-2 has been standardised in 2000 and its design was mainly motivated to have a lossless to *lossy* technique for binary image coding and to improve lossless performance of JBIG-1. JBIG-2 is designed to allow for *quality-progressive* coding (progression from lower to higher quality) as well as *content-progressive* coding, i.e. successively adding different types of image content.

Especially the latter form of progressiveness is a very interesting and innovative one. A typical JBIG-2 encoder segments a given binary image into several regions or image segements (based on content properties) and subsequently, each segement is separately coded with a different compression technique.

JBIG2 files consist of different types of data segments: image data segements are those that describe how the page should appear. Dictionary segments describe the elements that make up the page, like the bitmaps of text characters. Control data segments contain techical information like page striping, coding tables etc. Each segment is preceeded by a header indicating its type. In sequential mode, the headers are located directly in front of thier corresponding data, in random access mode the headers are gathered at the beginning of the file, allowing the decoder to acquire knowlegde about the documents overall structure first.

JBIG2 essentially distinguishes three types of image data segements: textual data, halftone data, and data which can not be classified into one of the two former classes. The image data segements of the latter class are encoded with a procedure very similar to JBIG-1 since no prior information can be exploited. However, the encoder distinguishes between the former two types of data.

Textual Data

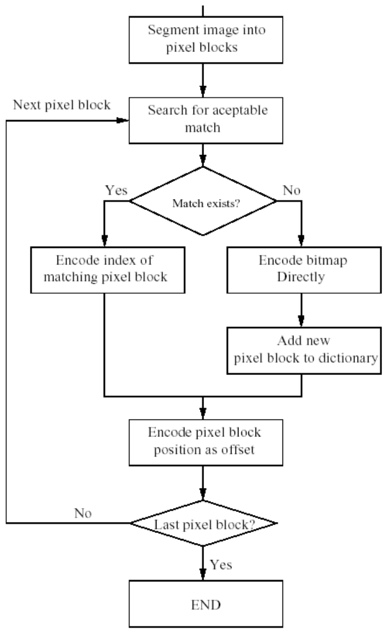
The basic idea is that text typically consists of many repeated characters. Therefore, instead of coding all the pixels of each character occurrence separately, the bitmap of a character is encoded and put into a dictionary (usually termed "pixel block"). There are two different ways how to encode pixel blocks in

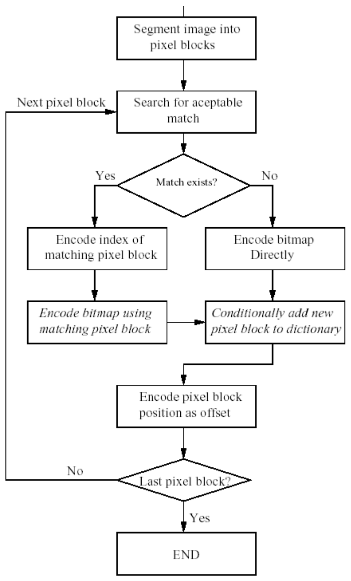
JBIG-2 (indicated in the corresponding header data).

Pattern matching and substitution: in a scanned image, two instances of the same characted do not match pixel for pixel, but they are close enough for a human observer such that they can be identified to be the same. Therefore, the idea is not to encode the pixel information but only a pointer into a dictionary of pixel blocks consisting of character bitmaps.

1. segementation into pixel blocks (contigeous areas of connected black pixels are enclosed by a rectangular bounding box, features for each block are extracted like height, width, area)
2. searching for a match in the dictionary (screening based on features, numerical score like Hamming distance)
3. coding of the pointer in case there is a good match
4. coding of the bitmap into the dictionary otherwise (like JBIG-1)

Obviously, this technique will produce errors in general. In applications where this is not acceptable (i.e. lossless encoding), residue encoding is applied: the encoder computes the difference between original and so-far encoded material which is additionally encoded similar to JBIG-1.


Segment image into pixel blocks
Next pixel block
Search for aceptable match
Match exists?
Yes
No
Encode index of matching pixel block
Encode bitmap Directly
Add new pixel block to dictionary
Encode pixel block position as offset
No
Last pixel block?
Yes
END

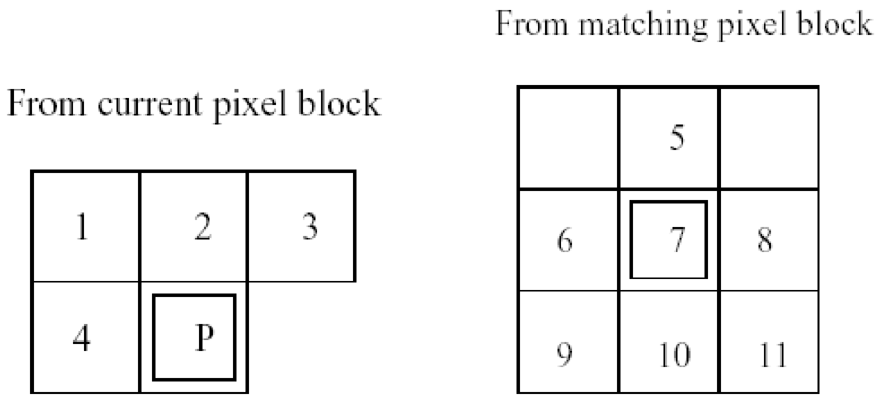

Segment image into pixel blocks
Next pixel block
Search for aceptable match
Match exists?
Yes
No
Encode index of matching pixel block
Encode bitmap Directly
Encode bitmap using matching pixel block
Conditionally add new pixel block to dictionary
Encode pixel block position as offset
No
Last pixel block?
Yes
END

Soft pattern matching: Overall, the technique seems to be fairly similar at first sight, however, there is a significant difference how the matches in the dictionary are used. While they are used for explicit replacement in the former approach, here they are used to acquire detailed context information for better arithmentic encoding.

1. segementation into pixel blocks (contigeous areas of connected black pixels are enclosed by a rectangular bounding box, features for each block are extracted like height, width, area)

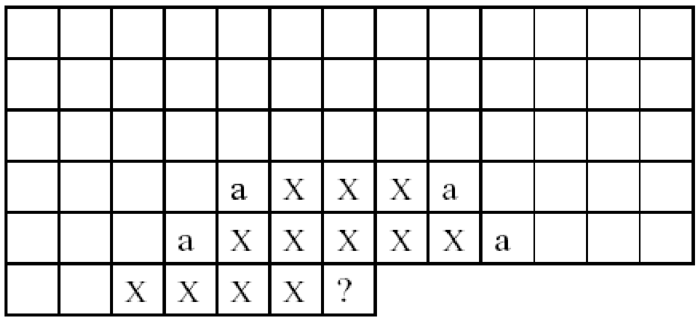
2. searching for a match in the dictionary (screening based on features, numerical score like Hamming distance)
3. coding of the pointer in case there is a good match
4. coding of the bitmap into the dictionary otherwise (like JBIG-1)
5. coding of the bitmap of the current block (the centers of the two matching blocks are aligned and the pixels are arithmetically encoded using the two contexts shown in the figure below).

As a result, a poor match does not result in an error as it occurs in the first approach but only in a less efficient encoding due to incorrect context information.



Halftone data

There are also two ways how to encode halftone data. The first approach uses a classical arithmetic coding technique close to JBIG-1, however, the context used is larger (up to 16 pixels) and may include up to 4 adaptive pixels as shown in the figure. This is important to capture regularities in the halftoning patterns better.



The second approach relies on descreening the halftone image (converting it back to grayscale) using an image tiling (where the grayscale value can be e.g. the sum of the set pixels in the block). The grayscale image is represented in Gray code (see below) and the bitplanes are arithmetically encoded similar to JBIG-1. The grayscale values are used as indices of fixed-size bitmap patterns in a halftone bitmap directory. Therefore the decoder applies simple lookup-table operations.

Apart from the loss already potentially introduces by some of the techniques, additional techniques to reduce the amount of data in a lossy manner may be employed: quantization of pixel-block position information, noise removal and smoothing, and explicit bit-flipping for entropy reduction.

3.1.4 Compression of Grayscale Images with JBIG

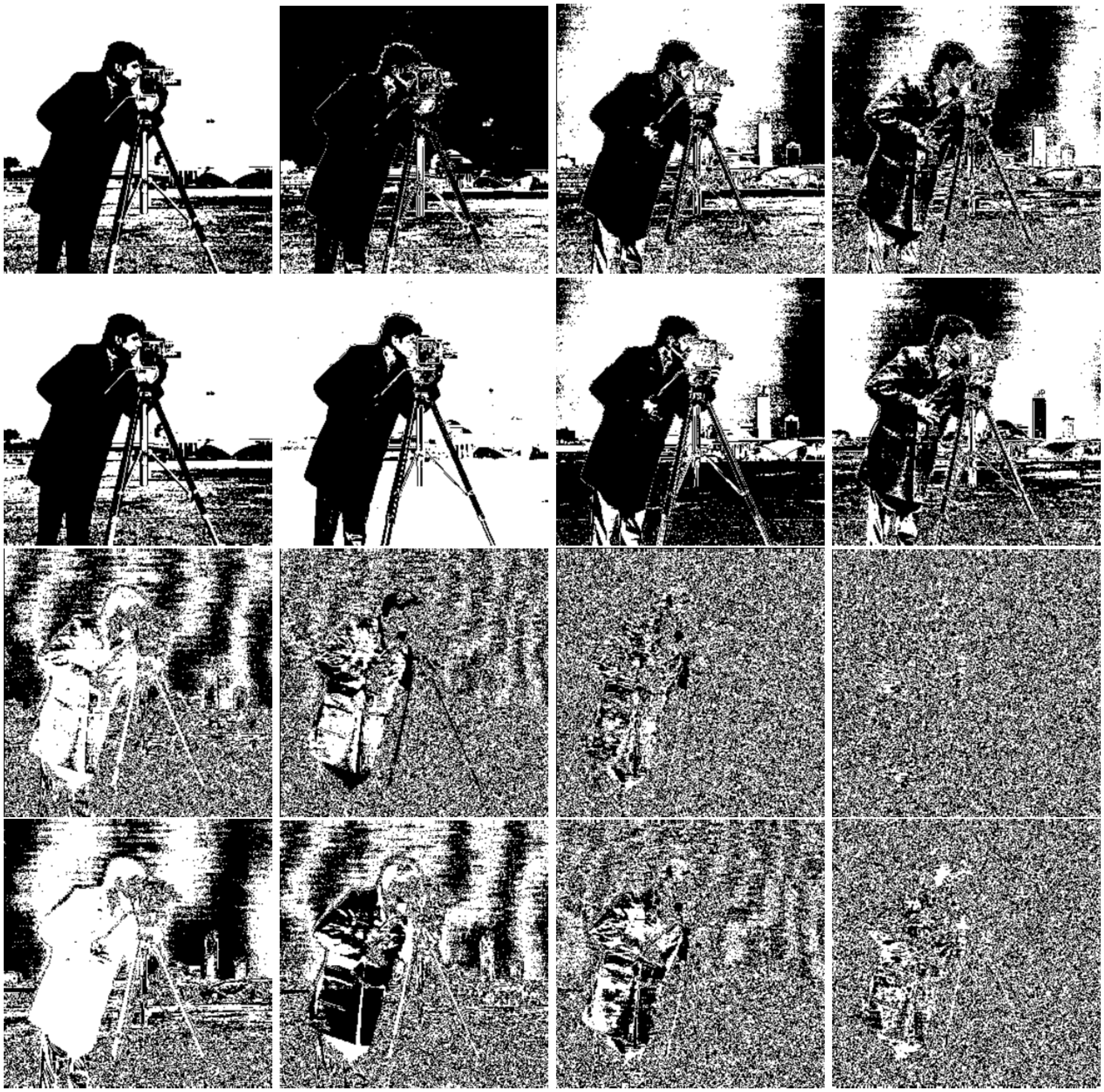
Like any other technique for compressing binary images, JBIG can be used to compress grayscale data. This is accomplished by successive coding of single bitplanes (which are binary images of course), e.g. we start with fusing the MSBs of all pixels into a bitplane, compress it, and carry on until we have compressed the bitplane containing all LSBs. All typical compression systems are able to take advantage of large aras of uniform colour. Therefore, the bitplanes should preferably like this. However, the classical binary representation leads to the opposite effect. Consider large areas in the original image where the grayscale changes from 127 to 128. In binary representation this corresponds to 01111111 and 10000000 which means that a change from 127 to 128 causes a bitchange in each bitplane which destroys large nearly uniform areas found in the original image in the bitplanes. This problem can be overcome by employing the *Gray Code*. In Gray code representation, two neighbouring decimal number only differ in a single bit in their respective binary representation. This is exactly what we want.

Binary Code $\longrightarrow$ Gray Code:

1. Starting with the MSB of the original binary representation, all 0 are kept until we hit a 1.
2. The 1 is kept, all following bits are inverted until we hit a 0.
3. The 0 is inverted, all following bits are kept, until we hit a 1.
4. Goto Step 2.

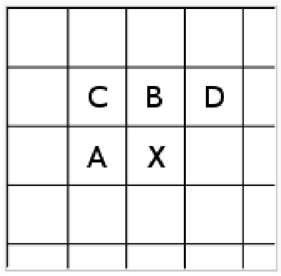
Gray Code $\longrightarrow$ Binary Code:

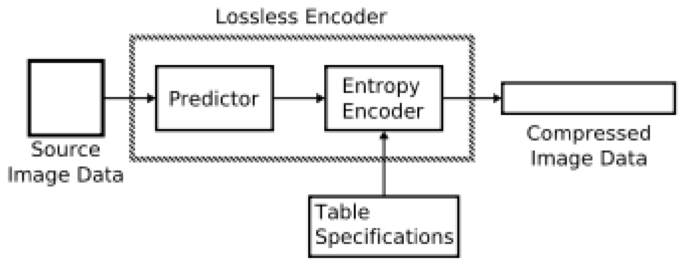
1. Starting with the MSB of the Gra code representation, all 0 are kept until we hit a 1.
2. The 1 is kept, all following bits are inverted until we hit a 1.
3. The 1 is inverted, all following bits are kept, until we hit a 1.
4. Goto Step 2.

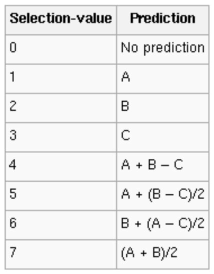


3.1.5 Lossless JPEG

Up to 6bpp we get decent results with successive JBIG encoding, for higher bit-depth other techniques should be used which are better able to exploit the inter-bitplane dependencies. The simplest choice for this application is the standard for lossless image compression developed by the Joint Photographic Experts Group (JPEG): lossless JPEG is a classical DPCM coder (and has nothing to do with the JPEG format we all know from digital cameras !). Based on a prediction for the actual pixel that should be encoded (involving neighbourhoods of that pixel), a residual is computed with respect to the real pixel value (exhibiting low entropy). The residual is compressed using a fixed Huffman code (from predifined tables which are part of the encoder and decoder). One can choose among 7 different predictor types, however, the standard does not contain an algorithm how this should be done. In non-standard compliant extension the best predictor is chosen automatically for the entire image, for an image block, or contineously. The residual data is represented as symbol pairs: category and size, only the former is Huffman encoded.





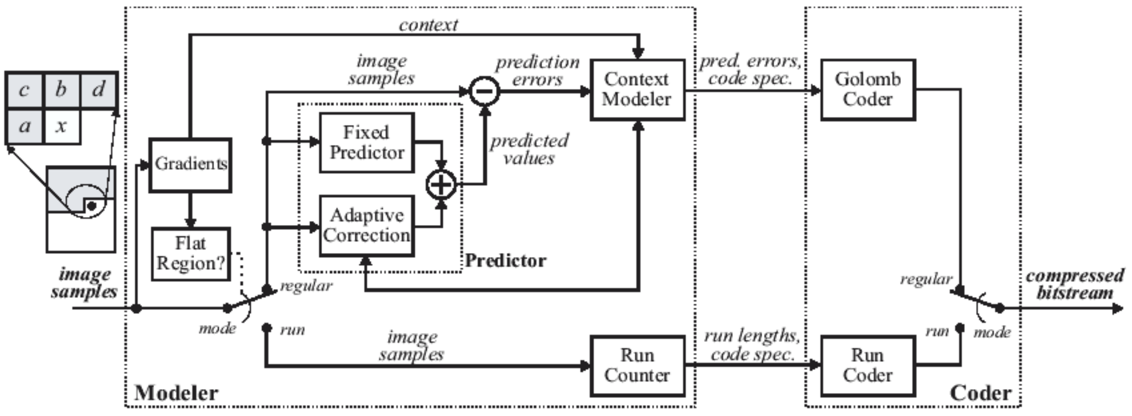


Selection-value	Prediction
0	No prediction
1	A
2	B
3	C
4	A + B – C
5	A + (B – C)/2
6	B + (A – C)/2
7	(A + B)/2

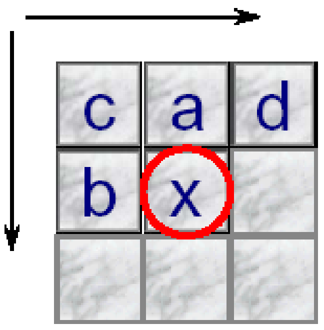
Lossless was used in medical imaging (DICOM standard) but not in many more areas due to its rather poor compression performance.

3.1.6 JPEG-LS

JPEG-LS is based on an IBM development called LOCO-I and is significantly superior to its predecessor lossless JPEG. It is not the lossless image coding algorithm achieving the highest compression ratios (which is achieved by the CALIC-type coders relying on arithmetic coding) but certainly the one with the best compromise between compression ratio and computational efficiency. Similar to lossless JPEG it is based on predictive coding, however, JPEG-LS distinguishes between two operation modes, the "regular mode" and the "run mode".



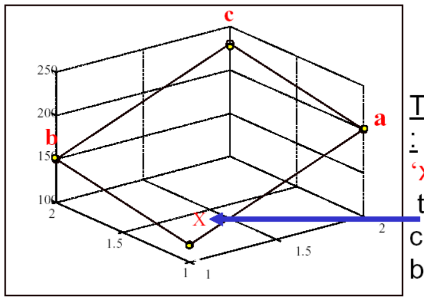
The run mode is activated in case run length encoding is found to be beneficial (where run length coding is performed without computing residuals). The regular mode employs an adaptive predictor based on computing simple edge characteristics in the neighbourhood of the pixel to be predicted.



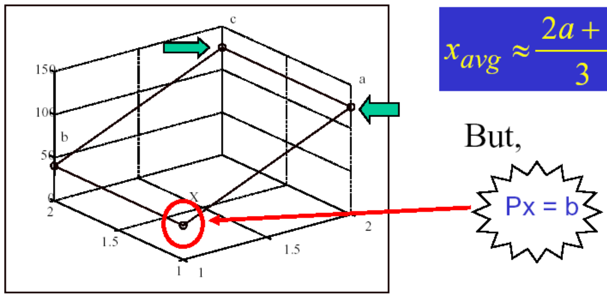
$$Px = \begin{cases} \min(a,b) & c \geq \max(a,b) \\ \max(a,b) & c \leq \min(a,b) \\ a+b-c & otherwise \end{cases}$$

In case a vertical edge is detected left to X, this predictor tends to predict a, in case a horizontal edge is found above X, the predictor tends to predict b, in case no edge is detected a plane-focused prediction is done. To see this, let us assume $a \leq b$: either (1) $c \geq b$ means a vertical edge and $min(a,b) = a$ is predicted, in case (2) $c \leq a$ means a horizontal edge and $max(a,b) = b$ is predicted.

When considering a plane spanned among the points a, b, c, X this can be seen easily. Moreover it gets clear that averaging a, b, c is not a very good idea in case there is no edge. The predictor is also interpreted differently, as being the median of three fixed predictors a, b, $a + b - c$ – and is therefore also termed as "median edge detector (MED)".



The assumption is :
'x' should be on the same plane created by a, b and c .



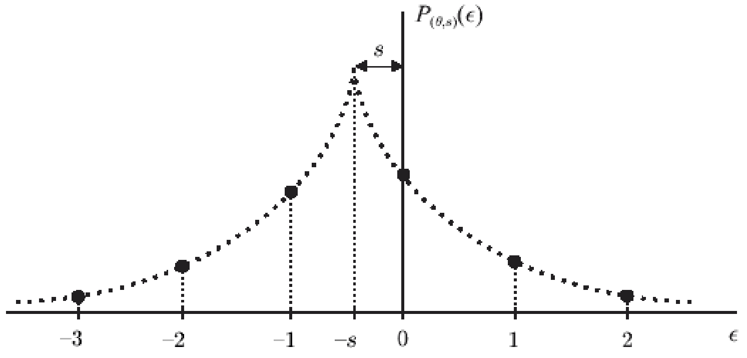
$$x_{avg} \approx \frac{2a+b}{3}$$

But,

After the prediction has been computed, the distribution of residual values is estimated based on context information. Residual values in general are known to be of two-sided geometrical distribution, the actual shape of the distribution is determined based on context information. In order to derive contexts, the quantities $d - a$, $a - c$, and $c - b$ are computed and quantized to result in 365 different contexts. According to the estimated distribution shape parameter, the values are encoded using Rice-Golumb codes (which are known to be optimal for geometric distributions.

3.1.7 GIF, PNG, TIFF

The GIF (Graphics Interchange Format) standard is a proprietory CompuServe format (which was not all all known when it became that popular as it is topday). For historical reasons (nobody believed in 1987 that it could ever be necessary to represent more than 256 colours on a computer screen) the number of colours that can be represented is restricted to 256 (out of 16 million for true colour images). This means that for more colours quantisation is applied (which means that compression is lossy). Apart from a huge number of



possible header data (e.g. for animated GIFs and local colour descriptions) the actual pixel data (in palette representation) are compressed using LZW (which is the patent problem).

PNG *Portable Network Graphics* is the license-free answer to GIF ("**P**NG **N**ot **G**IF"). PNG uses a LZ77 variant, is entirely lossless and is supported by the wordwide web consortium (W3C).

TIFF (Tagged Image File Format) supports various colour representations and image resolutions. TIFF is not a compression standard but a container file-format which supports uncompressed images and 6 different compression formats (5 lossless techniques, among them ITU-T T.4 and T.6 and a LZW variant, as well as lossy JPEG.

3.2 Lossy Image Compression

3.2.1 JPEG

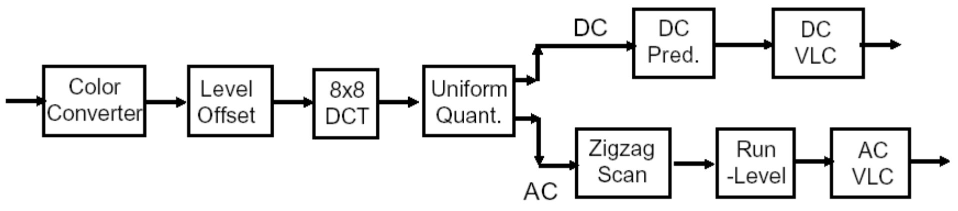
The Joint Photographic Experts Group (JPEG) has defined a set of standards for compression of contineous tone (grayscale) and colour images in the early 90s. This set includes lossless JPEG, and the lossy Baseline as well as Extended Systems.

JPEG Baseline

The codec specified for the baseline of the standard is defined up to 8bpp per colour channel and does not support progressive transmission but is based on a successive processing of 8 x 8 pixels blocks. The first stage is color conversion to YUV for decorrelation of color channels (which can be also subsampled in e.g. 4:2:0 mode). After the color conversion, the three channels are processed independently.

The next stage is to substract a fixed value from each pixel (level offset) in order to generate data centered around 0, which restricts the subsequently computed coefficients to a certain range.

After partitioning the image data into blocks of 8 x 8 pixels blocks, the DCT is applied to each of these blocks. The resulting coefficients are quantized using a separate strategy for each coefficient. This strategy is defined in the quantisation tables which can be user specified for specific types of data, otherwise the



- Color converter: RGB to YUV
- Level offset: subtract 2^(N-1). N: bits / pixel.
- Quantization: Different step size for different coeffs
- DC: Predict from DC of previous block
- AC:
  - Zigzag scan to get 1-D data
  - Run-level: joint coding of non-zero coeffs and number of zeros before it.

default tables are used. This is the stage where loss is introduced. In the following, DC and AC coefficients are treated differently in the way they are encoded. We will discuss the various stages in more detail in the following.

The discrete Cosine transform (DCT) projects the data onto a set of orthogonal basis functions consisting of cosines with different frequencies. The 2-dimensional version used in JPEG is built by multiplying two 1-dimensional functions (one for each coordinate). For each pair of frequencies (u, v) a coefficient is computed, both parameters range from 0 to 7 in JPEG. This means, we compute 64 coefficients.

$$h(m,n,u,v) = \alpha(u)\alpha(v)\cos\left[\frac{(2m+1)u\pi}{2N}\right]\cos\left[\frac{(2n+1)v\pi}{2N}\right]$$

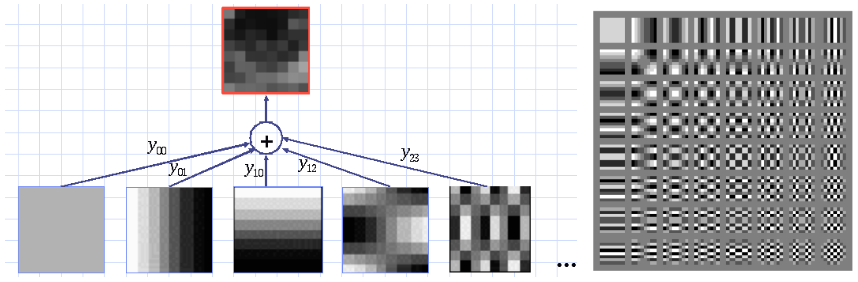
$$\text{where } \alpha(u) = \begin{cases} \sqrt{1/N} & u = 0 \\ \sqrt{2/N} & u = 1,\ldots,N-1 \end{cases}$$

$$T(u,v) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(m,n)h(m,n,u,v)$$

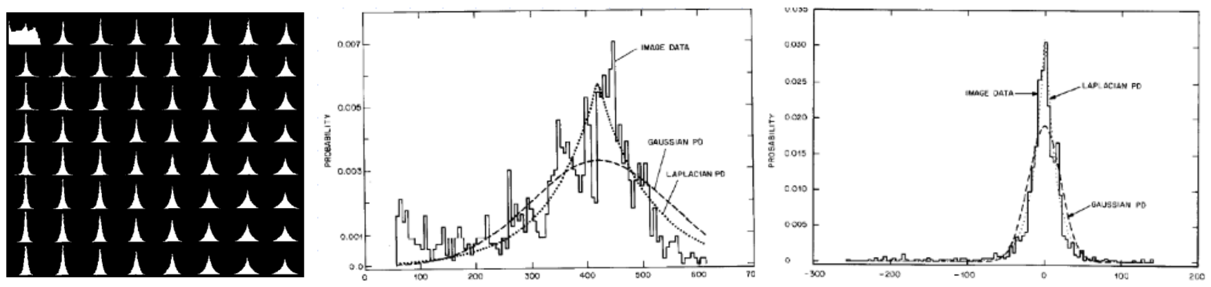
$$f(m,n) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} T(u,v)h(m,n,u,v)$$

Visually, this procedure corresponds to representing an image block by a weighted sum of “basis blocks” which correspond to the discrete cosine basis functions. These are shown in the figure. The transform determines the importance of each single block. In the reconstruction, for a single pixel $f(m, n)$ in the block we need to sum over all frequency parameters (all 64 coefficients are invloved when reconstructing a single pixel).

The 64 coefficients exhibit different properties. In particular, the coefficient for $u = 0, v = 0$ is called DC coefficient (the others are the AC coefficients). When inserting these values into the formula, it gets clear that this value is directly proportional to the average luminance of the block. The higher the frequencies get, the smaller the absolute value of the coefficients gets for typical imagery (this is due to the low frequency nature of typical image content). We also note that the histograms of the values (taken from many blocks in an image as shown in the figure) differ significantly: while the DC shows a distribution similar to a gray value histogram of an image (something close to uniform or normal distribution), AC coefficients are typically distributed following a Laplace distribution model. A we have seen, knowledge about the distribution statistics helps to



gain in compression efficiency.



The next stage in the process is to apply quantization to the computed transform coefficients. The basic operation is to divide the coefficient Y_{ij} by a value Q_{ij} and round the result to integer. It is important to note that Q_{ij} is potentially different for each coefficient and is defined in the so-called quantisation tables.

$\lfloor x \rfloor =$ largest integer smaller than x

$$l_{ij} = \left\lfloor \frac{Y_{ij}}{Q_{ij}} + 0.5 \right\rfloor = \text{round}\left(\frac{Y_{ij}}{Q_{ij}} \right)$$

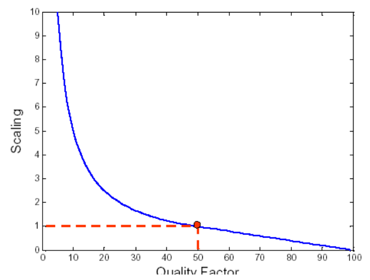
l_{ij} = label
$Y_{ij} = ij^{th}$ DCT coefficient
$Q_{ij} = ij^{th}$ Quant. Table Entry

Quantisation tables usually differ between luminance and color components and are provided in the form of default tables as shown in the figure. In case data with some specific properties are compressed, these tables can also be specified differently and optimised for a specific taget application (e.g. for fingerprint images, iris images, satellite images etc.). In this case, the tables are communicated to the decoder in the header of the compressed file. Note that the default tables have been optimised based on large scale experimentation involving thousands of subjects.

The fundamental idea in the quantisation table design is to quantise coefficients which correspond to “irrelevant” visual information more severly. This essentially means that the higher the frequency gets, the more severe quantisation is being applied. In order to “survive” quantisation, a high frequency coefficient needs to exhibit a significant size (i.e. the corresponding visual information needs to be really important).

As can be seen, color quantisation of more severe in most areas and much less sophistcated.

❑ quality factor ≤ 50: scaling = 50 / quality;
❑ quality factor > 50: scaling = 2 - quality / 50;



■ Luma Quantization Table

16	11	10	16	24	40	51	51
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

■ Chroma Quantization Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

To control compression ratio (and this is really the only means this can be done), the entries of the quantisation table Q_{ij} are scaled, i.e. multiplied by a factor. The larger the factor gets, the more severe the compression. However, the user does not control this factor explicitly, but via a "quality factor" ranging between 0 and 100. In the figure, the relation between those to magnitudes is displayed. It is important to note that the same quality settings do not at all assure similar file size for different images – actually, the file size may differ by a factor of two or even more, and also the quality is only comparable but not equal. This incapability of explicit rate control in JPEG is one of its biggest disadvantages.

The figure shows the effect of quantisation to three selected blocks in the Lena image. Due to quantisation, of course reconstruction errors are introduced which are distributed over the entire block of course (recall, that for each pixel being reconstructed, in the inverse DCT all coefficients are involved).

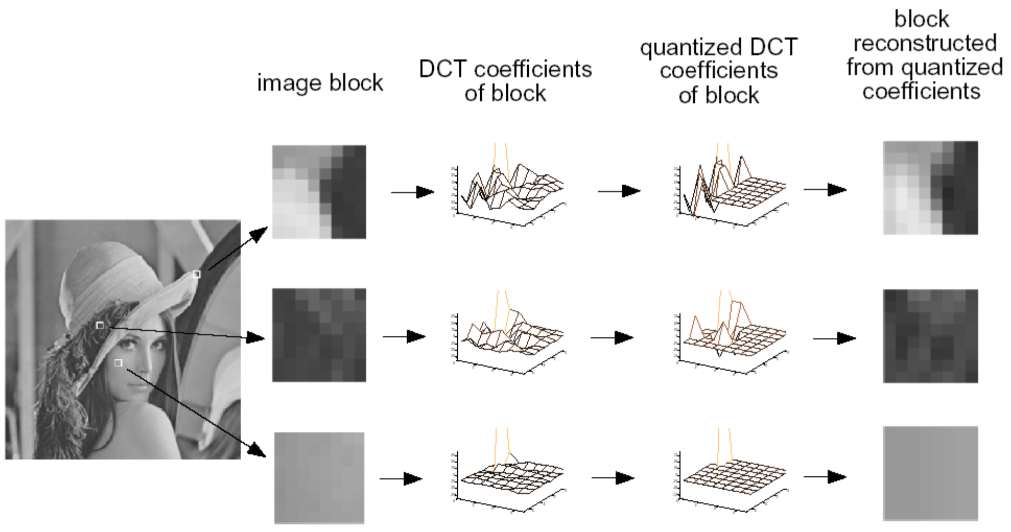
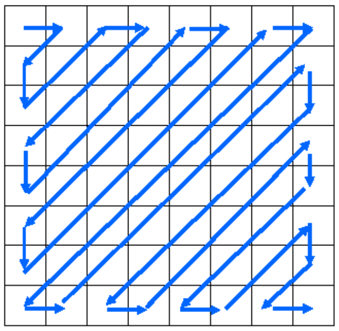


image block
DCT coefficients of block
quantized DCT coefficients of block
block reconstructed from quantized coefficients

In order to being able to apply entropy coding techniques, the 2-dimensional data structure operated on so far needs to be converted into a 1-dimensional one. This is done by the "zig-zag" scanning technique, which applies a scan starting from low frequencies to successively higher ones. The rationale behind this strategy is that after quantisation, most non-zero coefficients are concentrated in the upper left corner of the matrix, while the rest of the coefficients is 0 in many cases (recall: coefficients tend to be small and quantisation table entries are large in this area).

- Most non-zero coefficients are in the upper-left corner
- Zigzag scanning:



- Example

8	24	-2	0	0	0	0	0
-31	-4	6	-1	0	0	0	0
0	-12	-1	2	0	0	0	0
0	0	-2	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- Zigzag scanning result (DC is coded separately):
  24 -31 0 -4 -2 0 6 -12 0 0 0 -1 -1 0 0 0 2 -2 0 0 0 0 0 -1 EOB

EOB: End of block symbol. The remaining coeffs are all 0.

- Example: zigzag scanning result
  24 -31 0 -4 -2 0 6 -12 0 0 0 -1 -1 0 0 0 2 -2 0 0 0 0 0 -1 EOB
- (Run, level) representation:
- (0, 24), (0, -31), (1, -4), (0, -2), (1, 6), (0, -12), (3, -1), (0, -1), (3, 2), (0, -2), (5, -1), EOB

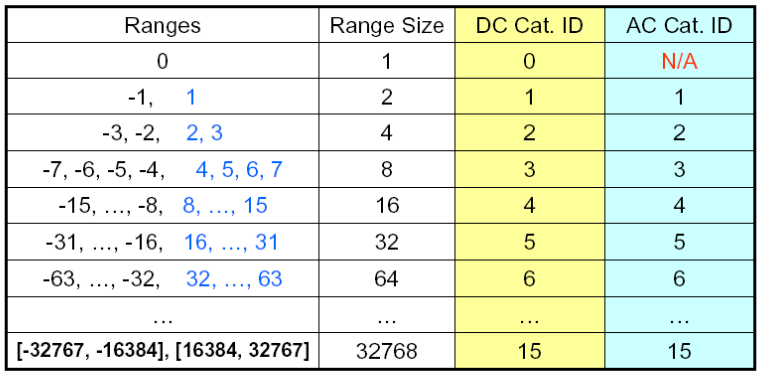
When considering the example result of the scanning process, we note that for increasing frequency, the runs of 0s get longer and 0 runs in general appear more often. This is also the reason for conducting the scan in this manner. The scanned data is then converted into a (run,level)-pair representation: run determines the number of preceeding zero coefficients and level is the actual magnitude of the coefficient. The DC coefficient is not included in the scan as it is coded separately. The data corresponding to a block is terminated by the EOB symbol, which also states that the non-coded coefficients before it are all 0.

However, coefficient magnitude (the range information) is not encoded directly in JPEG. Instead, a two layered way of representing the data has turned out to be more beneficial: the first layer (the category) determines the approximate size of the range, subsequently the actual value in this category is encoded. The figure shows the different categories for DC and AC coefficients together with their corresponding ranges. Note that the number of values in the different categories differs, i.e. increases for larger range values.

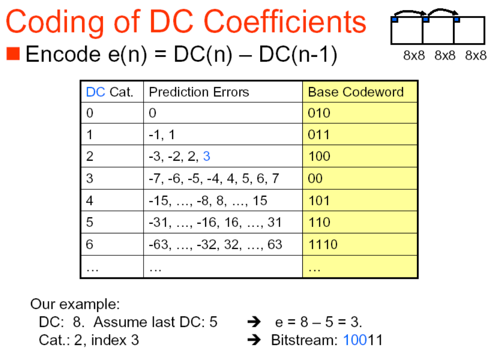
Differential encoding is applied to DC coefficients (note that this is the only part of the standard where inter-block correlations are exploited), and the prediction error is encoded using a VLC (Huffman codewords) for the category of the prediction error, index coding (i.e. the position in the category) is used to represent the value inside the category. The number of the category determines the number of bits maximally required to code the index.

The AC coefficient (run,level)-pairs are encoded differently. The 8 bit information NNNNSSSS (where NNNN is the run information and SSSS is the category information) is encoded in a Huffman codeword (fixed code), while the information about the actual value in the category is encoded like in the DC case in an index coding scheme.

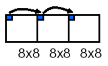
In the figure the entire processing chain of encoding and decoding a single block is visualised. For de-



Ranges	Range Size	DC Cat. ID	AC Cat. ID
0	1	0	N/A
-1, 1	2	1	1
-3, -2, 2, 3	4	2	2
-7, -6, -5, -4, 4, 5, 6, 7	8	3	3
-15, …, -8, 8, …, 15	16	4	4
-31, …, -16, 16, …, 31	32	5	5
-63, …, -32, 32, …, 63	64	6	6
…	…	…	…
[-32767, -16384], [16384, 32767]	32768	15	15



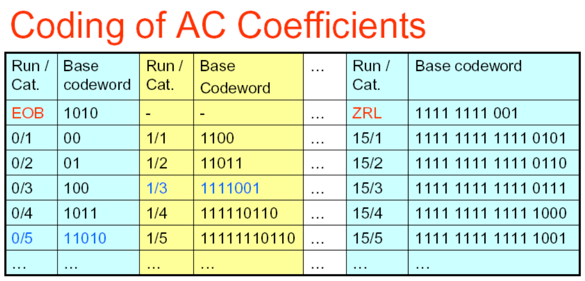
Coding of DC Coefficients



- Encode e(n) = DC(n) – DC(n-1)

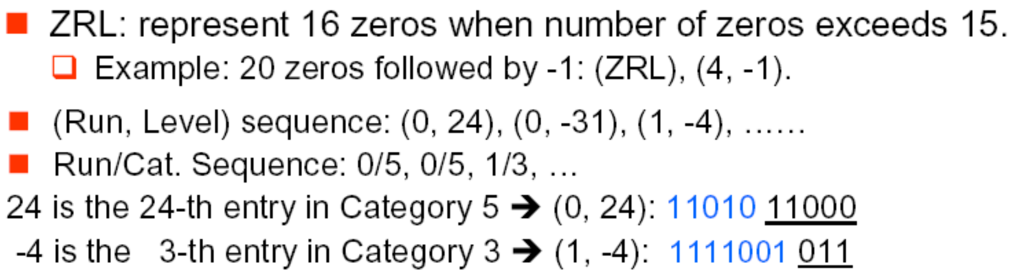
DC Cat.	Prediction Errors	Base Codeword
0	0	010
1	-1, 1	011
2	-3, -2, 2, 3	100
3	-7, -6, -5, -4, 4, 5, 6, 7	00
4	-15, …, -8, 8, …, 15	101
5	-31, …, -16, 16, …, 31	110
6	-63, …, -32, 32, …, 63	1110
…	…	…

Our example:
DC: 8. Assume last DC: 5 ➔ e = 8 – 5 = 3.
Cat.: 2, index 3 ➔ Bitstream: 10011



Coding of AC Coefficients

Run / Cat.	Base codeword	Run / Cat.	Base Codeword	…	Run / Cat.	Base codeword
EOB	1010	-	-	…	ZRL	1111 1111 001
0/1	00	1/1	1100	…	15/1	1111 1111 1111 0101
0/2	01	1/2	11011	…	15/2	1111 1111 1111 0110
0/3	100	1/3	1111001	…	15/3	1111 1111 1111 0111
0/4	1011	1/4	111110110	…	15/4	1111 1111 1111 1000
0/5	11010	1/5	11111110110	…	15/5	1111 1111 1111 1001
…	…	…	…	…	…	…



- ZRL: represent 16 zeros when number of zeros exceeds 15.
  - Example: 20 zeros followed by -1: (ZRL), (4, -1).
- (Run, Level) sequence: (0, 24), (0, -31), (1, -4), ……
- Run/Cat. Sequence: 0/5, 0/5, 1/3, …

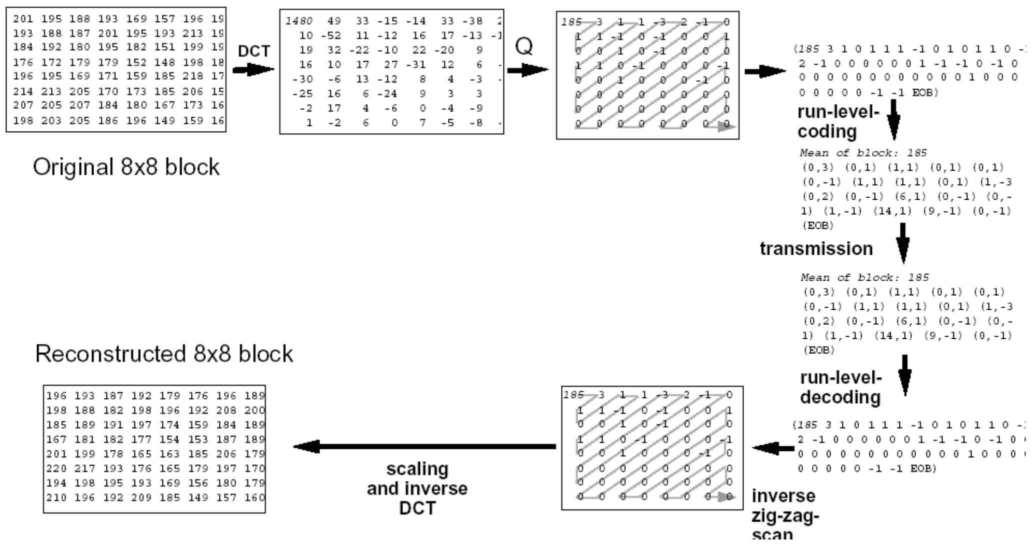
24 is the 24-th entry in Category 5 ➔ (0, 24): 11010 <u>11000</u>

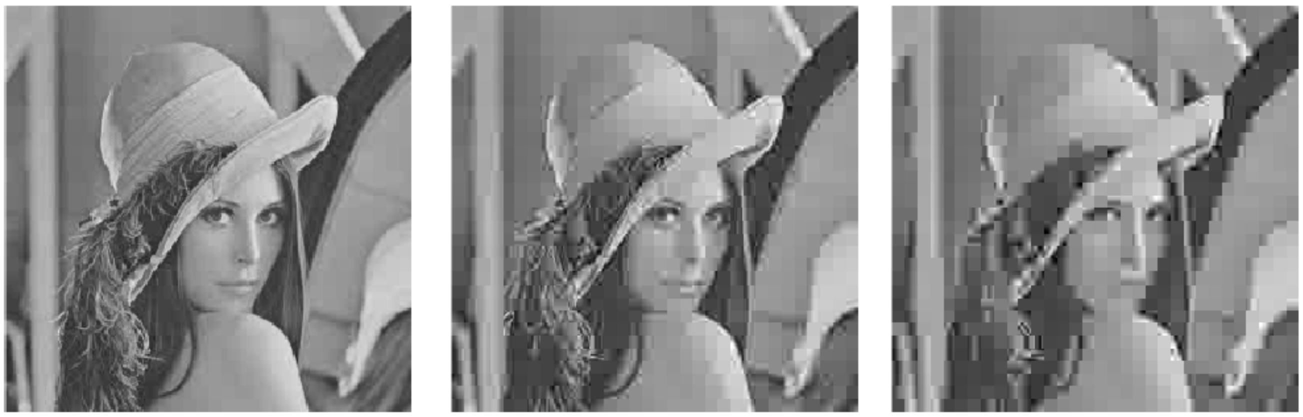
-4 is the 3-th entry in Category 3 ➔ (1, -4): 1111001 <u>011</u>

quantisation, the decoded coefficient values are simply multiplied by the scale value used during compression (this value needs to be communicated in the header, which is identical for each block in the case of JPEG but not in MPEG). Note that after inverse DCT, the error is dispersed among all pixels of the block.

The example shows typical JPEG quantisation errors: for high compression ratios, the block boundaries are clearly visible. For many blocks, only the DC coefficient is retained causing a uniform gray block in the reconstruction. Adjacent blocks with different gry value exhibit very disturbing boundaries.

A JPEG file is highly structured. A frame is a single picture which is preceeded by the main header and terminated by another marker sequence. A frame is composed of several scans, which are preceeded by corresponding table and header information (a scan usually corresponds to a spectral / color band). The next bitstream entity is a collection of segments (which is again preceeded by table and header information). A segment is composed of a couple of adjacent 8 x 8 pixel blocks.

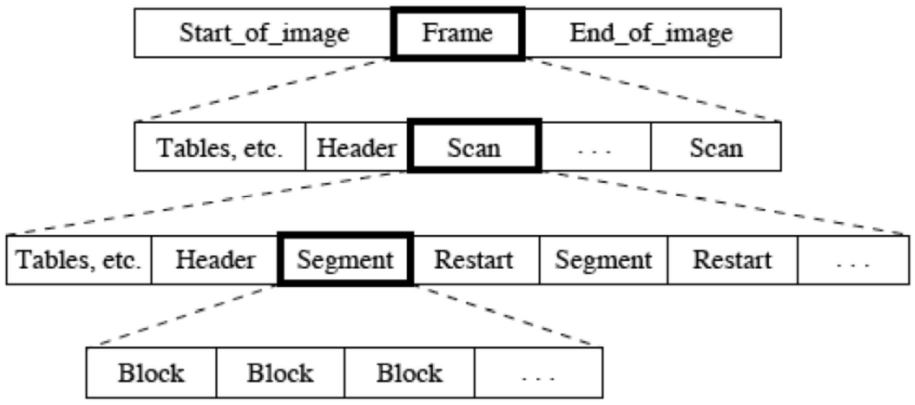



quantizer stepsize for AC coefficients: 25   quantizer stepsize for AC coefficients: 100   quantizer stepsize for AC coefficients: 200

JPEG had tremendous impact and is still the most widely used format for lossy grayscale and color image compression. The algorithm is very fast and due to the block-based processing it requires a small amount of memory only. On the other hand, the quality at low bitrates (i.e. high compression ratios) is limited, and the lack of exlicit rate-control is very inconvenient for transmission-based applications for example.

JPEG Extended System

The extended system adds support for data up to 12 bpp, optional arithmetic encoding of coefficients, and most importantly, progressive organisation of data. While in case of baseline JPEG, after interruption of a file transfer, an image is decoded only partially (depending on the number of blocks transmitted), progressive JPEG variants allow a reconstruction of the entire image based on only partially transmitted information. However, of course the quality of the data is reduced as compared to the original version. In progressive



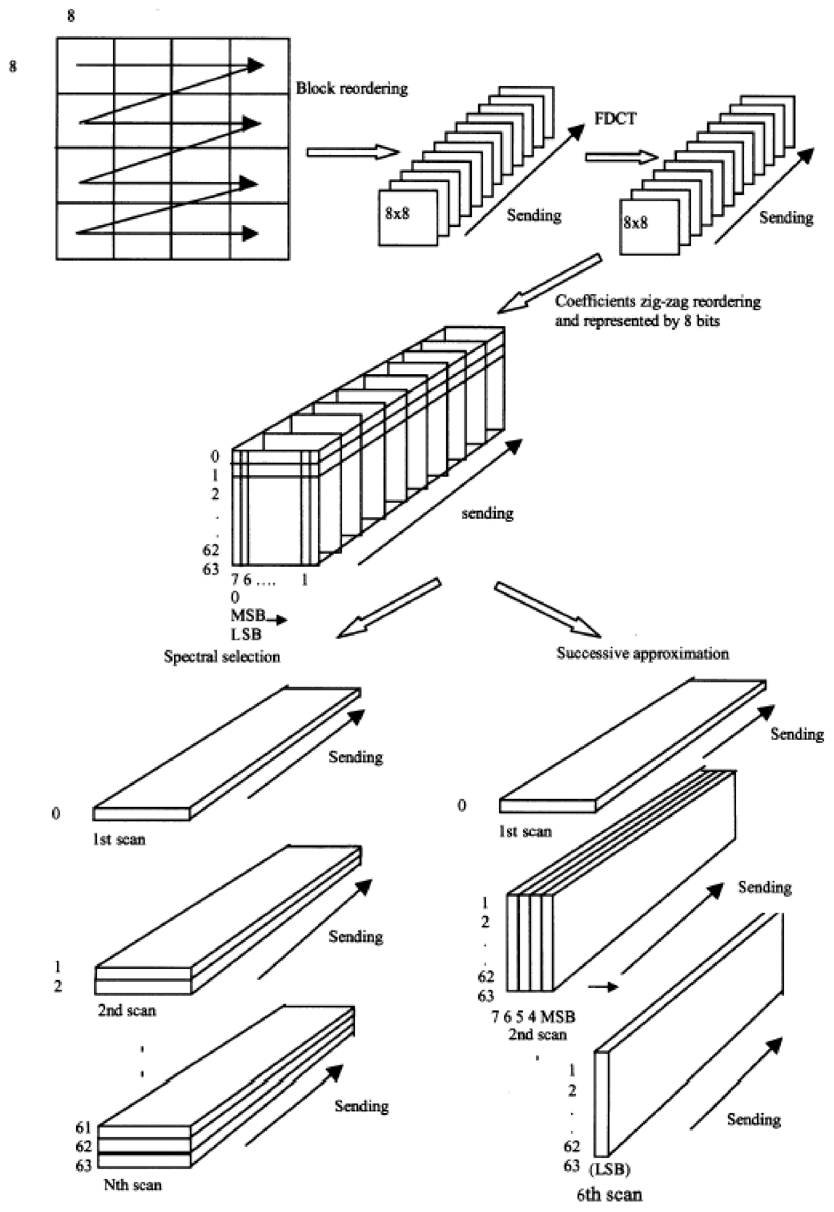
JPEG, the data is organised in "scans" which are encoded subsequently in the file. Based on a single scan, already an approximation of the entire image can be decoded. Three variants exist:

- Sequential progressive modes: single scans already contain full resolution, the image quality is increased with an increasing number of scans.
  - Spectral selection: Scans are organised according to DC and AC coefficients, e.g. the first scan may contain the DC coefficient of all blocks, the second scan first and second AC coefficients of each block, etc.
  - Successive approximation: Scans are organised according to bitplane representation of the quantised coefficients, e.g. the first scan contains the MSB of all coefficients, the second scan all next bits etc.
- Hierarchical progressive mode: the first scan contains a low resolution version of the images, subsequent scans improve on resolution.

Spectral selection and successive approximation may be interleaved in the "mixed-mode".

For the hierarchical progressive mode, we first construct a pyramidal representation of the image, e.g. a Gauss pyramid. The lowest resolution is JPEG compressed. For compressing the second pyramid level, we apply a prediction: the lowest level data is decoded, interpolated to the size of the second level and subsequently the difference between the second level and the interpolated decompressed lowest level is computed. This difference if JPEG compressed (with special quantisation tables) and stored. The same predictive procedure is applied to higher pyramid levels.

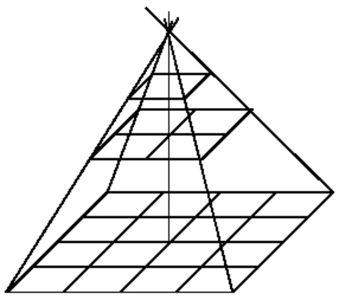
While desirable from an application viewpoint, the complexity of these schemes is higher as compared to baseline. While the sequential progressive modes are competitive in terms of compression, the hierarchical mode is not (and is also complicated to configure: which parameters with respect to scaling should be used at the different levels ?). Especially hierarchical progressive mode was not very successful. A significant disadvantage is that the decision which mode should be used has to be taken at coding time (i.e. different files are produced).

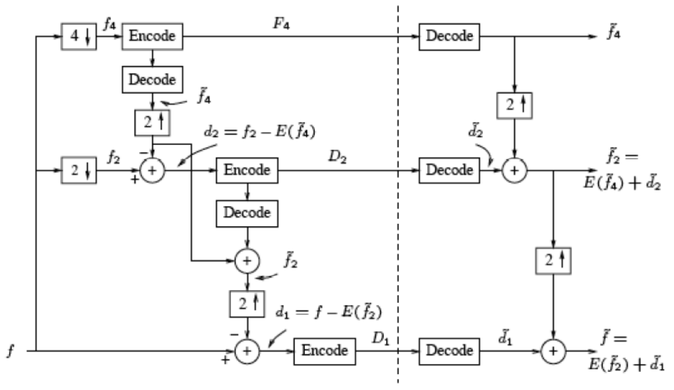


3.2.2 Wavelet-based Image Compression

The Wavelet Transform

When employing the Fourier Transform, we are not able to manipulate frequency locally. In case we modify a transform coefficient, this change is reflected in the entire signal (recall that for reconstructing a single pixel, all coefficients / basis functions are involved). For this reason the windowed Fourier Transform (of Short Time Fourier Transform STFT) has been introduced. To obtain corresponding basis functions, a Fourier basis function is modulated with a window function which is zero outside of some interval. With this trick, the FT can be made localized (to make it simple this can be seen as cutting a signal into pieces and applying





the FT onto these pieces independently). Since the window has to be defined somehow, we are facing intrinsic limitations: small windows do not capture low frequencies well, and vice versa, large windows do not capture shortlived high frequency phenonema well. The wavelet (i.e. little wave, cf. ondelette in French) transform offers a more general solution to this problem:

$$W_{a,b}(f) = |a|_{-1/2} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t-b}{a}\right) dt \tag{3.1}$$

$$\psi_{a,b}(s) = |a|_{-1/2}\psi\left(\frac{s-b}{a}\right) \tag{3.2}$$

The functions in equation (3.2) are termed wavelets, $\psi(s)$ is called "mother wavelet" as it is the prototypical function. Examples of some mother wavelets are:

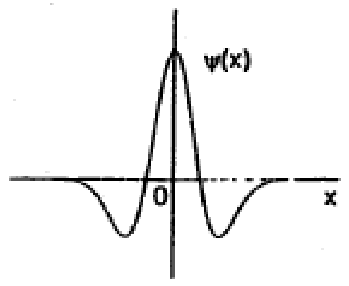
$$\psi(s) = (1-s^2)e^{\frac{s^2}{2}} \quad \text{Mexican Hat} \tag{3.3}$$

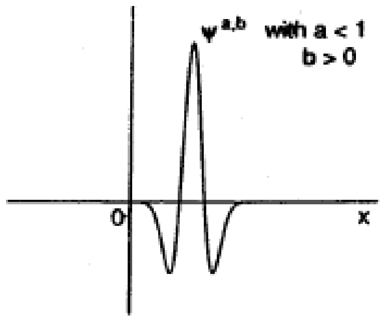
$$\psi(s) = \frac{\sin(2\pi s) - \sin(\pi s)}{\pi s} \quad \text{Shannon Wavelet} \tag{3.4}$$

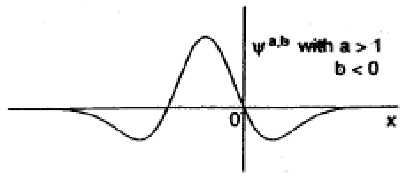
$$\psi(s) = \begin{cases} 1 & 0 \le s \le 1/2 \\ -1 & 1/2 \le s \le 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{Haar Wavelet} \tag{3.5}$$

The wavelet transform depends on two parameters a and b. In case a changes, the wavelets in equation (3.2) cover different "frequency bands" by changing the scale of the function: larger a lead to broad functions better suited to capture low frequency parts of the signal, while small a lead to slim functions suited better to represent high frequency parts of the signal. In case b changes, the localisation-center of the function (which is in $s = b$) is shifted. All wavelet functions are shifted and scaled versions of the mother wavelet.

Multiresolution Analysis (MRA) is the concept that is used to derive a fast algorithm and (bi)orthogonal wavelet basis functions with compact support. One of the fundamental problems in wavelet analysis is that functions unifying these desired properties do not have a closed expression in the time domain. The background of this approach is to represent a signal by differently scaled approximations of the signal







(different resolutions) and the corresponding detail data which is lost when changing from one resolution to the next lower one. In MRA, approximations and detail data are obtained by projecting the original signal onto two sets of basis functions (which are successively scaled to obtain various resolutions): the first set results in an approximation of the data, the second set results in the difference between the original data and the approximation.

In an MRA, the parameters a and b are subject to discretisation: $a = a_0^m$, $b = nb_0a_0^m$ with $m, n \in \mathbf{Z}$ and $a_0 > 1$, $b_0 > 1$. The most prominent choice is $a_0 = 2$ and $b_0 = 1$.

$$W_{m,n}(f) = 2^{-m/2} \int_{-\infty}^{\infty} f(t)\psi(2^{-m}t - n)\, dt$$

A MRA is spanned by approximation and detail spaces which are nested into each other, the functions $\phi(t)$ and $\psi(t)$ are the corresponding orthogonal basis functions for these spaces.

$$\phi(t) = \sum_n h(n)\phi(2t - n) \tag{3.6}$$

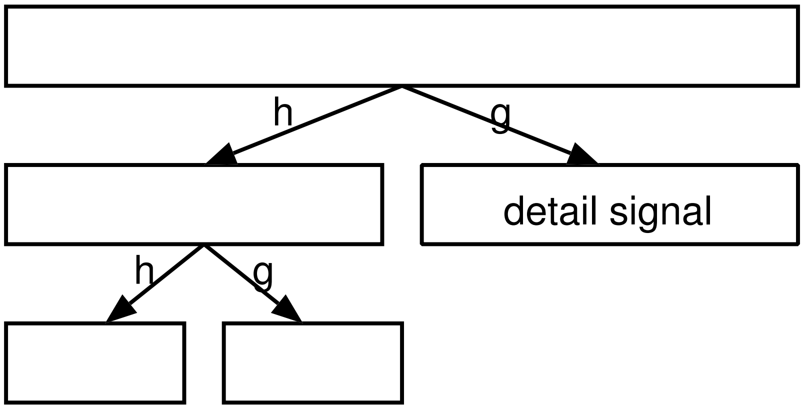
$$\psi(t) = \sum_n g(n)\phi(2t - n) \tag{3.7}$$

$g(n) = (-1)^n h(1 - n)$
$\phi(t)$ ... scaling function
$\psi(t)$ ... wavelet function

This “scaling equation” introduces a recursive relation between the scaling function and scaled versions of itself (functions of higher frequency are represented as weighted sum of scaled lower frequency functions). The same is true for the wavelet functions $\psi(t)$. Note the important fact that the sequence $h(n)$ uniquely defines the resulting functions for which we do not have a closed form expression.

The fast wavelet transform (or discrete wavelet transform DWT) is a way to compute transform coefficients in a recursive manner instead of computing inner products between signal and basis functions for each transform coefficient. A simplified example for the one-dimensional case:

$$\text{input} = (a, b, c, d, e, f, \ldots) \qquad h(n) = (1, 2, 3, 4)$$
$$WT_1 = a + 2b + 3c + 4d$$
$$WT_2 = b + 2c + 3d + 4e$$



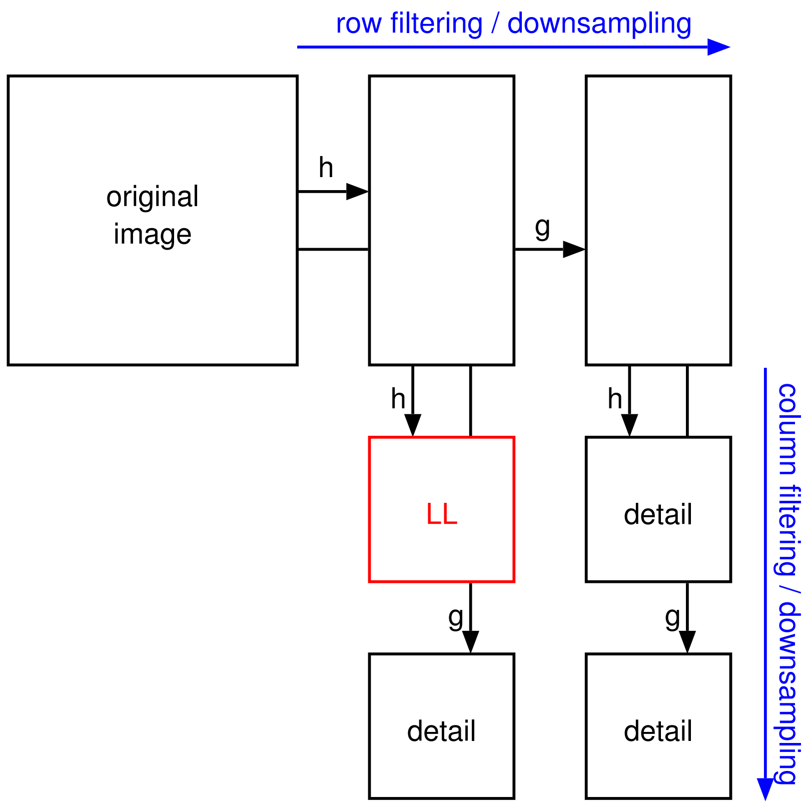
This procedure can also be interpreted as cascading low and high pass filters: the filters corresponding to the approximation spaces (scaling function) act as low pass filters, the detail space filters act as high pass filters (wavelet function). Variants how to deal with the signals edge (from stupid to more sensible):

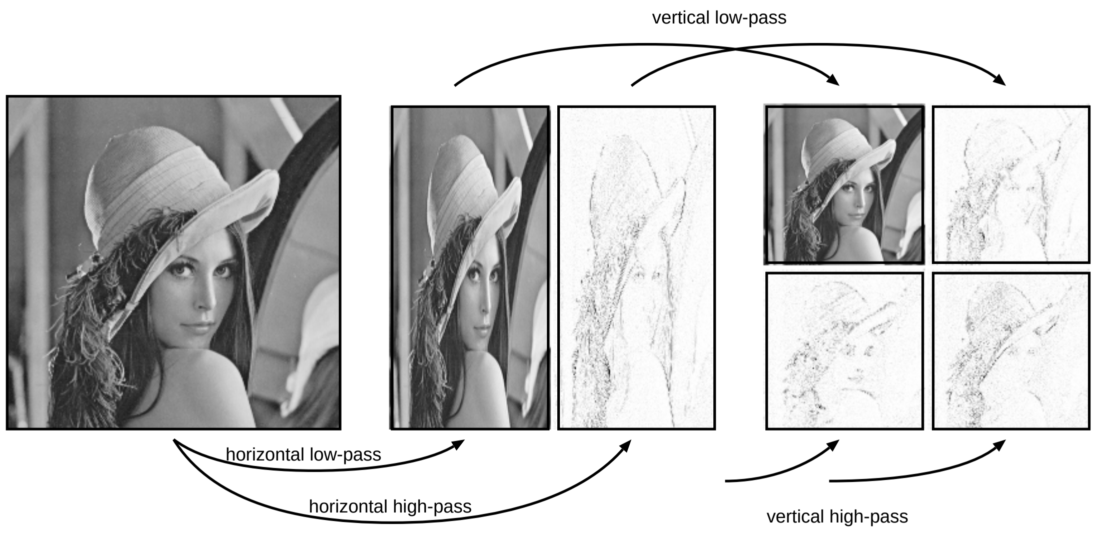
- Zero-Padding
- Periodisation
- Signal extension (copying)
- Mirroring

For decomposing images we require the transformation of two-dimensional functions. First, we transform the images lines, subsequently we apply the trnaformation to the already transformed columns of the image. By analgy to the one-dimensional case, we apply downsampling by 2 in both directions. The following figures illustrate this process.

We note that in the detail subbands, we notice edge information related to image content. In particular, the edge information is different with respect to the edge-orientation information: in two subbands, we see mainly vertical and horizontal edge information (here, the high pass part was applied in either horizontal or vertical direction only, i.e. to lines and columns), while one subband corresponds to diagonal edge information (this is where the high pass filter has been applied to both lines and columns).

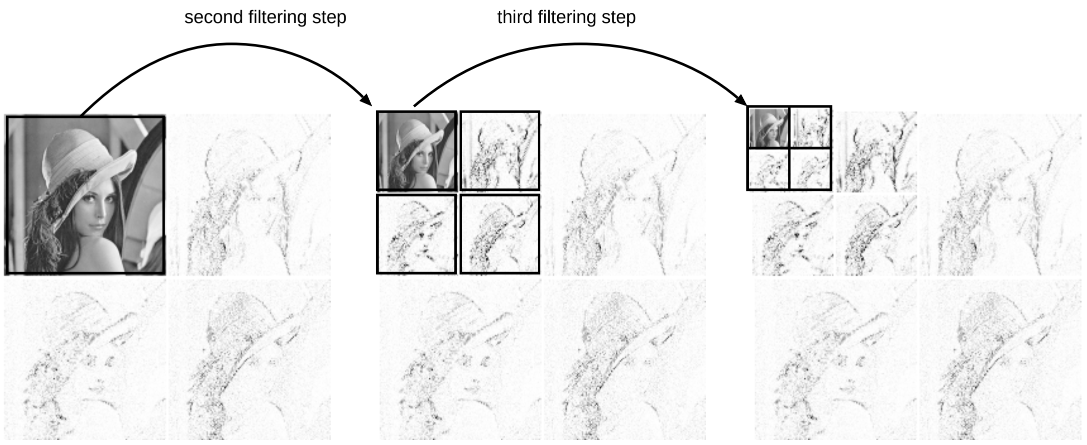
In case of the Fourier transform a single coefficient represents the frequency content of the entire image with respect to the frequencies u and v. In case of the wavelet transform a single coefficient represents the energy content in a frequency band at a specific location / position in the image. By gaining localisation in the spatial / time domain, we sacrifice localisation in the frequency band. Therefore, it depends on the type of application, which is the more suited transform (global frequency descriptions vs. localised ones).

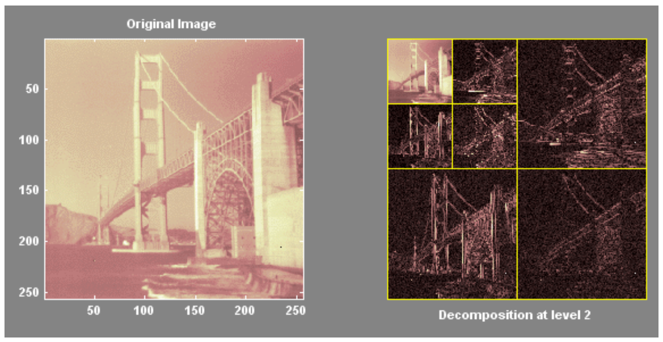


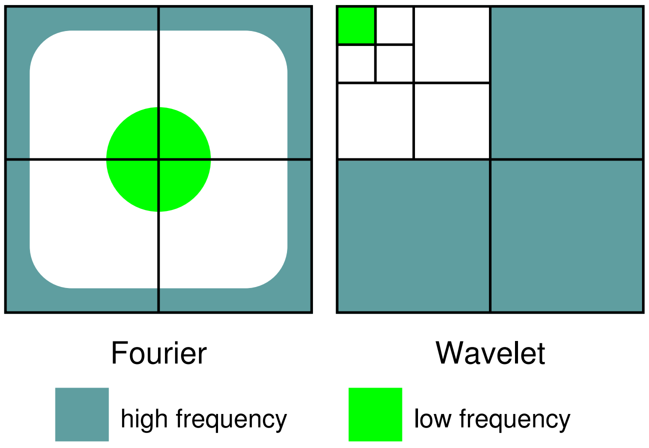


This is a very general phenonemon: we cannot have both perfect frequency and time / space localisation in time-frequency analysis; while the FT is one extremal point of the spectrum, on the other end we have impulses (or Dirac functions), wavelets range somewhere in-between.

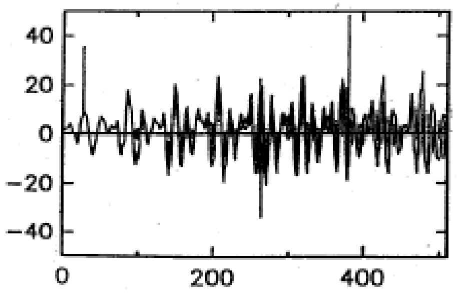
In the example shown, we exploit the advantage of the availability of localized frequency information in denoising. The displayed music signal contains “pop noise” which can originate from disc scratches or microphone artifacts. The FT domain does not any specific signs of that (rather high frequency) noise, so

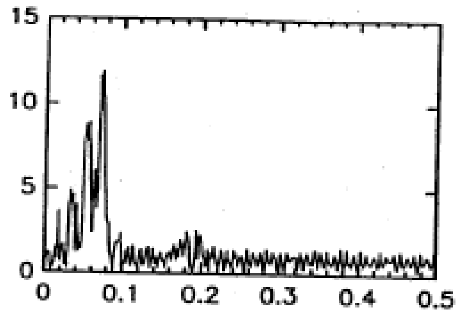






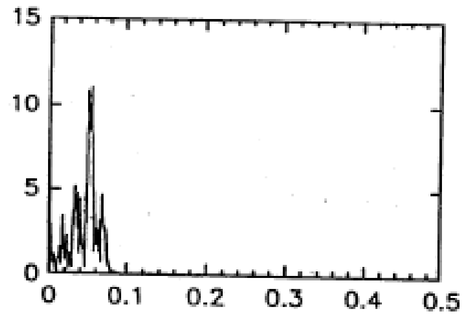
we apply a classical low-pass filtering.

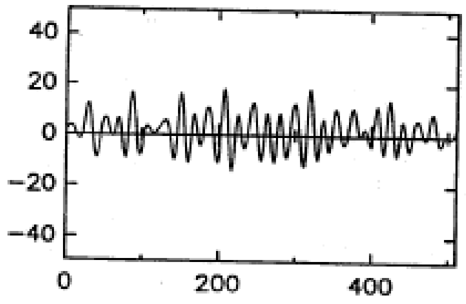




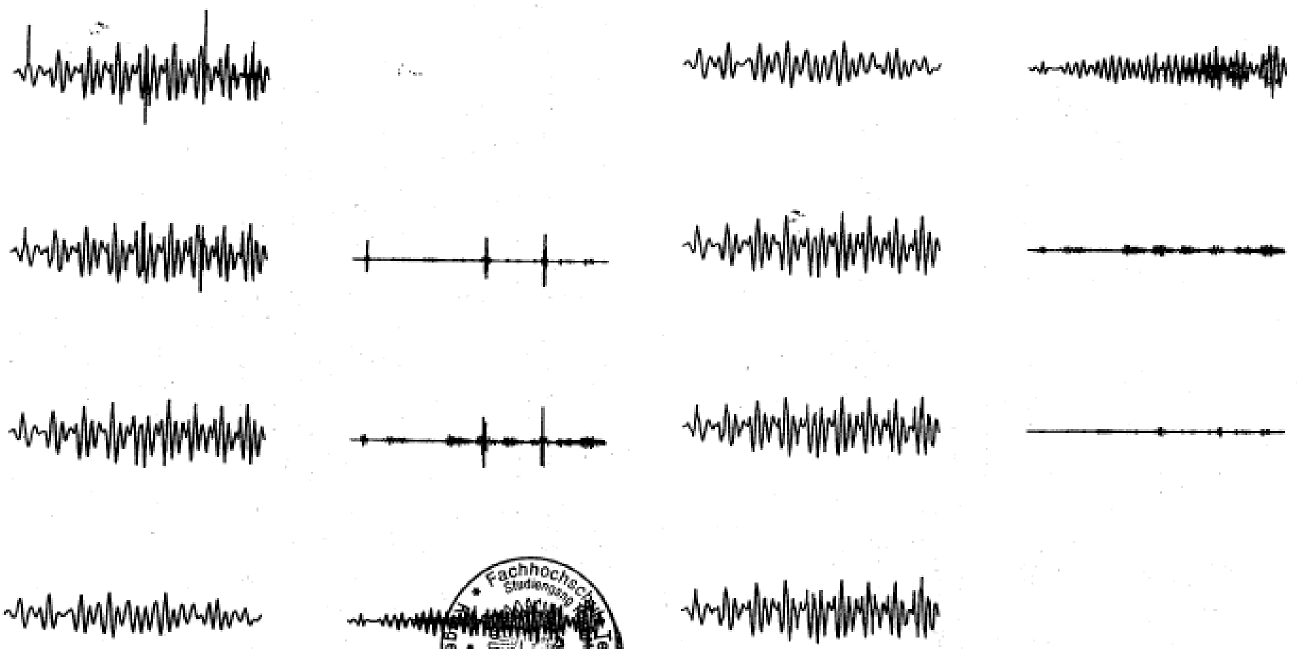
When considering the result, it get clear that the pop noise has been removed successfully. On the other hand, we of course also "succeeded" in removing the high frequency parts of the music signal resulting in significantly reduced quality.

When analysing the signal with a WT (note that in the example, the subsampling stage has been avoided), we note distinct spikes at three different locations spread accross several subbands. This means that while being of high frequency in general, the noise is spread across a couple of frequency bands which makes it specifically difficult to remove with a global low pass filtering. In the wavelet domain, the classical





denoising approach is to threshold the detail subband coefficients (the assumption is that coefficients affected by noise have a high energy).



When applying this technique to our data we realize that in the reconstructed data, high frequency content has been preserved on the one hand, on the other hand the spikes of the pop noise have been removed successfully.

The *wavelet packet transform* generalises the DWT by also conducting a recursive decomposition procedure to the high frequency subband. The result is a balanced tree which implies also a balanced frequency resolution among the subbands generated. When considering the entire tree of subbands, it is no longer clear which subbands should be used to represent a signal (which are best suited in terms of compression). In order to adaptively determine the most suited set of subbands, the "best basis algorithm" can be applied: in the fully decomposed tree, a cost function is computed for each subband. Subsequently, starting from the bottom of the tree, we compare the costs of a parent leave with the sum of the costs of the children leaves (note that the cost functions should be additive to allow fast computation). If the cost of the parent is higher, this decomposition stage is profitable and we assign the costs of the children to the parent node indicating "decompose". This scheme is applied recursively until we end up at the top of the tree. JPEG2000 Part 2 allows to apply this scheme and the FBI fingerprint standard uses a fixed WP decomposition structure to account for the high frequency nature of fingerprint images.

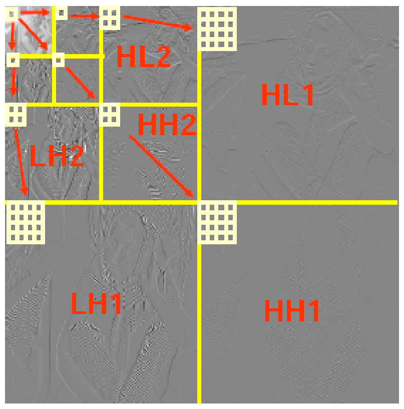
First Generation Techniques

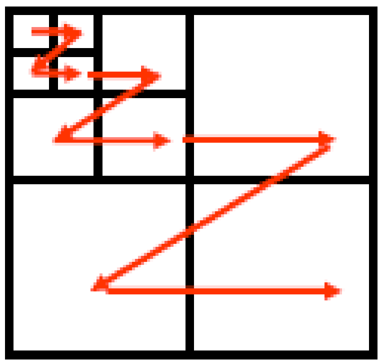
Techniques of this type have been developed soon after the DWT has been introduced in the mid 90s. They follow the classical scheme of transform coding (transform – quantisation – entropy coding) and simply replace the block-based DCT by the WT. As it can be expected, the gain as compared to JPEG for example was not significant in most cases. However, Analog Devices developed a chip for hardware-based wavelet image compression (ADV 401) using a very simple technique and the abovementioned FBI fingerprint compression standard also uses an approach like this.

Second Generation Techniques: Zerotrees

These schemes try to exploit an almost obvious property of the wavelet transform domain: the similarity of detail subbands across scales. First attempts try to accomplish this in a direct manner by predicting higher frequency subbands by interpolated versions of their lower frequency counterparts of the same orientation. Also, significant effort has been invested into exploiting these self-similarities by means of fractal compression in the wavelet domain (however, after is has been found that spatial domain fractal coding can be interpreted as a sort of simple wavelet based coding with specific filters these attempts have no longer been made).

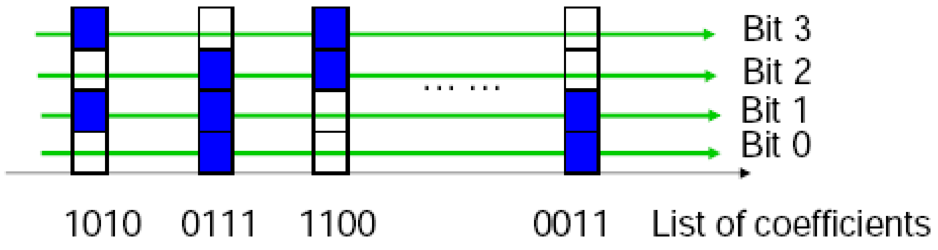
The key concept for exploiting inter-subband correlations is the idea of zero-trees. The corresponding *zerotree hypothesis* is stated as follows: "In case a WT coefficient is found to be insignificant with respect to a threshold T at a lower resolution, it is likely that its decendants in higher resolutions are also insignificant with respect to T." Due to the spatial relation of the coefficients tree stuctures across subbands can be established which can be encoded very efficiently. In case we observe an insignificant (e.g. zero after quantisation) coefficient at some resolution, we scan its corresponding children coefficients at higher resolutions for being insignificant as well. In case this property holds, we encode the coefficient at the lower resolution as **zero-tree root ZTR**. As a consequence, all its decendants do not have to be encoded. For example, in case a zerotree spans across three resolutions, we encode 21 coefficients by the single ZTR symbol.





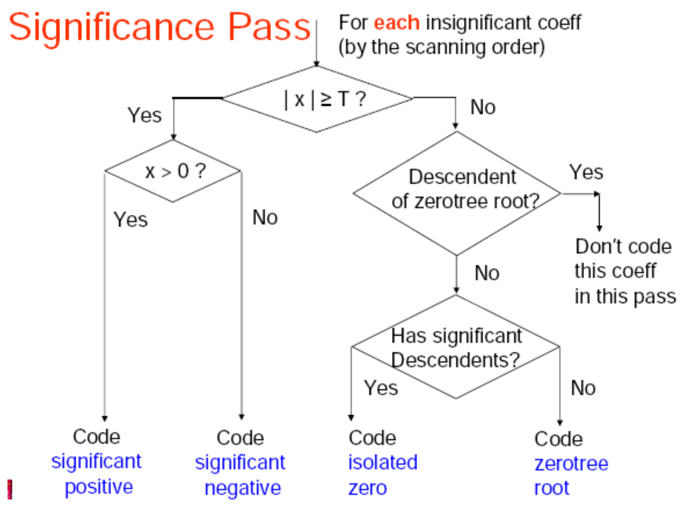
The procedure as established by "embedded zerotree wavelet" (EZW) coding is motivated by two reasons: First, contrasting to DCT based schemes edge-like structures are represented by only a few coefficients (as opposed to DCT where this information is spread across many high frequency coefficients) of the wavelet transform – however, we need to efficiently encode the position information of these coefficients. Second, there is great interest in establishing embedded / scalable dataformats which are intrinsically suited for progressive transmission (embedded: having all lower bit rate codes of the same image embedded at the begin of the bit stream, bits are generated in the order of importance and the encoder can terminate encoding at any point allowing a target bitrate to be met exactly).

The procedure starts by defining an initial threshold $T = \lfloor MaxCoeff/2 \rfloor$. The idea is to generate a "significance map" which encodes the positions of coefficients which are found to be significant with respect to a given threshold. Note that this corresponds to an analysis for which coefficients bits are set in the MSB bitplane.



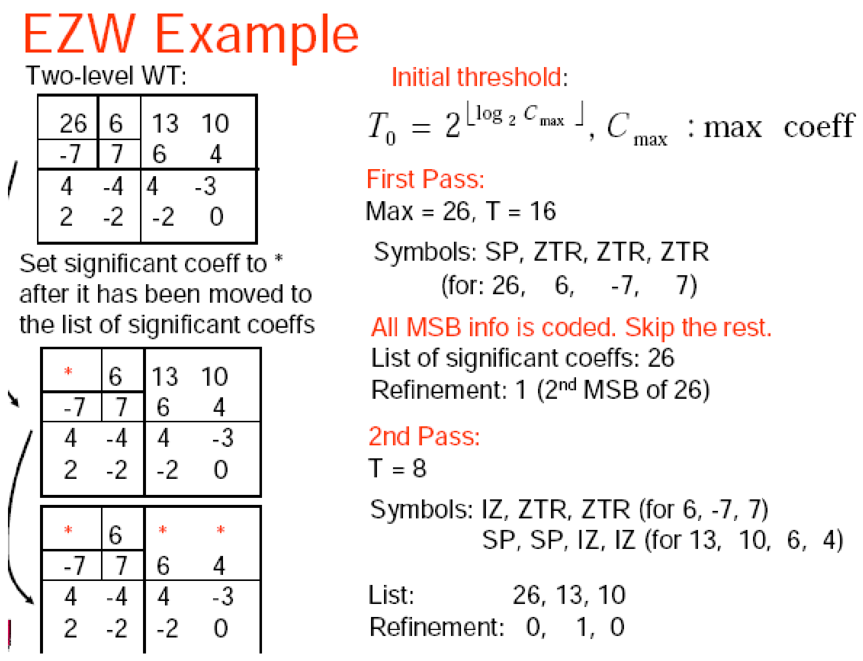
After a complete scan over the transformed image has been done, the threshold is divided by two and again, significance is determined against this new threshold. This corresponds to the analysis of the next bitplane. This procedure is repeated until all bitplanes are processed.

For each bitplane, the coefficient information is encoded in two passes: the significant pass (dominant pass) and the refinement pass (subordinate pass). At initialisation, all coefficients are insignificant.


Significance Pass
For each insignificant coeff (by the scanning order)
| x | ≥ T ?
Yes
No
x > 0 ?
Yes
No
Descendent of zerotree root?
Yes
Don't code this coeff in this pass
No
Has significant Descendents?
Yes
No
Code significant positive
Code significant negative
Code isolated zero
Code zerotree root

In the significance pass, four syntax symbols are used to encode information about the coefficients that have not been found significant so far in previous passes. For significant coefficients, significant positive (SP) or significant negative (SN) is encoded. For insignificant coefficients, either ZTR (in case all decendants are insignificant) or isolated zero (IZ) is encoded. At the highest frequency subband, ZTR and IZ are reduced into a single symbol. The coefficients for which SP or SN is coded are added to the list of significant coefficients.

In the refinement pass, an additional bit of the binary representation of coefficients already found to be significant is encoded. The scan order follows the scheme depicted in the figure concerning subbands, inside subbands coefficients are simply scanned line by line.



EZW Example

Two-level WT:

26	6	13	10
-7	7	6	4
4	-4	4	-3
2	-2	-2	0

Set significant coeff to *
after it has been moved to
the list of significant coeffs

*	6	13	10
-7	7	6	4
4	-4	4	-3
2	-2	-2	0

*	6	*	*
-7	7	6	4
4	-4	4	-3
2	-2	-2	0

Initial threshold:

$$T_0 = 2^{\lfloor \log_2 C_{\max} \rfloor}, \; C_{\max} \; : \text{max coeff}$$

First Pass:
Max = 26, T = 16

Symbols: SP, ZTR, ZTR, ZTR
(for: 26, 6, -7, 7)

All MSB info is coded. Skip the rest.
List of significant coeffs: 26
Refinement: 1 (2nd MSB of 26)

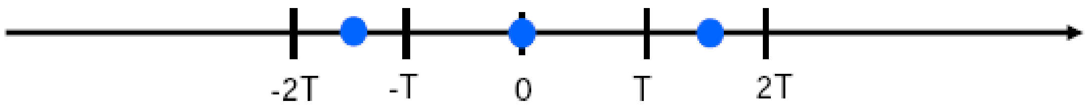
2nd Pass:
T = 8

Symbols: IZ, ZTR, ZTR (for 6, -7, 7)
SP, SP, IZ, IZ (for 13, 10, 6, 4)

List: 26, 13, 10
Refinement: 0, 1, 0

The symbols output by the embedded coding procedure are finally arithmetically encoded. Due to the fact that we have four different symbols, a non-binary variant has to be employed. For the significance pass 4 contexts are used depending on the significance of the parent and immediately neighbouring coefficients. Refinement data uses a single context.

In the reconstruction process it should be noted that bitplane encoding corresponds to uniform quantisation with deadzone.



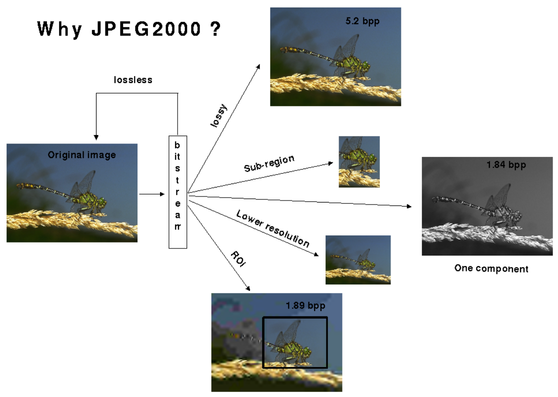
For SP, we set 1.5T, for SN we set -1.5T, and 0 for ZTR and IZ. Based on the refinement bits, coefficients are successively reconstructed with more accuracy. Note that this perfectly allows embedded decoding of the data where decoding may stop at any coefficient position.

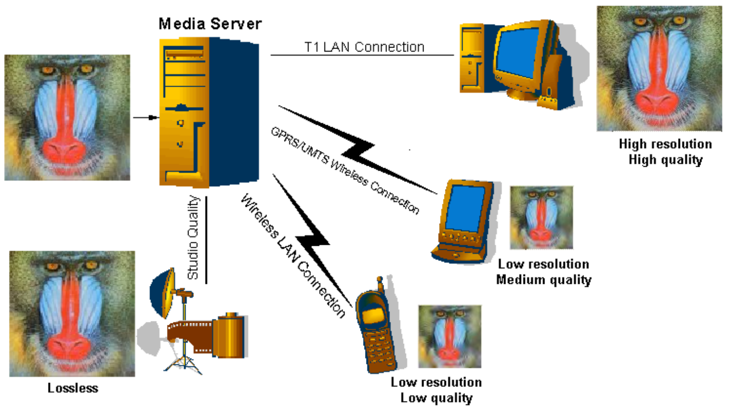
The processing of the lists has turned out not to result in the best possible compression and computation performance. As a consequence a technique termed "set partitioning in hierarchical trees (SPIHT)" has been developed, which replaces the lists by sets of coefficients and achieves even better compression performance.

Mainly due to patent-related issues but also due to the fact that the exploitation of inter-subband correlations in expensive in terms of computation and memory requirements, zerotree-based schemes have not been considered in the context of JPEG2000 standardisation.

Third Generation Techniques: JPEG2000

JPEG follows an "old" coding paradigm in the sense that for serving a specific application, a specific encoding option has to be used to produce an application specific file (lossless vs. lossy, progressive or not, hierarchical vs. sequential progressive). This means, that this decision has to be taken at encoding time. If a different application should be served based on an older bitsream, usually transcoding into a different format is required.





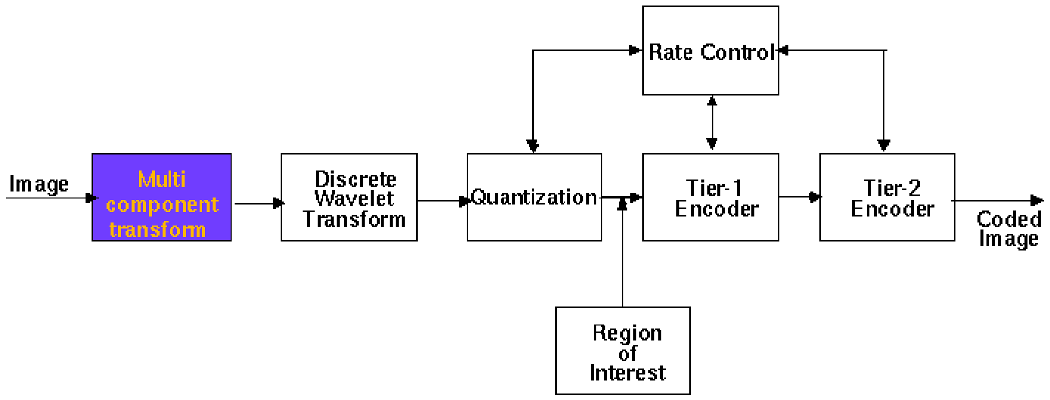
Therefore, JPEG2000 has been designed following a "new" coding paradigm where a generic bitstream is produced, out of which different bitstreams can be extracted allowing for adaptation to different application needs. No coding decision has to be taken at encoding time (apart from that if quantisation is applied, the reconstructed data is always lossy of course), the decisions which parts of the data are used to reconstruct image material is done at decoding time. This makes the format highly flexible but implies on the other hand, that the informations required for flexible decoding have to be present in the bitstream.

JPEG2000 is not based on zerotree encoding since

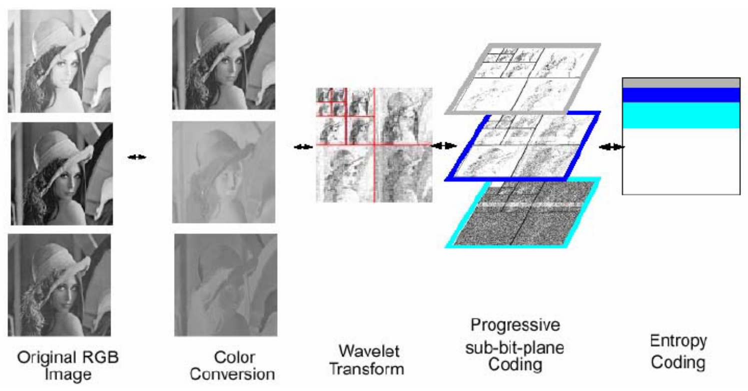
- there are many patent issues with zerotree codecs,
- the irregular coding structure traversing many subbands imply high memory consumption and expensive hardware design, and
- zerotree coding offers no resolution scalability.

All functionalities supported by the various flavours of JPEG are supported by a single bitstream (lossless + lossy, different types of progressiveness), better quality at low bitrates and explicit rate-control are featured. JPEG2000 is a typical transform coding scheme which is based on the DWT and follows the following stages.

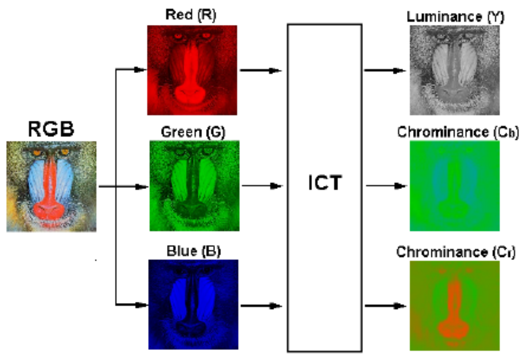
- For large images, optional tiling is applied, i.e. cutting the image into smaller pieces where each tile is subsequently compressed independently. For small images, there is only one tile consisting of the image itself.
- DC-level shift: the image data is transformed to result in approximately zero mean, i.e. for 8 bpp data, 128 is subtracted from each pixel.



- For classical (RGB) color images, a "multi component transform" is applied, decorrelating the color bands.
- The DWT is applied to each tile independantly.
- Uniform quantisation with a deadzone quantiser is optionally applied (this leads to lossy compression), for a lossless stream this is skipped.
- After Quantisation, optional region of interest coding is applied.
- Tier-1 coding applies progressive arithmetic coding of coefficient binary representations within code-blocks.
- Tier-2 coding aggregates the final bitstream from all codeblock bitstreams in a rate-distortion optimal way



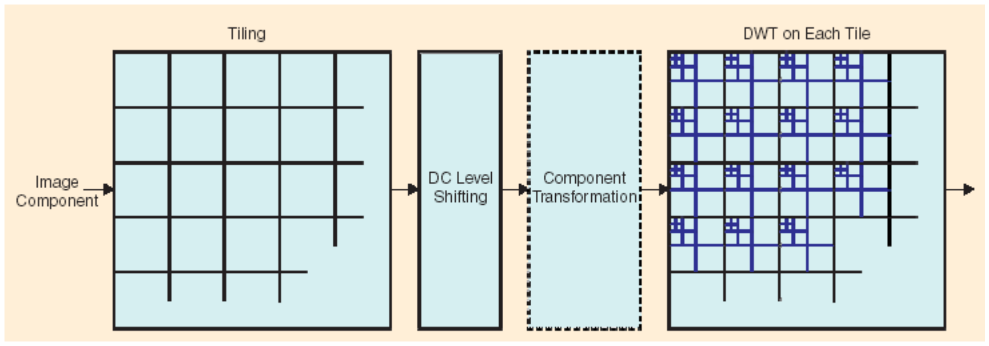
$$\begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0.299 & 0.586 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



The color transform used in JPEG2000 comes in two flavours: one for lossless coding, the other one for lossy coding. The lossy version is optimal in terms of decorrelation but leads to rounding errors, while the lossless one is an integer-based approximation of the lossy one and can be inversed without numerical error. In case that it is clear the quantisation is applied when producing the final bitstream (i.e. target is a minimal size of the bitstream but less generality), the irreversible mode should be used. In all other cases, the reversible scheme is preferable (which is also faster).

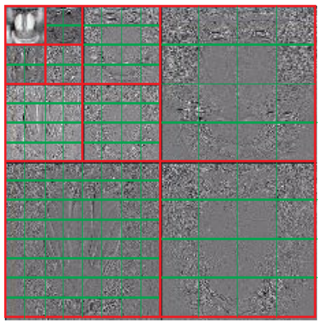
$$Yr = \left\lfloor \frac{R + 2*G + B}{4} \right\rfloor \qquad G = Yr - \left(\frac{Ur + Vr}{4}\right)$$
$$Ur = R - G \qquad R = Ur + G$$
$$Vr = B - G \qquad B = Vr + G$$

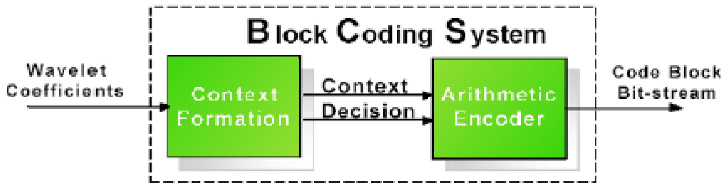
Image tiling is required only for very large images. When applied to small images, blocking artifacts similar to those introduced in JPEG appear. This can be limited by using overlapped tiling. Still, tiling should be restriced to VERY large image data !



Similar to the multi-component transform, also the DWT comes in two flavours in order to support lossy and lossless compression in an optimal manner. The lossless variant using 5/3 integer filters is based on integer operations only and is therefore also faster as compared to the 9/7 floating precision filters for lossy coding, which are optimised for energy compactation.

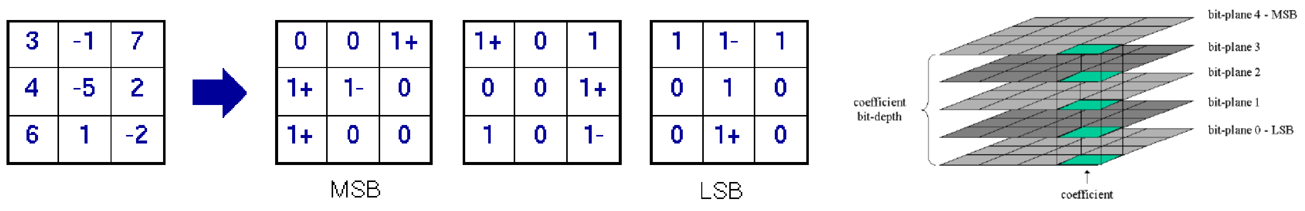
After the DWT, the transform domain is partitioned into smaller squared units, so-called “codeblocks” (default size is 64^2 coefficients, a codeblock cannot be smaller than a subband, so codeblock size and decomposition depth need to be jointly adjusted). In the following, Tier-1 coding is applied to each of these codeblocks independently by first identifying suitable contexts the information of which is fed into an arithmetic coder.





For each codeblock, the result is an independent bitstream which is subsequently processed in Tier-2 coding. Note the difference to zerotree-based schemes: in JPEG2000 we operate on coding units which are even smaller as single subbands whereas in zerotree-based coding subband borders have been traversed.

Tier-1 encoding itself relies on coding the binary representation of the codeblocks' coefficients based on their *significance*. Each coefficient contributes to several binary bitplanes. The first time (i.e. in the most significant bitplane a "1" is set) a coefficient contributes to the binary representation, the sign is coded in addition to the coefficients' binary information.



The encoding starts with the first bitplane (beginning with the most significant bits) where the binary representation of a coefficient contributes a "1". The number of empty bitplanes ("leading zero bitplanes") is signalled in the bitstream.

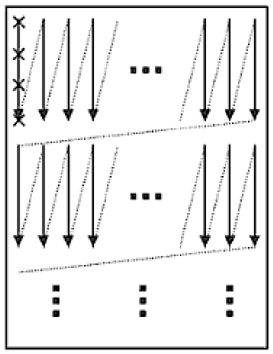
Each bitplane of a codeblock is scanned in a predifined order: starting from top left, the first four bits of the first colun are scanned, then the first four bits of the second column, until the entire width of the codeblock is covered. Then the second four bits of the first column are scanned and so on. At the lower end of the codeblock the remaining scans per column may involve four or less bits. Each coefficient in a bitplane is coded in one of three coding passes which are intended to estimate symbol probabilities of one out of three different classes of bits (the data in a codeblock associated with one of these passes is called *chunk*):

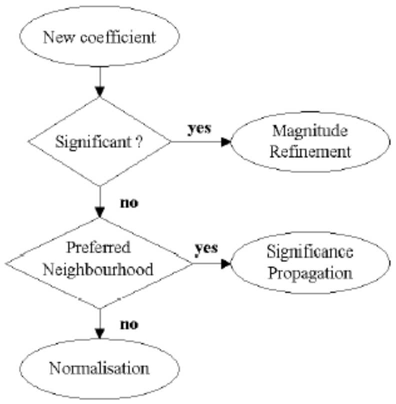
- Significance Propagation (skipped for the first bitplane)
- Magnitude Refinement (skipped for the first bitplane)
- Cleanup (Normalisation)

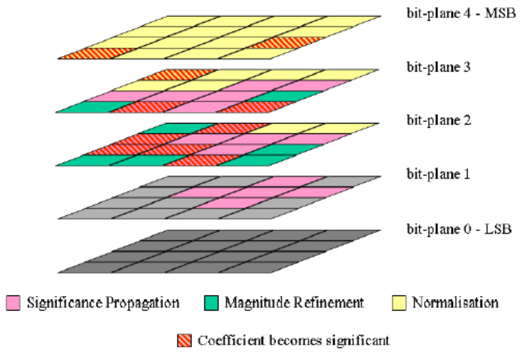
A coefficient is said to be significant at some bitplane if there is at least one non-null bit encoded in previous bitplanes. If the most significant bitplane is being processed, all samples are predicted to remain insignificant. This is the reason for skipping Significance Propagation and Magnitude Refinement for the first bitplane.

During the Significance Propagation pass, a bit is coded if its location is not significant so far, but at least one of its eight-connected neighbours is significant (significance is predicted). For the encoding nine different contexts are used (depending on which / how many neighbours are significant and in which subband / at which decomposition level the codeblock is). If a coefficient is significant it is given a value of 1 for creation of the context (and the sign is addiitonally encoded), otherwise 0. Significance Propagation only includes only bits of coefficients that were insignificant and have a non-zero context, all other coefficients are skipped. The context is passed to the arithmetic coder and the coding is performed.

The second pass is the Magnitude Refinement pass, during which all bits that became significant in a previous bitplane are coded. In this pass, the bits from coefficients that are already significant (except those that have just become significant in the immediately preceeding significance propagation pass) are being coded. The context used is determined by the significance state of the neighbours and if this is the first refinement bit or not.







The final pass (Cleanup or Normalisation Pass) encodes all remaining bits that have not been covered by the two previous coding passes (i.e. coefficients that are insignificant and had the context value 0 during the propagation pass). Beside the neighbour context, also run-length context can be used in case all four locations in the column of the scan are insignificant and each has only insignificant neighbours.

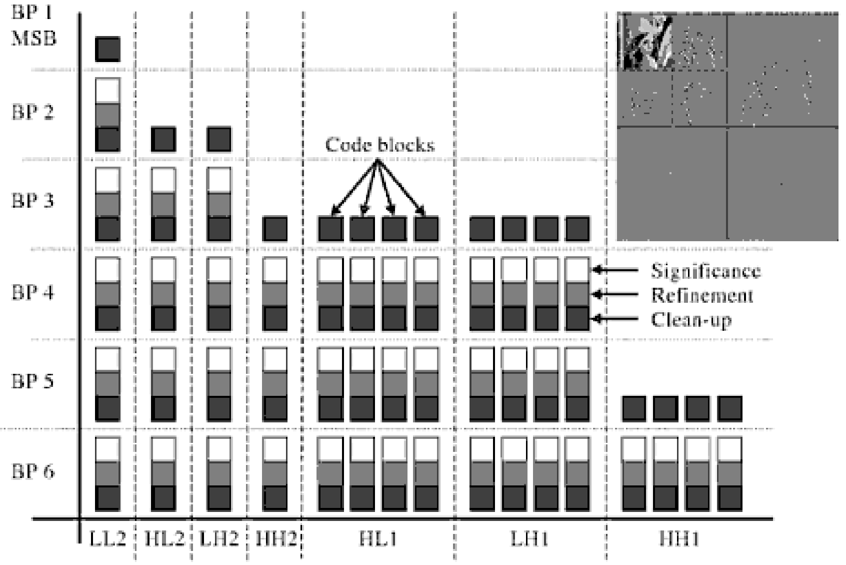
Overall, four different coding primitives are used, each of them associated with appropriate contexts, overall only 18:

- Zero Coding (ZC): used in the significance propagation and cleaup passes for the p-th bit if the coefficient is not yet significant (0 if it is still insignificant at the current bitplane, 1 if becomes significant).
- Sign Coding (SC): in the significance propagation and cleaup passes, used when a coefficient first becomes significant.
- Magnitude Refinement (MR): in the refinement pass to refine one more bit of a significant coefficient.
- Run-length Coding (RLC): in cleanup pass, used to code the significance of multiple consecutive insignificant coefficients.

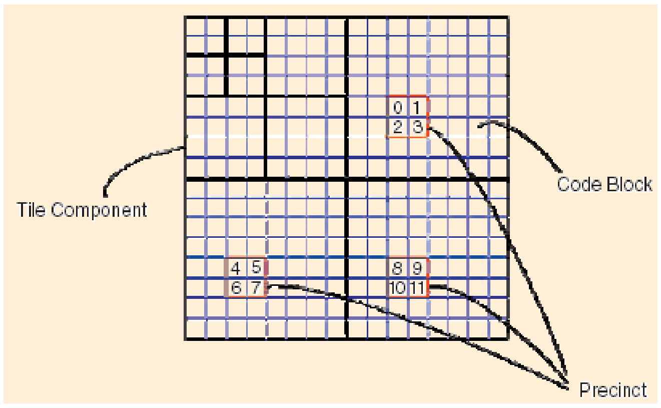
While the output of cleanup passes is always arithmetically encoded, there is an option to restict arithmetic coding for the other two passes to the four most significant bitplanes, while the remaining data is written in raw mode: *arithmetic-coding bypass or lazy mode*. This is used to trade of computational complexity for coding efficiency.

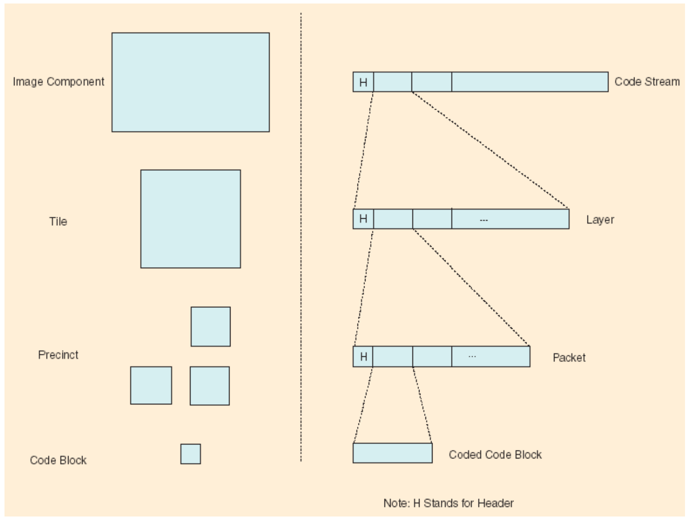
In the figure we have a visualization of how the (chunks of the) three coding passes are distributed among codeblocks in an entire wavelet decomposition tree. While in the lower frequency subbands, only a single codeblock is contained in a subband, the number of codeblocks increases with the frequency. The leading non-zero bitplane gets lower (resp. higher in number) as the frequency increases (this is due to the decreasing magnitude of the coefficients as frequency increases). As can be seen, in this bitplane only chunks corresponding to the Cleanup pass are present.

After the creation of the single bitstreams for each codeblock, Tier-1 encoding is finished. Subsequently, the final bitstream is assambled in Tier-2 coding.



For enabling Tier-2 coding, codeblocks are organised into three spatially consistent rectangles, so-called “precincts”: at each resolution level, a data structure (typically consisting of four codeblocks) from each subband is collected and fused forming the precinct. For each codeblock bitstream, the encoder stores temporal information with respect to possible truncation points and the resulting errors when actually using these points. The main task of Tier-2 coding is to assemble the codeblock bitstreams into the final bitstream using this stored temporal information in a rate-distortion optimal manner.

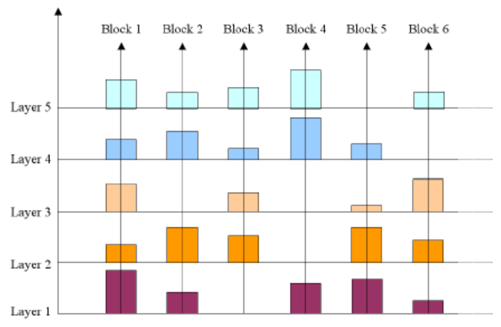


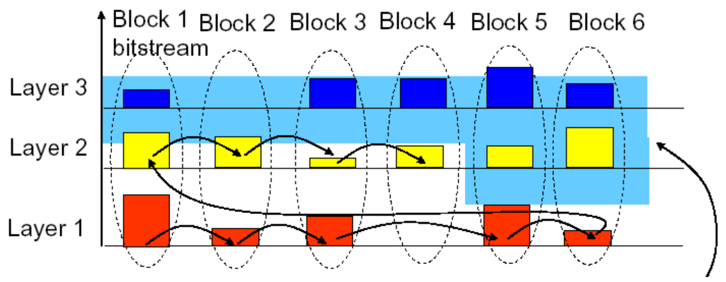


The smallest unit in the bitstream is a packet (body), which consists of code block chunks of each codeblock in a precinct. A collection of packets, one from each precinct of each resolution level, constitutes a (quality) layer. So, a packet can be interpreted as one quality increment for one resolution level at one spatial location (since precincts correspond to spatial locations). By analogy, a layer can be interpreted as one quality increment for the entire full resolution image. Each layer successively improves the image quality so that the decoder is able to decode the codeblock contributions contained in each layer in sequence.

The information stored in the codeblock header is generated as follows. First, the information if data is contained in the packet is signalled. For each subband and each codeblock in the precinct inclusion information is encoded (if no new coding passes included skip codeblock): in case it is the first inclusion of the codeblock, the number of leading insignficant bitplanes is additionally coded, in any case the number of new coding passes (i.e. chunks) and corresponding length information is coded in a scheme called “tag-tree coding” (which is a quadtree based scheme). Bit stuffing is applied to prohibit appearance of certain marker codes.

Based on the temporal information stored with each codeblocks’ bitstream, it is determined how many chunks of each codeblock are fused into the first quality layer to obtain a certain target bitrste and distortion. Each subsequent layer is formed by optimally further truncating the codeblocks’ bitstreams. Rate control can be done in two manners: first, by adjusting quantiser step sizes, and second (which is computationally more involved), by ommitting certain coding passes when building packets.





The final bitstream is organised as a succession of layers the number of which can be defined (this determines the granularity of the progressiveness). Each component is encoded separately but the final bitstream is interleaved on a layer basis.

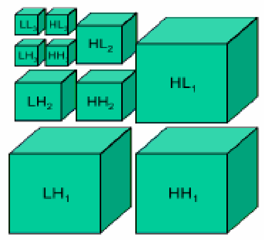
There are four types of progressiveness possible for the final bitstream which are achieved by appropriate ordering of the packets in the bitstream:

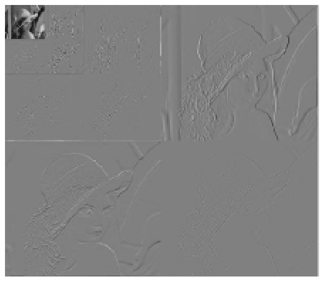
- quality progressiveness
- resolution progressiveness
- spatial location progressiveness
- component progressiveness

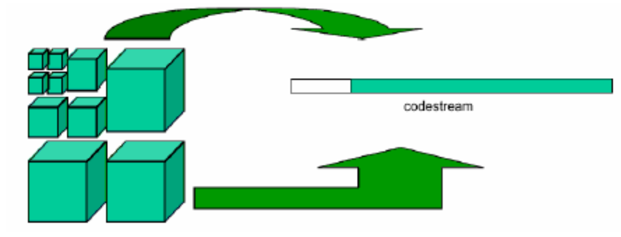
Once the bitstream has been assembled in a certain order, different types of progressiveness are obtained by only reordering the bitstream and decoding the information in this diferent ordering (to enable this, the corresponding information from packet headers need to be extracted and interpreted).

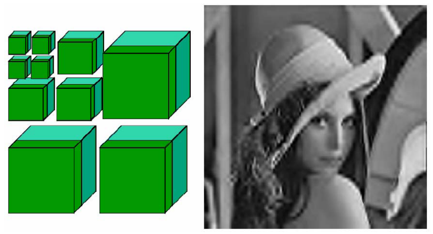
In the examples we see a schematical visualization of how the data is organized to obtain several quality and resolution levels, respectively.

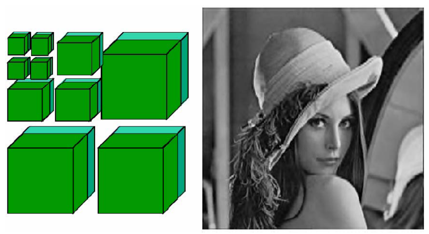
It should be noted that the high flexibility of the data format of JPEG2000 has to be paid with some restrictions in terms of coding performance, which is due to the required packet header informations. In

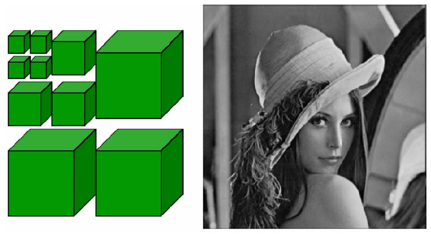


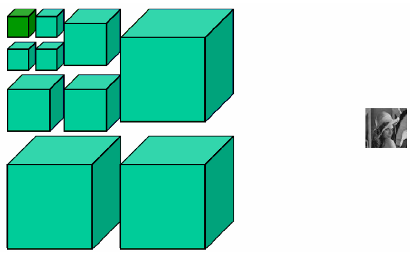


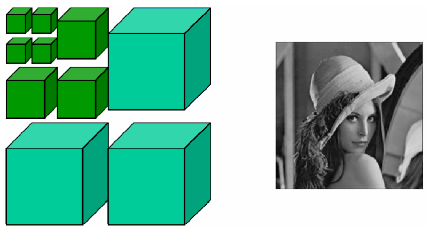


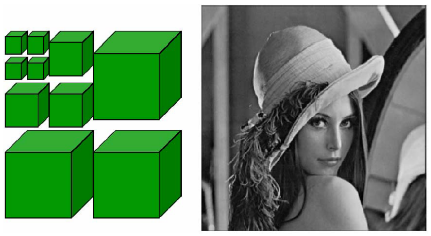






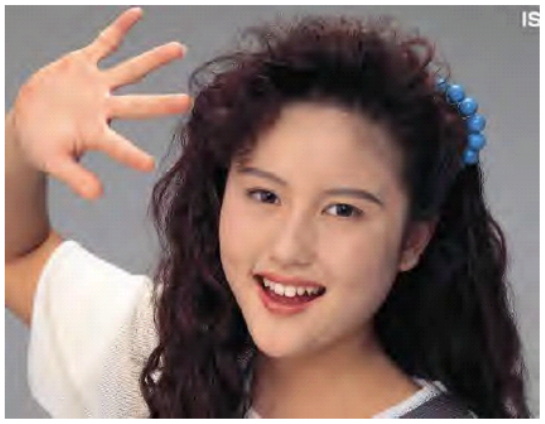


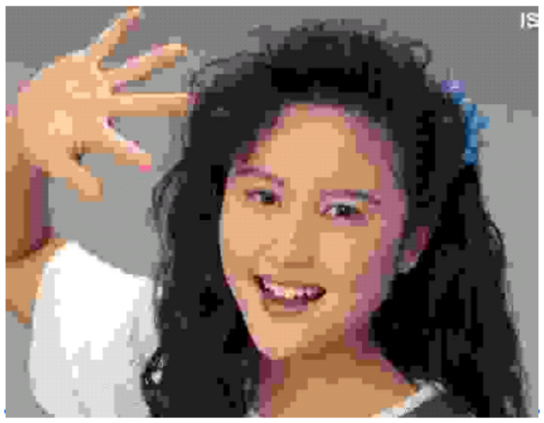




order to limit this impact, a small number of quality layers has to be formed, which causes the number of chunks included in each packet to be large thus minimizing the number of packets (and therefore packet headers) which limits corresponding coding overhead. Nevertheless, the coding performance (especially in the low bitrate area) when compared to JPEG is impressive.

Examples for Coding performance (at 0.125 bpp as compared to JPEG):





Region of interest coding is one of the remarkable properties of JPEG2000. This functionality is important in applications where certain parts of the image are more important than others, which implies that it is desirable to encode these areas with higher quality. Also during transmission, these regions should be

transmitted first (or with higher priority). The standard does NOT define how a RoI information can be generated (i.e. given an arbitrary image, how to automatically determine which areas should be the RoI), but does specify how these areas should be encoded based on a given RoI location information. Examples from biometrics imagery: in surveillance video, face detection can be applied to identify faces and encode those in a RoI in order to get better face recognition results. Similarily, eye detection can be applied to face images to identify iris data and encode these in a RoI in order to get better iris recognition results.

The RoI coding the Part 1 of the standard is based on the MAXSHIFT technique, while the more general scaling-based technique and the sequence-based mode are covered in Part 2 of the standard. The general idea of the general RoI idea is to scale (shift) coefficients such that the bits associated with the RoI are placed in higher bitplanes than the bits associated with the background. As a result, during the encoding procedure, the most significant RoI bitplanes are placed in the bitstream before any background bitplanes. Depending on the actual scaling values, the less important RoI bits may be encoded together with the non-RoI bits.

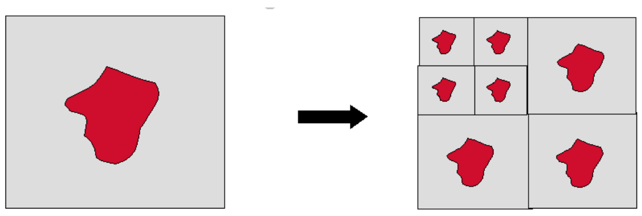
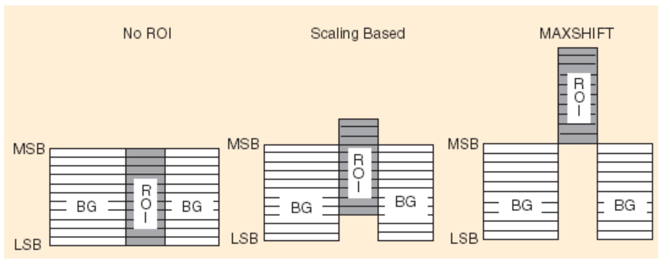
If the bitstream is truncated, or the encoding process is terminated before the whole image is fully encoded, the RoI will be of higher fidelity than the rest of the image.

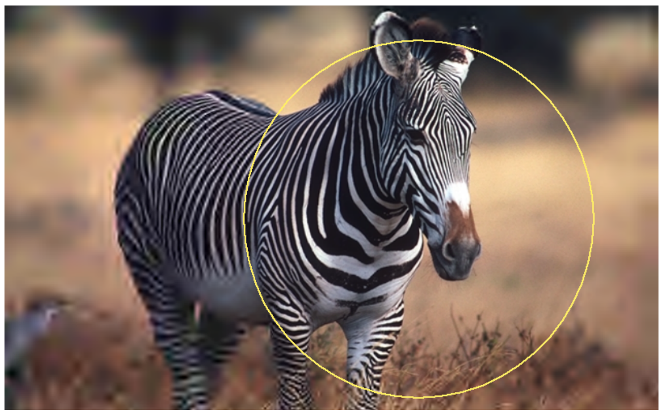
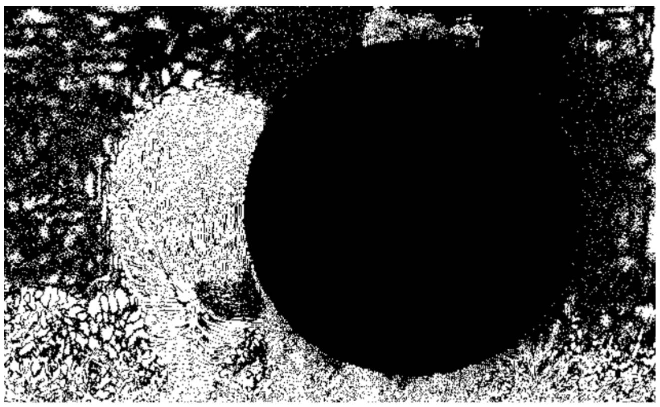
The general scaling-based RoI technique works as follows:

1. Compute DWT.
2. Based on the defined geometrical RoI information, the RoI mask is derived indicating the set of coefficients which is affected by RoI encoding.
3. Optional quantisation is applied.
4. The coefficients outside of the RoI are downscaled by a specified scaling value.
5. The resulting data is Tier-1 and Tier-2 encoded.

In the general scheme, the decoder requires the scaling value as used in the scaling and the geometrical RoI information which is included in the bitstream. In the specific MAXSHIFT technique of JPEG2000 of Part 1, the scaling value is chosen in a way that the minimum coefficient belonging to the RoI is larger than the maximum coefficient of the non-RoI areas. In decoding, the decoder simply scales up each non-RoI coefficient.

An important advantage as compared to the general scaling mode is that the decoder does not need to have the geometrical RoI information due to the magnitude separation of the coefficients, which is good with respect to data amount (RoI information is not stored), encoder complexity (RoI information is not encoded), and decoder complexity (RoI is not decoded). Additionally, different bitrates (i.e. qualities) can be defined for RoI and background (which cannot be done in the general scaling-mode). The disadvantage as compared to the general scaling-based approach is a sightly increased datarate which is due to the separation of the data (context information can be expoited less efficiently).

<u>Parts of the JPEG2000 standard suite</u>:

1. Core coding system as being described.
2. Extensions: more flexible forms of decompositions (in terms of filters and subband structure), sequence mode in RoI coding, metadata sets and JPX as file format.
3. Motion JPEG 2000: defines a file format called MJ2 (or MJP2) for motion sequences of JPEG 2000 images. Support for associated audio is also included.
4. Conformance: test procedures for conformance testing and bitstreams.
5. Reference software: JasPer (C) and JJ2000 (JAVA)
6. Compound image file format: JPM can be used to store multi-page documents with many objects per page. Although it is a member of the JPEG 2000 family, it supports the use of many other coding or compression technologies as well. For example, JBIG2 could be used for regions of text, and JPEG could be used as an alternative to JPEG 2000 for photographic images.
7. abandoned

8. JPSEC (security aspects): Encryption, source authentication, data integrity, conditional access, ownership protection. The underlying techniques to protect the content include digital signatures, watermarking, encryption, scrambling, and key generation and management. These techniques will be enabled in JPSEC by means of a registration authority. More specifically, all techniques have to be previously registered in a central repository, the registration authority, which uniquely identify these techniques.

9. JPIP (interactive protocols and API): JPIP is a client-server protocol which may be implemented on top of HTTP, but is designed with a view to other possible transports. To facilitate its deployment in systems with varying degrees of complexity, JPIP handles several different formats for the image data returned by the server: these include ordinary image formats, such as complete JPEG or JPEG 2000 files, and two new types of incremental "stream" that use JPEG 2000's tiles and precincts to take full advantage of its scalabilty properties.

10. JP3D (volumetric imaging): is concerned with the coding of three-dimensional data, the extension of JPEG 2000 from planar to volumetric images. An essential aspect of volumetric coding, because of the enormously increased quantity (compared to 2D) of potential sample points, is the use of non-uniform grids to concentrate the data in the regions where it is most significant.

11. JPWL (wireless applications): is standardising tools and methods to achieve the efficient transmission of JPEG 2000 imagery over an error-prone wireless network. More specifically, JPWL extends the elements in the core coding system described in Part 1 with mechanisms for error protection and correction. These extensions are backward compatible in the sense that decoders which implement Part 1 are able to skip the extensions defined in JPWL. The JPWL system supports the following functionalities: Forward Error Correcting (FEC) codes, data partitioning and interleaving, and unequal error protection.

12. ISO Base Media File Format (common with MPEG-4)

Major applications for the JPEG2000 standard so far:

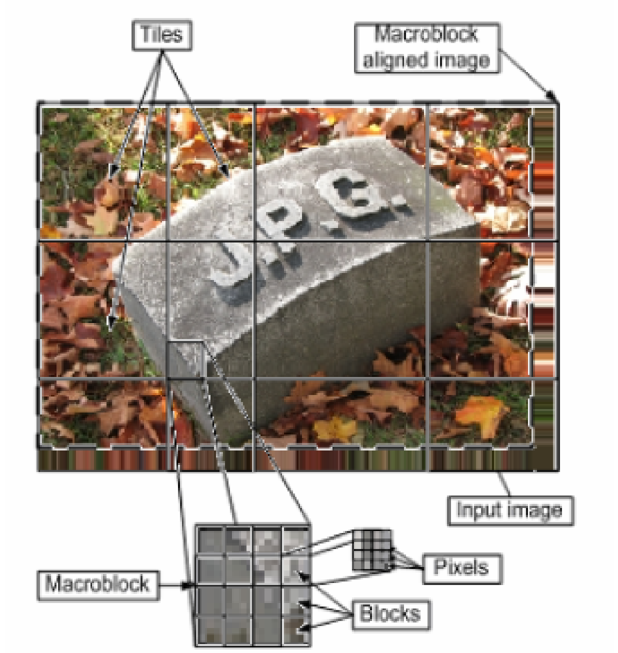
- Digital Cinema Initiative (DCI): each "digital movie" you watch in Austrias cinemas is decoded MJPEG2000.
- DICOM: of specific interest due to the lossless variants.
- Biometrics data interchange standard.

Due to the inability to penetrate the consumer camera market with this format, the widespread use of JPEG2000 has not been achieved so far. It seems that there is no reason to replace JPEG in this market segment. It will be interesting to see what the impact of JPEG XR will be.

3.2.3 JPEG XR – Microsoft HD Photo

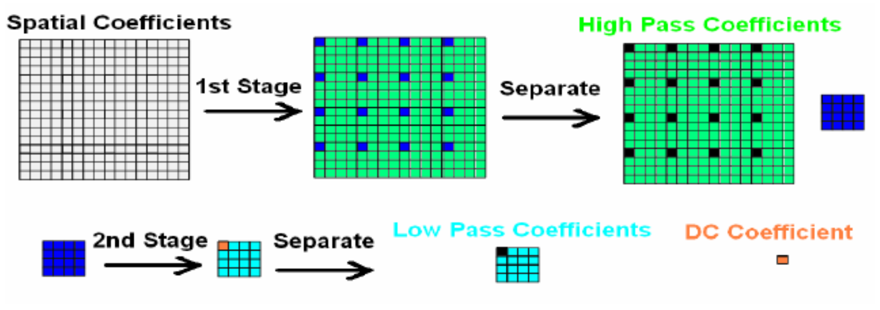
Originally developed by Microsoft and termed "HD Photo", JPEG XR got standardized by ITU-T and ISO in 2009, which makes it the most recent still image coding standard. The original scope was to develop a coding scheme targeting “extended range” applications which involves higher bit-depths as currently supported. However, much more than 10 years after JPEG2000 development and 10 years after its standardisation it seems to be reasonable to look for a new coding standard to eventually employ “lessons learnt” in JPEG2000 standardisation. In particular, the focus is on a simpler scheme which should offer only the amount of scalability actually required for most applications (as opposed to JPEG2000 which is a rather complex scheme offering almost unconstraint scalability). JPEG XR shares many properties with JPEG and JPEG2000 but exhibits also elements of the recent H.264 video standardisation. JPEG XR is a classical transform coding scheme showing the classical three-stage design: transform, quantisation, and entropy encoding. JPEG XR supports lossless to lossy compression. Up to 32 bits per color channel are supported, lossless color conversion for RGB and CMYK is specified, e.g.

$$V = B - R$$
$$U = R - G - \lceil \frac{V}{2} \rceil$$
$$Y = G + \lfloor \frac{U}{2} \rfloor.$$

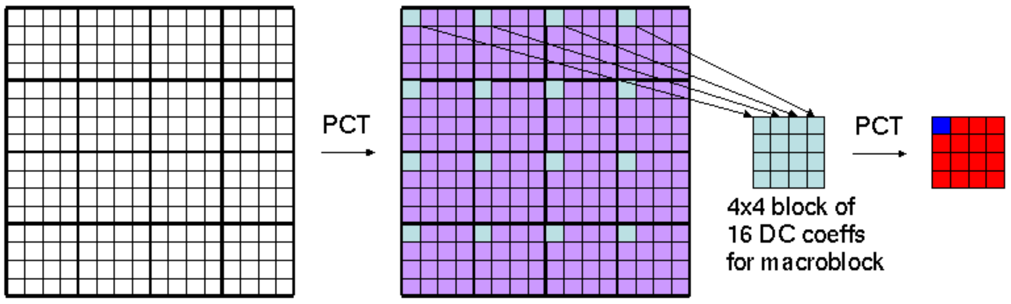


Similar to JPEG2000, larger image data can be partitioned into smaller tiles. Each tile is composed of macroblocks, the tiles at the images’ edge might be padded to exactly accomodate an integer number of macroblocks. Macroblocks themselves consist of 16 blocks which accomodate 16 pixels each. Note that these smallest building blocks are smaller as compared to JPEG (8 x 8 pixel blocks) and cannot be adaptively recombined into larger blocks as possible in H.264.

The transform applied in JPEG XR is inspired by the DCT, however, it is an integer-based transform allowing lossless reconstruction. The entity the transform is applied to is a macroblock, however, the first



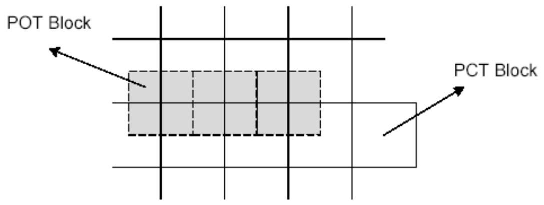
stage is to apply the transform to all 4 x 4 pixels blocks of a macroblock. Subsequently, the resulting coefficients are partitioned into 240 “high pass (HP) coefficients” and 16 coefficients corresponding to the lowest frequency in each block. The latter are aggregated into a square data layout (4 x 4 coefficients) onto which the transform is applied for a second time. The result are 15 “low pass (LP) coefficients” and a single “DC” coefficient (per macroblock). It is interesting to note that the concept of recursively applying a filtering operation is “borrowed” from the wavelet transform. Obviously, this also corresponds to three scalability layers: DC, LP, and HP coefficients, similar to the scans being built in the spectral selection JPEG progressive mode.

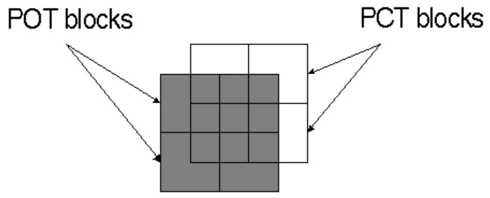


In fact, the transform used in JPEG XR is more complicated as compared to JPEG, it is a so-called “two-stage lapped biorthogonal transform (LBT)” which is actually composed of two distinct transforms: The Photo Core Transform (PCT) and the Photo Overlap Transform (POT). The PCT is similar to the widely used DCT and exploits spatial correlation within the 4 x 4 pixels block, however, it suffers from the inability to exploit inter-block correlations and from blocking artifacts at low bitrates. The POT is designed to exploit correlations accros block boundaries as well as mitigate blocking artifacts. If those transforms are concatenated appropriately, good coding performance at low computational complexity is achieved, the hierarchical employment further improves coding performance.

Each stage of the transform can be viewed as a flexible concatenation of POT and PCT since the POT is functionally independent of the PCT and can be switched on or off, as chosen by the encoder (this is signalled by the encoder in the bitstream). There are three options: disabled for both PCT stages, enabled for the first PCT stage but disabled for the second PCT stage, or enabled for both PCT stages.

The regions of support for the PCT and POT consist of 4 $\times$ 4 block regions that are offset relative to each other by 2 pixels in each dimension as shown in the figure. Overlap operators at both levels can be





turned off to minimize ringing artifacts related to long filters, as well as to enable a very low complexity decoding mode. Overlap operators at both levels can be turned on at very low bit rates to mitigate blocking artifacts.

The PCT and POT are composed of several elementary transforms, out of which the Hadamard transform is the most well known one. All these transforms are integer based, i.e. perfect reconstruction is guaranteed by this design. PCT is almost equivalent to a DCT – when transforming an image by DCT, the inverse PCT gives a very good approximation of the original image and vice versa.

Quantisation is different to JPEG – a common quantisation factor is used for all HP and LP coefficients inside a macroblock, but these quantisation factors can be different for each macroblock, color bands, and tiles. The quantised value is obtained by dividing the coefficient by the quantisation parameter and rounding to integer. For the lossless mode, quantisation parameter is 1.

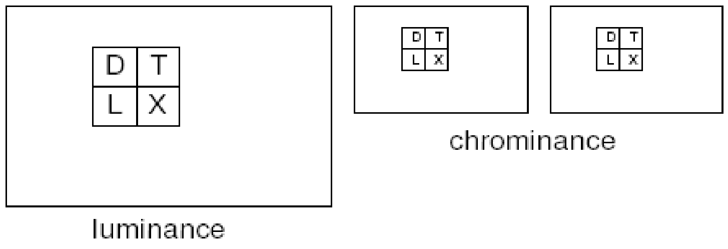
Whereas in JPEG only DC prediction is applied, in JPEG XR three types of prediction are applicable: DC prediction, LP prediction, and HP prediction. The DC coefficient of a macroblock (yes, there is only a single one !) can be predicted in one of the following modes:

- Prediction from left neighbor
- Prediction from top neighbor
- Prediction from left and top neighbors
- no prediction

In order to determine the prediction mode, the differences between the DCs of the top-left and top macroblocks and those between the DCs of the top-left and left macroblocks are computed. Only in case the difference in one direction is significantly larger than the other (by a factor of 4), the direction of the smaller difference is predicted, otherwise prediction from both neighbors is chosen. In case the error is too large, no prediction is applied.

The first row or first column of the LP coefficients can be predicted from adjacent macroblocks as well. Options are:

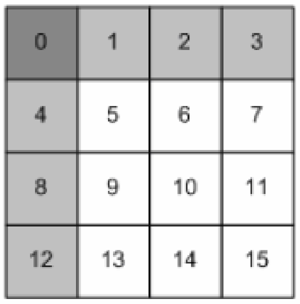
- Prediction of first column from left
- Prediction of first row from top
- no prediction

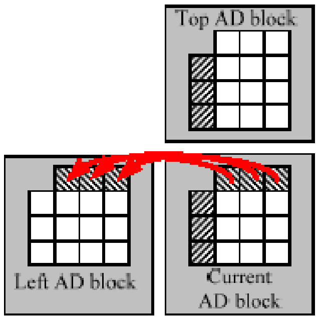


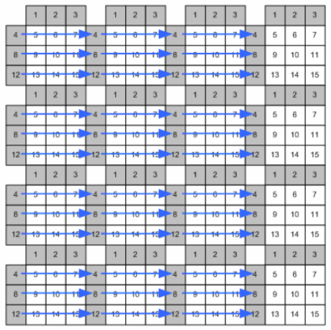
LP prediction is only applied in case of top OR left prediction of the DC coefficient and matches the prediction direction (and only in case the quantisation parameter of the involved macroblocks matches). In all other cases there is no LP prediction from adjacent macroblocks.

The first row or first column of HP coefficients in each block is predicted in HP prediction. Options are identical to LP prediction, however it has to be noted that HP prediction is only applied from the blocks in the same macroblock. If an individual block does not have a neighbor in the direction specified by prediction mode, no prediction is performed for that block.

For each macroblock, the HP prediction mode is derived from the energy of the LP coefficients of that macroblock. If the energy of the first column of LP coefficients is much smaller than the energy of the first row of LP coefficients, prediction from top is chosen as the HP prediction mode. If the energy of the first row of LP coefficients is much smaller than the corresponding energy of the first column, prediction from left is chosen. Otherwise, no prediction is performed. The energy of the LP coefficients in the chroma channel is also used in deriving HP prediction mode.



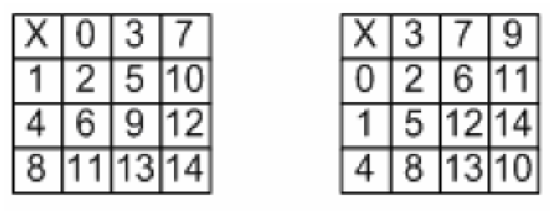




Contrasting to JPEG, there is no fixed pattern to scan the 2-D coefficient array into a 1-D vector but adaptive coefficient scanning is applied. In the figure the initial scan patterns for LP and HP coefficients (horizontal) and HP (vertical), respectively, are shown.

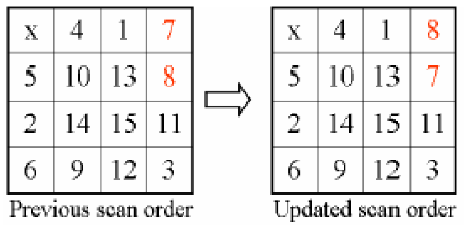
Scan patterns are adapted dynamically based on the local statistics of coded coefficients. Coefficients with higher probability of non-zero values are scanned earlier, i.e. the corresponding scan order is adapted.

This is done by successively updating lists where the number of non-zero values is recorded and where the coefficients are sorted in decreasing order according to the number of observed non-zero values. As soon as the ordering get incorrect locally, a local bubble-sort exchange between two coefficients' positions takes



X	0	3	7
1	2	5	10
4	6	9	12
8	11	13	14

X	3	7	9
0	2	6	11
1	5	12	14
4	8	13	10



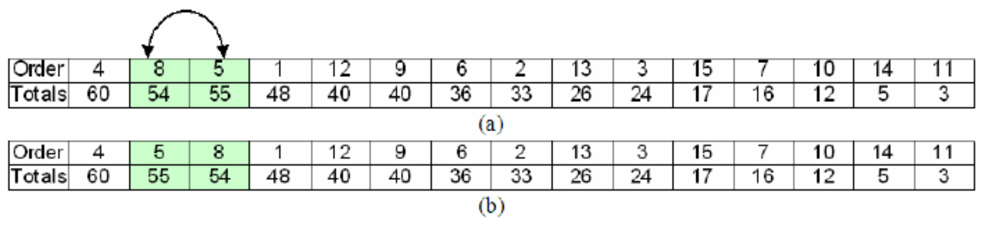
x	4	1	7
5	10	13	8
2	14	15	11
6	9	12	3

Previous scan order

x	4	1	8
5	10	13	7
2	14	15	11
6	9	12	3

Updated scan order

place. Note that the decoder needs to be in synch (has to generate the same statistical information during decoding) to be able to perform the corresponding operations.



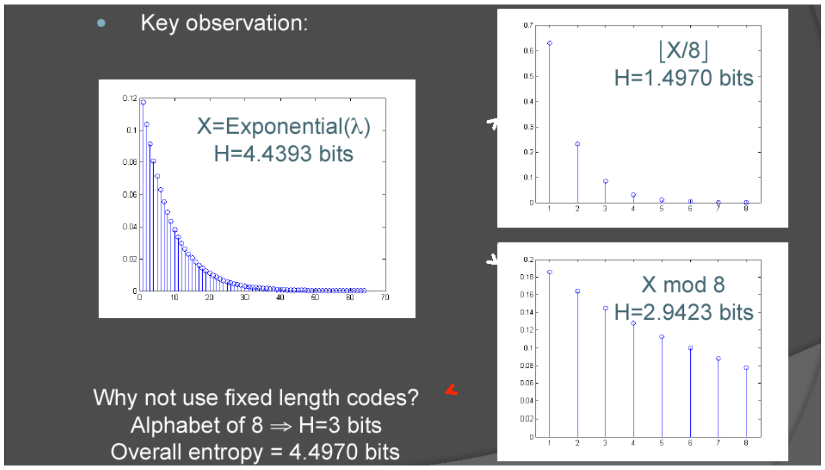
Order	4	8	5	1	12	9	6	2	13	3	15	7	10	14	11
Totals	60	54	55	48	40	40	36	33	26	24	17	16	12	5	3

(a)

Order	4	5	8	1	12	9	6	2	13	3	15	7	10	14	11
Totals	60	55	54	48	40	40	36	33	26	24	17	16	12	5	3

(b)

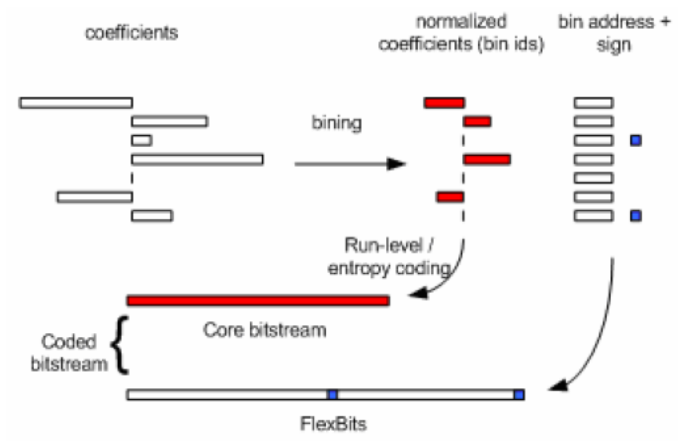
The entropy coding stage has also some similarities to techniques in JPEG, however, also here the concept of adaptivity is used. At first, it is observed that coefficient data can be partitioned into most significant bits and bits of lower significance as shown in the figure. The entropy of the more significant bits is low and these can be efficiently encoded using a VLC. On the other hand, the bits with lower significance exhibit a rather high entropy (they are somewhat close to noisy data as we already know) so that VLC does not work well and FLC is applied. These data is termed "Flex Bits".



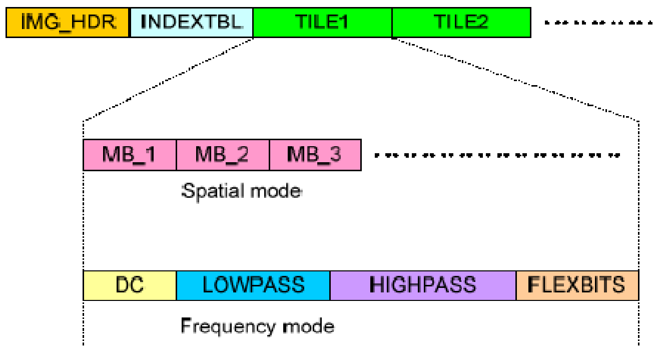
- Key observation:

Why not use fixed length codes?
Alphabet of 8 ⇒ H=3 bits
Overall entropy = 4.4970 bits

The entire process is termed "adaptive coefficient renormalisation" and is pretty similar to the JPEG concept of category and index in the category, but in JPEG XR these data are separated and not encoded in adjacent manner. The actual VLC encoding is done by coding the coefficient "category" and trailing zeros for all but the first coefficient, where also preceeding zero-runs are recorded, and the last coefficient, where a "last" symbol is coded. Different tables are used depending on the required categories and and switched in an adaptive manner to optimize coding gain.



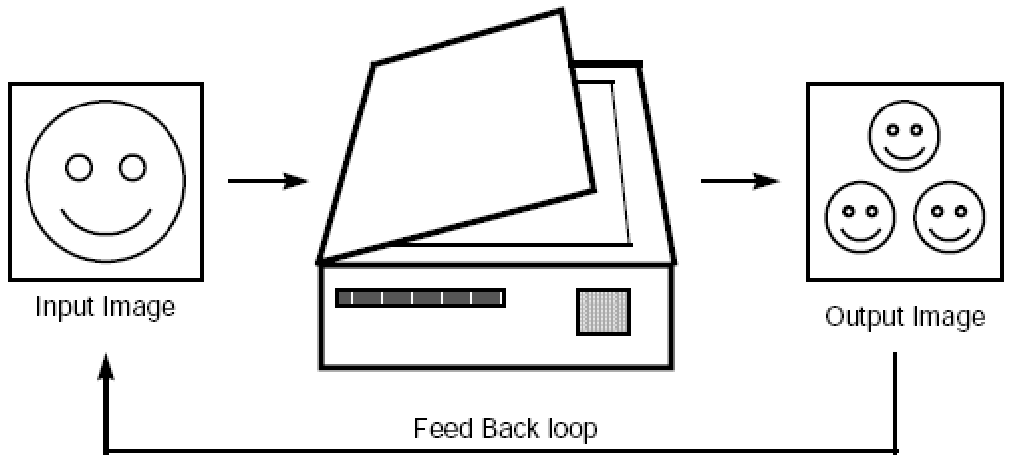
The JPEG XR bitstream can be assembled in two modes, i.e. "spatial mode" and "frequency mode". In both modes, the main header is followed by eventual index tables and other generic parameters. In spatial mode, the macroblocks are arranged one after the other (similar to JPEG baseline coding mode). In frequency mode, first all DCs are assambled, followed by LPs, HPs, and finally flexbits. Frequency mode corresponds more to spectral selection in progressive JPEG with 4 scans only.



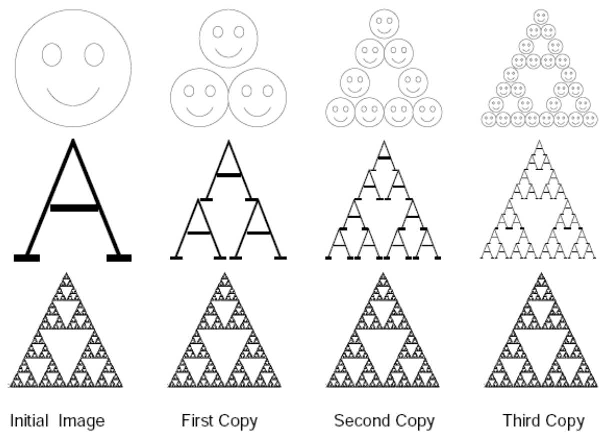
3.2.4 Fractal Image Compression

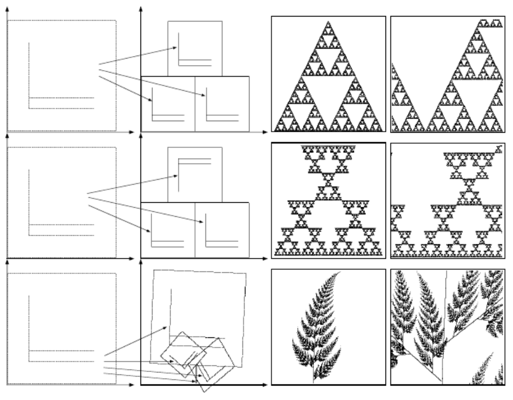
Imagine you have a copymachine that acts like the one in the figure: upon input of an image, the image is downscaled, and copied three times to different positions of the resulting image. This procedure can be repeated again and again.

When you apply this process repeatedly using always the same copy instruction as shown, the result is always the same "Sierpinski triangle", no matter which image you take as a starting point. Once the copy rules are changed (i.e. different functions are used in the copy machine to downscale and displace the image material), this process terminates in different final images, depending only on the transformations

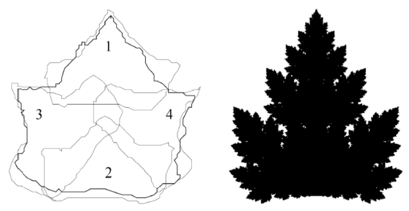


being applied, but not on the starting image. The copy machine simple means that the transformations are applied to a start image iteratively.





The transformations involved typically are *affine transforms* of the form given in the figure where (e_i, f_i) represent a translation vector and (a_i, b_i, c_i, d_i) form a matrix which is responsible for scaling / rotating / flipping the data. For example, the three transformations in the initial copy machine are defined by: $a_i = d_i = 1/2$, $b_i = c_i = 0$, (which means that the matrix is identical for all three but the translation vector is different), $e_1 = f_1 = 0$, $e_2 = 0$, $f_2 = 1/2$, $e_3 = f_3 = 1/2$.



$$v_i(x,y) = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix}$$

Not all existing transformations do result in a stable result (called *attractor*) after a couple of iterations – the transformations applied need to be "contractive", i.e. a given transformation applied to any two points in the input image must bring them closer together. This is a natural condition, since otherwise the points would spread out and the attractor would be of infinite size.

A transformation v_i is said to be contractive if for any two points P_1, P_2 the distance

$$d(v_i(P_1), v_i(P_2)) < s\ d(P_1, P_2)$$

for $s < 1$. The Euclidian metric serves well in our case. The example matrix in our three transformations once applied to any initial point (x, y) will result in $(1/2x, 1/2y)$, $(1/4x, 1/4y)$, etc. obviously satisfying the condition for contractivity. The "Banach Fixed Point Theorem" finally states that any contractive mapping in a reasonable space once applied iteratively to an arbitrary start value converges to a unique fixed point. This is why the copy machine always generates the same image after some iteration idependent of the start image used.

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

Now what has this to do with image compression and why do we use the term fractal here ? First, if we consider these attractors we notice that these are complicated geometrical structures, which are generated by very simple transformations. For the Sierpinski triangle, only three transformations are required, each of which is determined by 6 number, overall 18 numbers. In terms of compression, we need to store only 18 numbers to generate an attractor of arbitrary resolution (!) since we can start with any arbitrary start image (the resolution of which determines the final resolution of the attractor image). This is hell of a compression ratio !! Its called fractal compression since the attractors generated from these transformations (which are called iterated function systems IFS) are fractals (e.g. they are self similar). However, when we want to apply compression we do not generate an image from its description (this is decoding) but we have the image given and are looking for its compact description – this is called *inverse problem* in fractal compression. So, given an attractor we are looking for the IFS that converges to it. We have seen how this can be done: find global selfsimilarities in the image and identify the corresponding transformations (the theory behind it is the "collage theorem" which bounds the accuracy of the IFS once we have found approximate self-similarity based descriptions of the image.

But how does this relate to images ? Common images we want to compress are no fractals (no self-similar geometric structures) ! We need to generalise the concept of IFS to partitioned IFS (PIFS), where the concept of self-similarity is relaxed to a localised one: only parts of the image are self-similar to other parts of the image data, transformations are only applied to parts of the image. All parts of the image are generated by transformations, the set of all transformations is called PIFS. In order to converge to an attractor, the PIFS needs to be contractive of course. In terms of the copy machine, for each transformation only a specific part of the image is accessible, the other parts are covered.

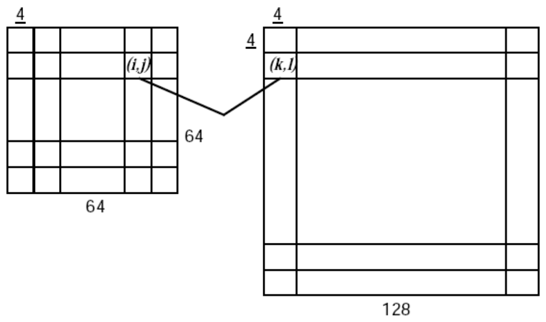
When treating images, it gets obvious that not only geometric simlarity is important but also gray scale similarity. Therefore, in the transforms used for grayscale images also gray scales are manipulated as shown in the figure (where (x, y) are the spatial coordinates and z is the grayscale): s_i is a multiplicative factor controling contrast and o_i is an additive element controling brightness. In terms of the original geometrical transformation v_i we may state that this map defines how the partitioned domains of an original are mapped to the copy, while o_i and s_i determine contrast and brightness modification of the transform.

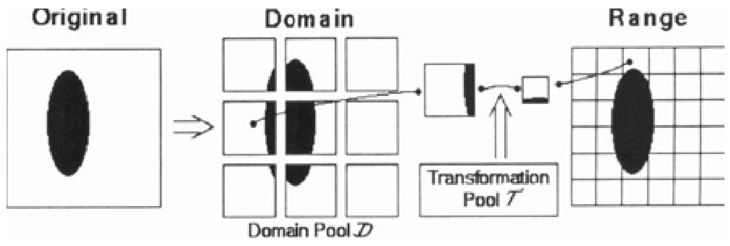
Now having seen what PIFS mean in the context of images, we are facing the inverse problem when we want to compress an image with the fractal technique. In the following we describe how the local transformations are found given an image to be compressed.

We take two copies of the image, one of which is downscaled to a quater of the original size. Subsequently, we partition both images into equal-sized tiles: the first copy, having original size, is tiled into non-overlapping blocks (the ranges), while the second copy can have non-overlapping, highly overlapping, or arbitrary organised tiles (the domains) of the same size (note that due to downscaling, the corresponding areas in the original images have four times the size).

In order to solve the inverse problem, we need to find for each range a transformation in side the image, where the range is the corresponding attractor (we want to generate the range as part of the image by an IFS). When having done this for all ranges, the set of all IFS constitutes the PIFS we are looking for, i.e. each range can be generated by a transformation.

For each range, we look through the set of all domains (the downscaling has already shrunk the corresponding areas such that the transformation is contractive anyway) by considering all 8 isometries for each domain (rotations and mirrored rotations), for each isometry optimizing o_i and s_i.





Given two squares containing n pixel intensities $d_1, \ldots, d_n$ (domain pixels) and $r_1, \ldots, r_n$ (range pixels), the aim is to select o_i and s_i to minimize the quantity

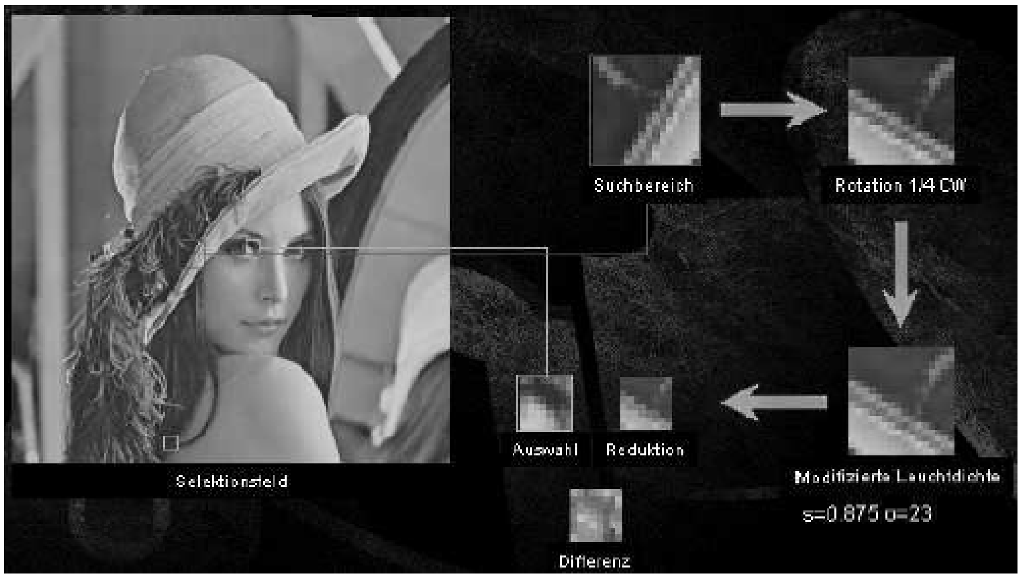
$$R_i = \sum_{j=1}^{n} (s_i d_j + o_i - r_j)^2 \ .$$

The minimum is attained when the partial derivatives with respect to o_i and s_i are zero, which occurs when

$$s_i = \left[n^2(\sum_{j=1}^{n} d_j r_j) - (\sum_{j=1}^{n} d_j)(\sum_{j=1}^{n} r_j) \right] / \left[n^2 \sum_{j=1}^{n} d_i^2 - (\sum_{j=1}^{n} d_i)^2 \right]$$

and

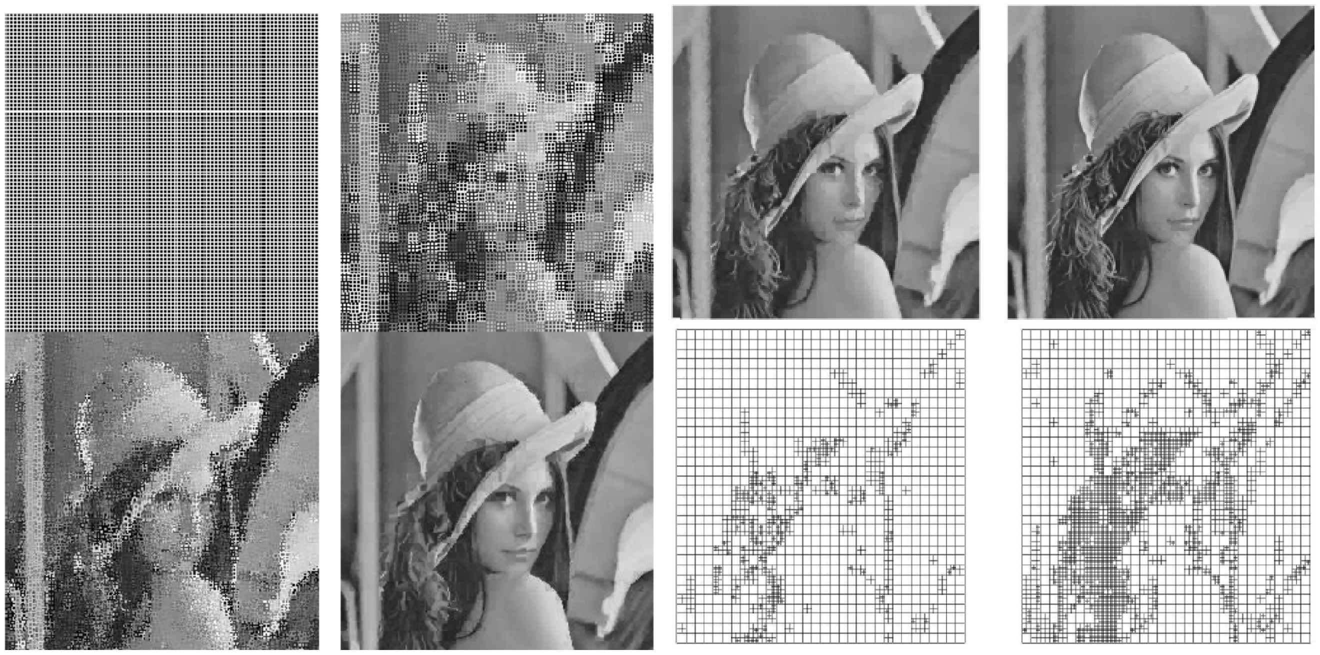
$$o_i = \left[\sum_{j=1}^{n} r_i - s_i \sum_{j=1}^{n} d_i \right] /n^2 \ .$$



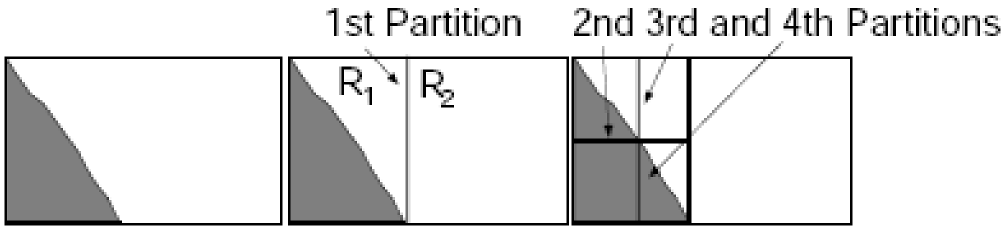
So we get an R_i for each domain and select the domain with the smallest value. What do we need to store per range to be able to generate it as an attractor of the corresponding transformation: the position (or the index) of the domain and its isometry (which gives the geometrical part of the transform since the downscaling already provides contractivity) and the computed values for o_i and s_i.

Decoding is a very simple procedure: we simply apply the transformations stored to an arbitrary start image until convegence has been reached, usually not more than 10 iterations are required for decoding.

So far, there is no way to control quality or bitrate apart from the selection of the size of the ranges (large ranges will lead to high compression ratio but low quality). We can apply a threshold to the best R_i found for a range: if the error is too high, we not store the transformation parameters but partition the range into its four quadrants and put them on a "waiting list". After all ranges of the inital range-size have been processed, also the domain-pool is partitioned into according smaller tiles and the process is repeated. Again, there may be a threshold used for controling quality and already partitioned ranges may be partitioned further. It has to be noted that one partitioning increases the bitrate for a range by a factor of four, since four instead of one transformations (and their parameters) need to be stored. If ranges get



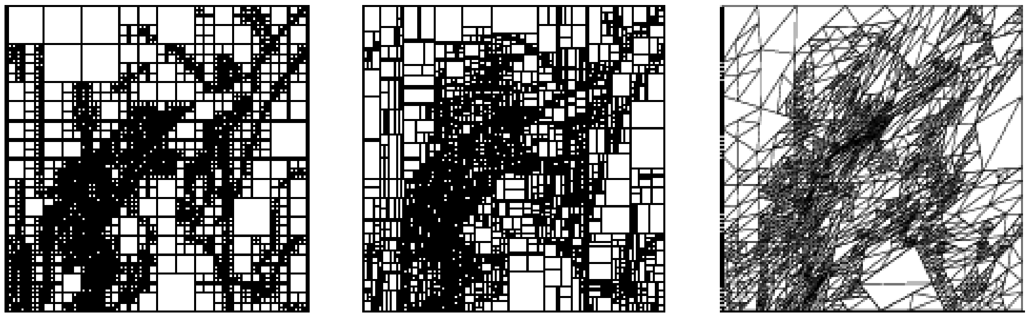
very small, it is less expensive to store them directly instead of storing transformations. Of course, o_i and s_i can be subjected to quantisation reducing the bitrate.

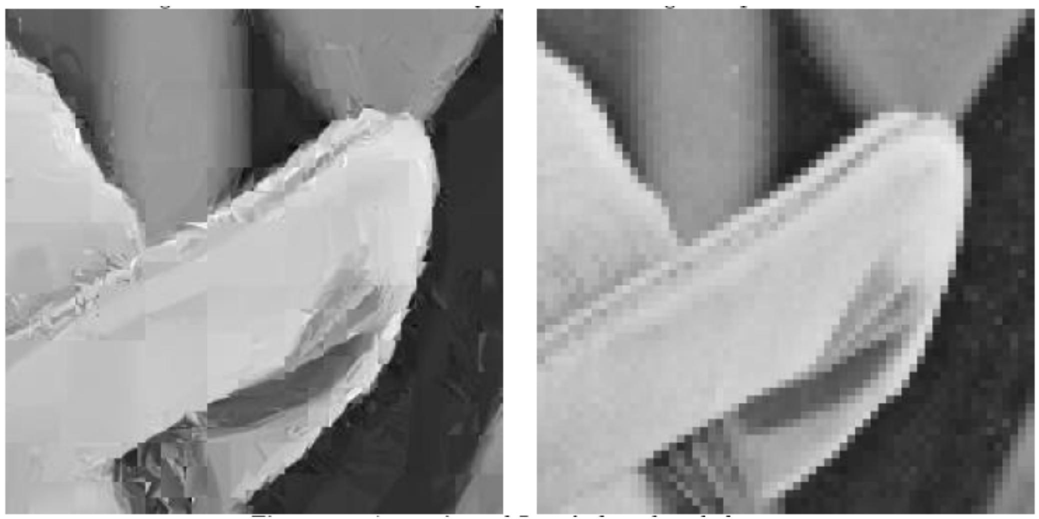


Quadtree partitioning leads to blocking artifacts mainly in homogeneous areas in case edges cross block borders. Therefore, other partitioning schemes have been developed which are better able to adapt to edge directions like the HV and triangulation based schemes. However, the coding cost for domain information increases with more complex partitioning schemes.

The coding quality (rate / distortion quality of fractal coding is typically superior to JPEG at low bitrates but inferior at high bitrates. When compared to JPEG2000 or JPEG XR, fractal coding is inferior across the entire range of possible bitrates. Therefore, it is hard to come up with sensible application scenarios. On eof the most interesting aspects is that of fractal interpolation. As decoding a fractal file can be done with virtually arbitrary resolution data, the decoding process generates a huge amount of artificial "fractal data" which is of highly non-linear nature and looks quite pleasing as compared to standard intrapolation techniques due to the inherent irregularity. ANother interesting aspect is the high decoding speed.

Fractal compression suffers from very high encoding runtime, it is highly assymetric and therefore only suited for retrieval-based applicaitons. The complexity of encoding is determined by the amount of overlap-



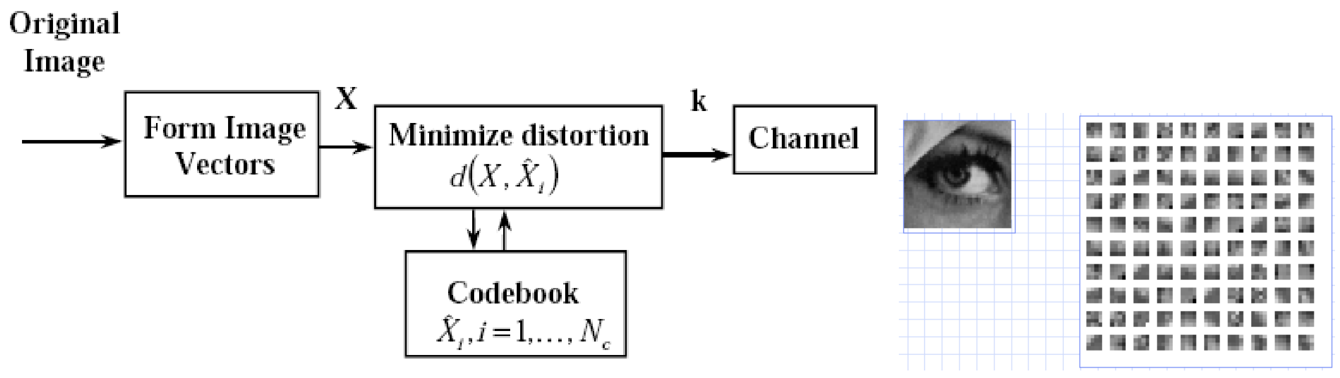


ping of the domainpool. The size of the domainpool also affects R / D performance: the more domains are available, the better will the obtained match be. On the other hand, a large set of domains requires a larger pointer to address (i.e. store) the pposition of the domain.

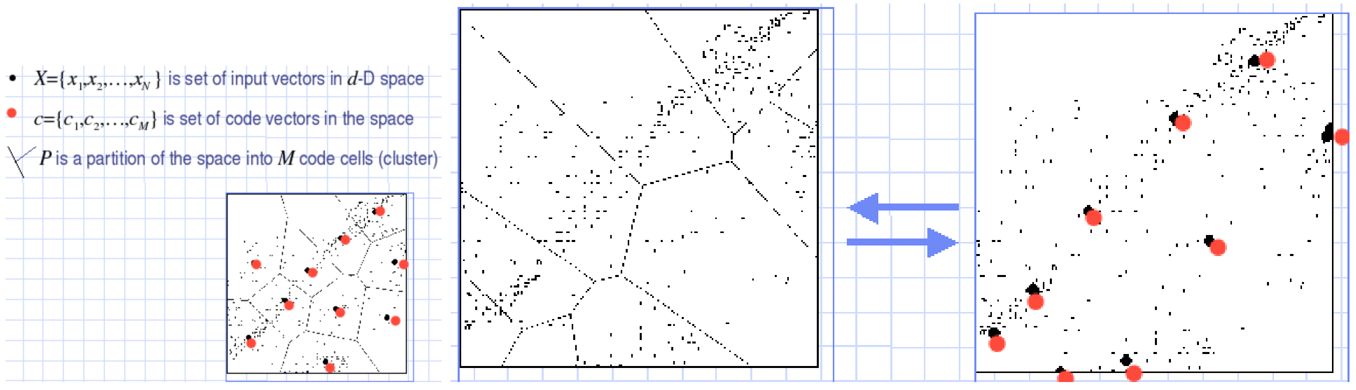
Significant work has been done to accelerate the encoding speed of fractal compression. Some techniques ("geometric search") use the assumption that domains spatially close to the position of a range have a higher probability to be similar, therefore, the number of domains considered is higher closer to the range center. Other approaches are based on a classification of ranges and domains. Domains are assigned to different classes based on some criterion. For a range domain match, first the range is classified as well, and subsequently only domains of the same class are considered. Due to the adaptation of contrast and brithness in the matching process, a classification following the average block-luminance or similar does not make sense. As an example, a block can be subdivided into four quadrants and for each quadrant the average brightness is computed. Finally, the ordering of the four averages defines different classes, the number of which can be increased further by additionally considering brightness variance for example. Speed is enhanced significantly, but matching quality is also reduced to some extent.

3.2.5 Vector Quantization for Image Compression

The original image is decomposed into n-dimensional vectors (either pixel blocks or a 3-dimensional vector formed from colour components. Both the encode and the decoder share a previously generated codebook consisting of vectors of the same dimensionality. Each vector from the image is compared to the entries in the codebook and the most similar one is identified (based on some error metric). Subsequently, the index of the most similar codevector is stored for the entire block. Note that there are some interesting properties shared by VQ and fractal compression. In both techniques, for each block the most similar block from a collection is searched (in the case of fractal coding with additional adaptation). However, the important difference is that VQ uses an external codebook whereas fractal coding uses the image itself as “internal” codebook. A further similarity is that also in VQ, adaptive partitioning can be used to control quality and that similar strategies are used to accelerate coding speed.

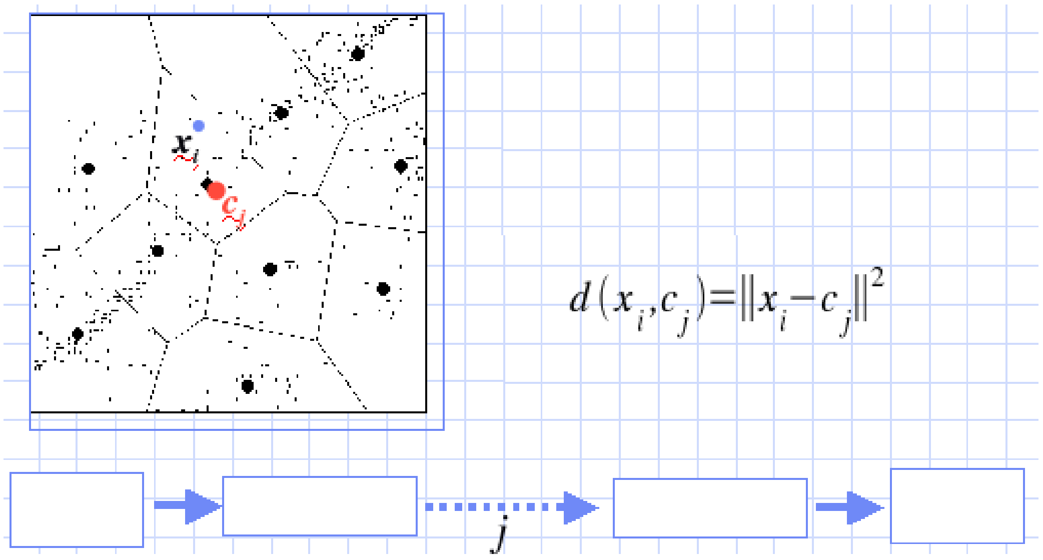

Original Image
Form Image Vectors
X
Minimize distortion $d(X,\hat{X}_i)$
k
Channel
Codebook $\hat{X}_i, i=1,\dots,N_c$

Decoding is even simpler as compared to fractal coding: the indices in the file are mapped back to their corresponding codebook vectors and these vectors are inserted into the reconstructed image. This lookup-table operation is obviously very fast.

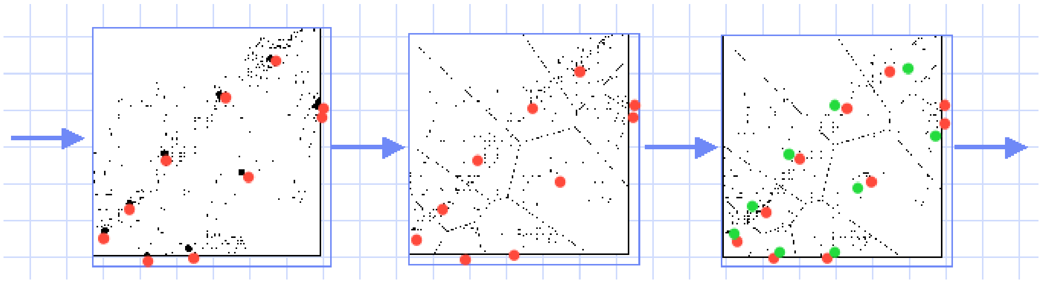

$X=\{x_1,x_2,\dots,x_N\}$ is set of input vectors in d-D space
$c=\{c_1,c_2,\dots,c_M\}$ is set of code vectors in the space
P is a partition of the space into M code cells (cluster)

Vector quantisation transforms the vectors of the input image into indices that represent clusters of vectors.

In the coding process, a data vector is mapped to the representative codebook vector for the cluster the block is situated in. The corresponding codebook vector has the minimum distance to the data vector.



VQ can be used to tailor the codebook to a specific application context. For this purpose, a set of training vectors (representing the target application context) is used to construct a corresponding codebook. The most prominent algorithm is the "Linde-Buzo-Gray (LBG)" procedure. We start with a set of training vectors, some initial codebook, a distortion measure d, and a fractional distortion change threshold ϵ. We set the iteration counter I to 1 and initialize the average distortion over all training vectors D^0 to a large number. If $\frac{D^{I-1}-D^I}{D^{I-1}} \leq \epsilon$, the algorithm terminates. Subsequently the codebook is updated by replacing the codevector of each decision region (which is the region containing the trainingsvectors that have been mapped to a single codevector) by a codevector that minimizes the quantisation error of that region. For MSE, the centroid (average) of the training vectors is that vector.



The centroid is found by computing the mean of the corresponding components of the training vectors. Now we increase the iteratin counter and re-evaluate the above termination criterion. A fundamental issue for this algorithm is the design of the initial codebook. This can be done by starting right away with a random codebook of the desired size or by applying the "splitting" techniques which starts with the overall centroid of the trainingsset, which is split into two parts, applied the LBG algorithm, splits again and so on until the desired size of the codebook is reached.

By analogy to fractal compression there exist several techniques to accelerate the encoding process like classified VQ or tree structured codebooks. Also many variants exist like mean/residual VQ (after the formation of image blocks the mean is computed for each block and stored in quantised manner (using eg.

DPCM); the quantsed means are substracted from the corresponding image vectors to yield a residual vector with zero mean, which is processed with VQ using a suited residual codebook) or interpolative / residual VQ (which is similar to the technique described before except that instead of the mean, a severely downsampled version of the block is stored and interpolation is used instead of the replication of the mean over the entire block).

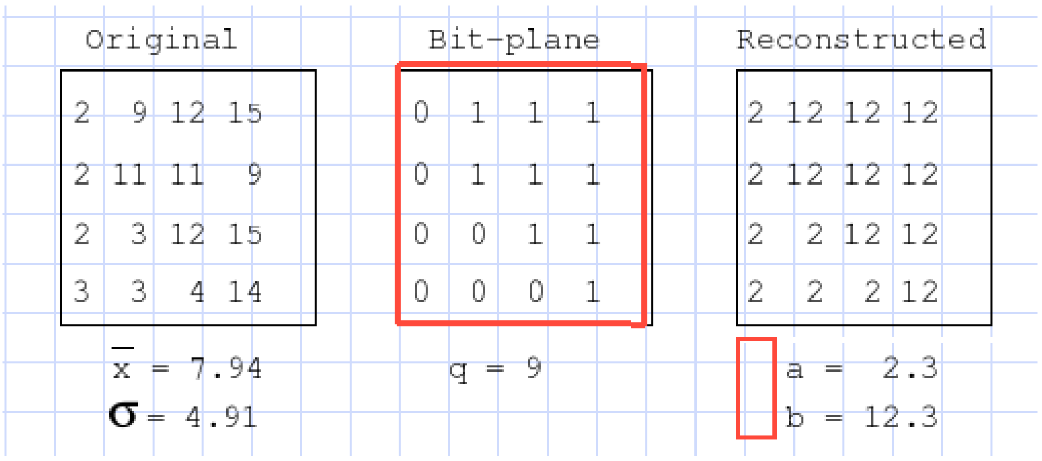
VQ never made it into still image standardisation due to the significant assymetry of the process and the lower R / D performance as compared to competitors.

3.2.6 Block Truncation Coding

An image is divided into squared blocks of some fixed size (typically 4×4 pixels in non-adaptive schemes. For each block, the mean $\hat{x}$ and standard deviation σ is computed. The idea is to construct a compression scheme which preserves mean and standard deviation on a block basis. A two level quantiser is designed which sets all pixels to 1 which are larger than the mean, the others are set to 0. The number of pixels in the block is m, the number of pixels larger than the mean is q. The bitplane is stored together with the mean $\hat{x}$ and standard deviation σ.

Reconstruction is done by setting all 1's to a value b and all 0's to a, which guarantees the abovementioned preservation:

$$a = \hat{x} - \sigma\sqrt{\frac{q}{m-q}} \quad \text{and} \quad b = \hat{x} + \sigma\sqrt{\frac{m-q}{q}}$$



Original

2	9	12	15
2	11	11	9
2	3	12	15
3	3	4	14

$\bar{x}$ = 7.94
σ = 4.91

Bit-plane

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

q = 9

Reconstructed

2	12	12	12
2	12	12	12
2	2	12	12
2	2	2	12

a = 2.3
b = 12.3

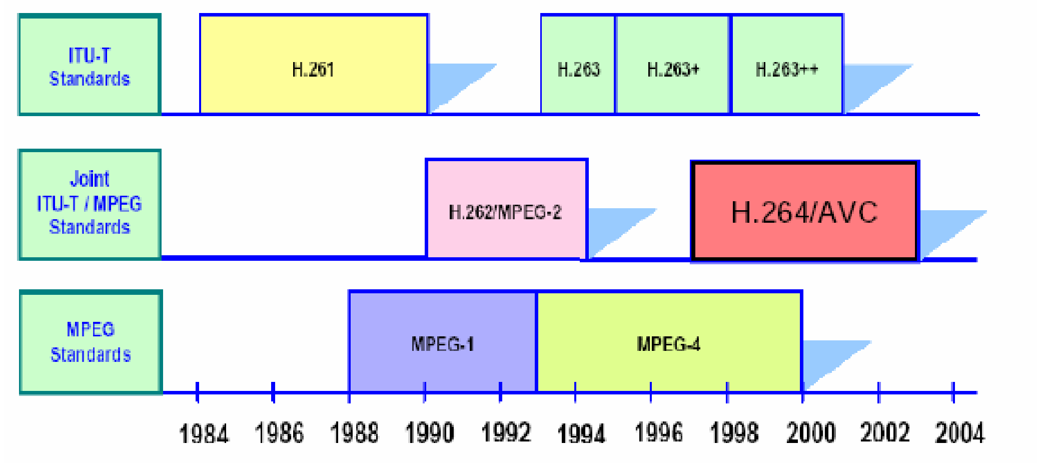
Absolut moment BTC is a variant which replaces the standard deviation by the first absolute central moment (which is the sum of the absolute differences from the mean, devided by the number of pixels), which is then preserved together with the mean.

Similar to fractal coding and VQ, BTC can be used in an adaptive manner by starting with a larger block size and in case of too large errors block partitioning is applied. BTC is very fast but not competitive in terms of R / D performance.

Kapitel 4

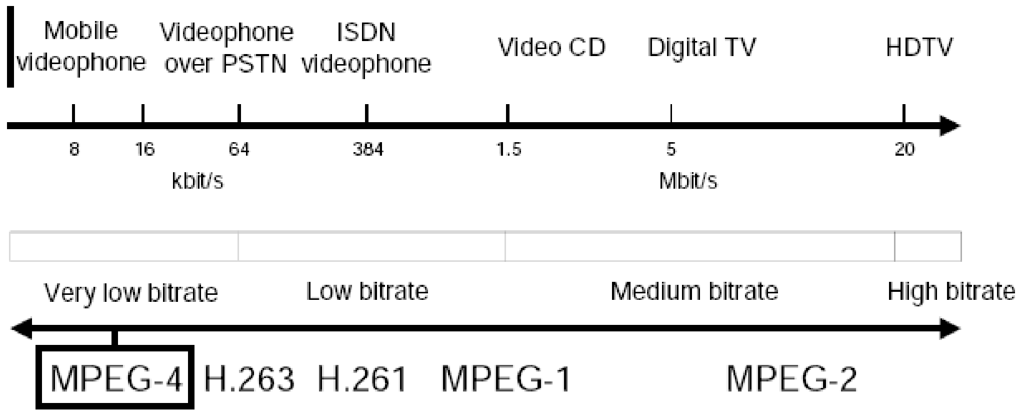
Video Compression

In video compression there are no dedicated techniques which allow for lossless compression. Therefore, we do not discuss lossless techniques here. In video data, we have additional temporal correlation which can be exploited (of course, video frames samples at 60 frames per second tend to be highly similar which calls for exploitation). However, not all techniques / standards do actually support exploitation of the third dimension by inter-frame compression techniques but support only intra frame compression. These techniques will be described first.



The figure describes major standards as issued by the two major standardasation bodies in the area, i.e. ISO/IEC MPEG (responsible mainly for the entertainment industry) and ITU-T (international telecommunication union, responsible for telecommunication industry). Standardisation started in 1985. It is clearly visible that the work on a single standard can extend over a significant amount of years. In most cases, standards are completed part by part and several amendments are issued before a standard is actually completed. The so-called “Joint Video Team” (JVT), a joint effort of ISO and ITU-T, is responsible for the latest standardisation in the area (H.264 / MPEG-4 AVC and SVC).

The different standards have not only been developed at different points in time, they have also different target application areas. The figure compares application areas and target bitrates. Only the more recent standards are meant to cover a wider range of bitrates (and application domains). This is usually accomplished by introducing the concept of *profiles* and *levels* in MPEG-2,4.



To illustrate the need for compression of video data, let us consider HDTV: with 1080 lines having 1920 pixels each at a frame rate of 60 frames / sec in three colour bands this results in 337 Mbytes/sec. Obviously, this calls for a reduction of the bitrate.

4.1 Motion JPEG and Motion JPEG2000

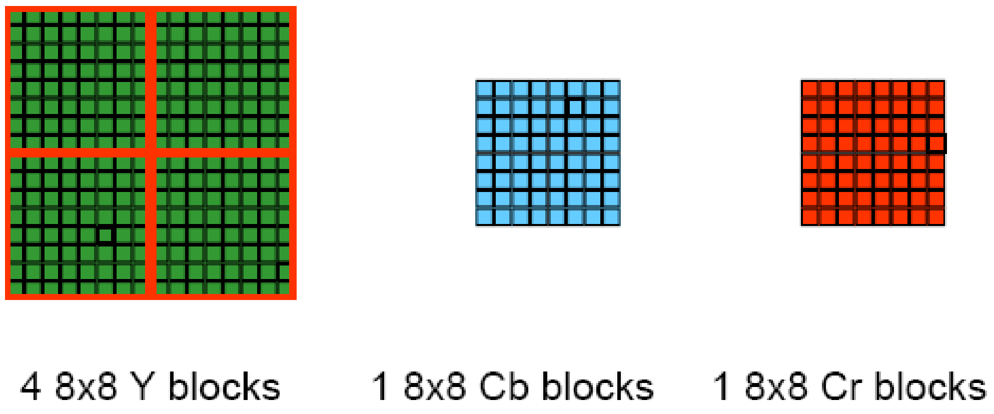
MJPEG and MJPEG2000 perform a frame-by-frame compression of video data (intra-frame compression) and provide synchronisation structures for associated audio. While being inferior in terms of achieved compression ratio to dedicated video coding standards, these technqiues are required in application where random access is needed for each single frame (e.g. video editing applications). MJPEG2000 is of course capable of performing lossless compression due to the corresponding capabilities of the JPEG2000 compression engine. The Digital Cinema Initiative (DCI) employs MJPEG2000 due to its higher quality as compared to other standards, the higher bitrate is simply not important in its application domain.

4.2 MPEG-1

MPEG stands for “motion picture experts group”, a ISO standardisation body resonsible for standardisation in the area of video entertainment systems. While being most well known in the area of video compression (MPEG-1,-2,-4), also standards in the area of multimedia metadata descriptions for searching multimedial content (MPEG-7) and in the area of protection and adaptation of multimedia items (MPEG-21) have been developed.

MPEG-1 (ISO 11172) consists of 5 main parts: systems (multiplexing audio and video and representation of time information for synchronisation), video (which is described in the following), audio (which is covered in the audio chapter), compliance (specification of test procedures to test compliance of decoders to parts 1,2,3), and technical report (including software simulations of parts 1,2,3). The main media target was storage of audio and video at 1.5 Mbits / sec on CD-ROM.

MPEG-1 simultaneously supports intraframe and interframe compression. It is restricted to progressive video (classical frame-by-frame video in full resolution in each frame), 4:2:0 chroma subsampling, and 8bpp video data (per colour channel). The satndard syntax supports DCT, motion-compensated prediction, quantisation, and VLC. It is important to note that MPEG-1 does not standardise the encoder (here flexibility is possible plugging in recent improvements and developments and new algorithms) but the bitstream and the decoder. For example, motion estimation is not standardised at all (although in almost all encoders, block matching is used).



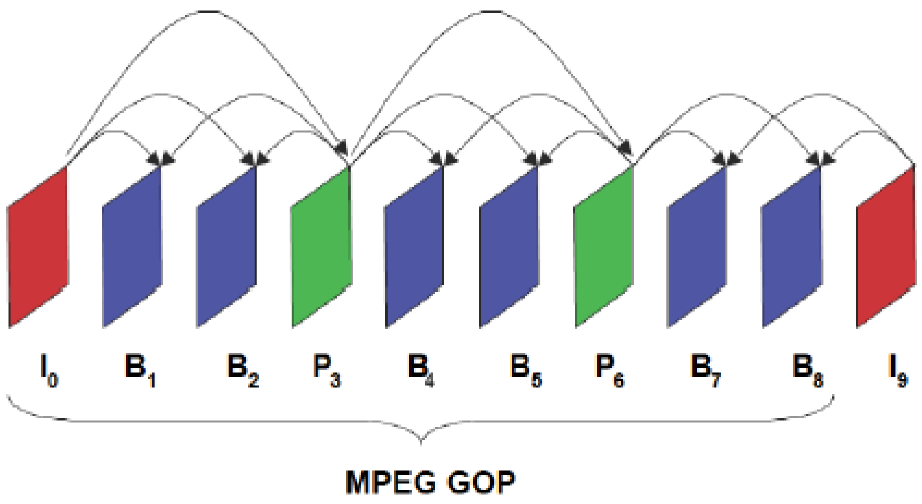
There are three different compression modes for the frames of an image sequence:

- Intraframe compression: frames marked by I (I-frames) are strictly intraframe compressed. These frame serve as random access points to the sequence and can be decoded without any reference to other frames.
- Interframe compression:
  - frames marked by P (P-frames) are encoded relative to the closest I-frame or P-frame in the past. Motion vectors and prediction errors are coded. If motion estimation fails, blocks in P-frames can be encoded in I-frame mode.
  - frames marked by B (B-frames) are encoded relative to two frames, one in the past, one in the future, bing either P- or I-frames. If motion estimation fails, blocks in P-frames can be encoded in I-frame mode.

  Both P- and B-frames cannot be decoded independently.

The relative number of I, P, and B frames depends on the target compression rate, nature of the content, and access and coding delay requirements. A group of pictures (GOP) ontains a single I frame, and an arbitrary number of P and B frames. The maximal size of a GOP is constrained to 132. There are several possibilities for GOP structures, e.g. .... I I I I I I I ...., ... I B B P B B I B B P B B I ...., ...I P I P I P I P I P .... etc.

An entity not existing in JPEG but very important in MPEG are *macroblocks*: a macroblock consists of 4 8×8 pixels block luminance data and 2 (one for each chroma channel) 8×8 pixels chroma data.



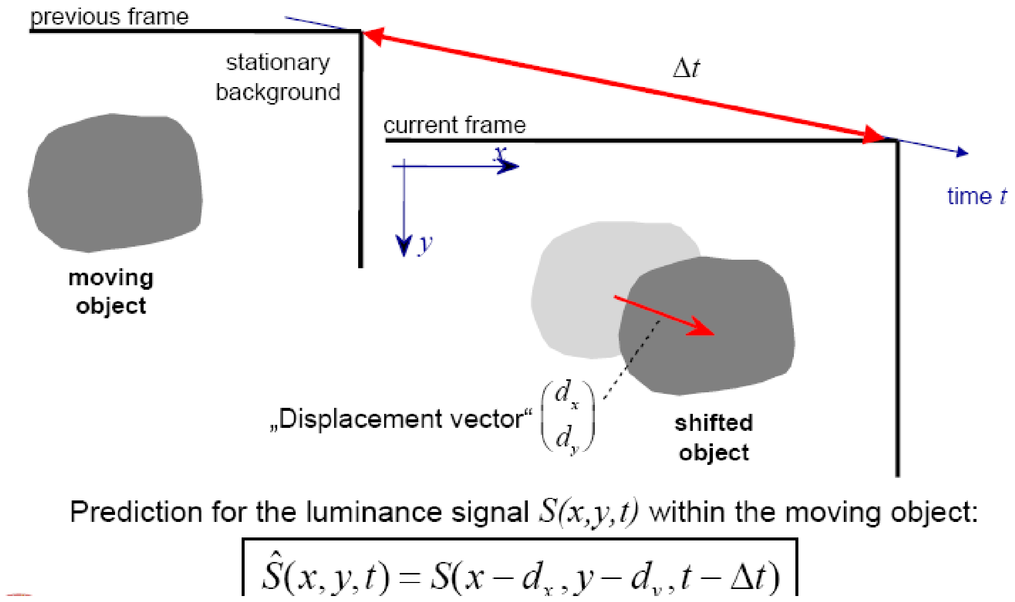
I Frames are compressed in similar manner as JPEG; the DCT is applied to 8×8 pixels blocks, quantisation is applied based on a quantisation matrix (can be user defined) which is scaled to control data rate and quality. Quantisation is followed by a zig-zag scan and a runlength / Huffman coding procedure (also based on tables which can be user defined). Redundancy among quantised DC coefficients is reduced via DPCM, coded data is separated into luminance and chrominance data.

There is one important difference as compared to JPEG: the strength of quantisation can be different on a macroblocklevel and the corresponding scale factor (MQUANT) is stored in the macroblock header. This value can be set according to spatial activity of the block or the buffer status in constant bitrate applications.

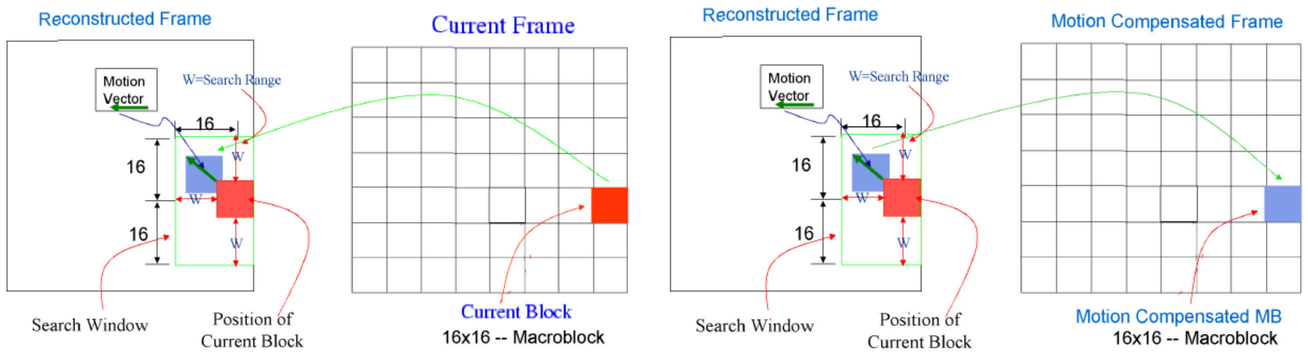
Interframe coding is based on *motion-compensated prediction* which consists of a *motion estimation* process which is followed by *motion compensation*. The most natural idea for exploiting temporal correlation is to compute the difference between adjacent frames and to only encode the difference plus addiitonal reference frames. However, this strategy fails in case of moving objects or camera movement since in these situations significant errors are introduced. The basic idea to resolve this issue is to estimate the motion that has occured between the two frames considered, i.e. in case of a moving object to estimate its displacement. Once the movement is known (encoded in a displacement or "motion vector"), we can predict the luminance signal within the moving object and incorporate this knowledge into the computation of the difference: the frame to be coded is predicted based on the luminance signal of a reference frame plus the motion information, the predicted frame is then substracted from the frame to be coded and this difference is encoded.

Motion-compensated prediction is done on the basis of macroblocks. Motion vectors are assumed to be constant across a macroblock (small scale motion is neglected as a consequence). Thus, a common motion vector is estimated and assigned to an entire macroblock (for both the luminance and chroma data). For B frames, two motion vectors are computed and stored for each macroblock. Motion vectors are encoded using DPCM.

In the following, the process of **block matching** is described which is a common choice for computing motion vectors. However, only the usage of motion vectors is standardised and any appropriate algorithm can be used to compute them (this is where recent technology can be plugged in).

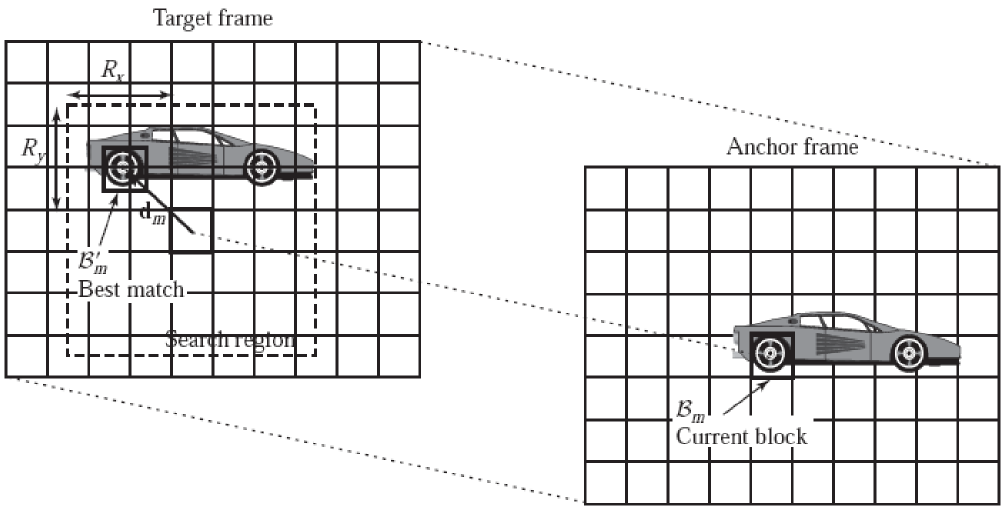

previous frame
stationary background
Δt
current frame
x
y
time t
moving object
„Displacement vector“ $\begin{pmatrix} d_x \\ d_y \end{pmatrix}$
shifted object
Prediction for the luminance signal $S(x,y,t)$ within the moving object:
$\hat{S}(x,y,t) = S(x-d_x, y-d_y, t-\Delta t)$

For each macroblock in the current frame (which is to be coded), a squared search window in the reference frame (which is an already encoded and decoded frame to avoid drift; note that the decoder never has access to an original frame, which is why the scheme is based on a reconstruction) is centered at the same position. Then, the block in the search window most similar to the block in the current frame is search by comparing the block to all blocks in the search window (*full search*). The displacement of the found block to the center of the search window (which is the position of the block in the frame to be coded) is represented by the motion vector. The criterion to find the most similar block requires some error criterion which is usually the sum of absolute pixel differences (SAD) due to the required high speed of the process. Note the similarity to the encoding algorithms in VQ and fractal compression.



After the most similar blocks in the reference frame have been identified for all macroblocks in the current frame, the *predicted frame* is constructed by pasting the identified blocks from the reference frame to the position as indicated by the motion vectors – this is the motion compensated prediction, a prediction to the current frame based on pixel data of the reference frame taking “motion” into account. Subsequently, the difference between the current frame and the predicted frame is computed and compressed together with the motion vector field (where the latter is encoded losslessly with DPCM). The compression of the prediction error frame is identical to the compression of the I-frames except for the quantisation matrix which has identical entries for all frequencies (recall the similarity to hierarchical progressive JPEG !).

It has to be noted that it is not exactly motion that is estimated in this process – actually we are just looking for a well suited block for prediction, there are techniques for really computing motion between frames like optical flow which are too costly in this context. Hovever, in many cases motion can be captured well which is beneficial for coding the motion vector field: if the motion vectors are very similar (they are in case of camera motion or zoom), DPCM is most effective. This effect can also be observed for "fractal motion estimation" where luminance and contrast adaptation are employed during the block matching process: on the one hand, the error frames contain much less energy and can be compressed better, on the other hand the motion vector field gets less smooth (no motion is captured any more but block similarity is the only criterion) and can therefore be compressed less effective.

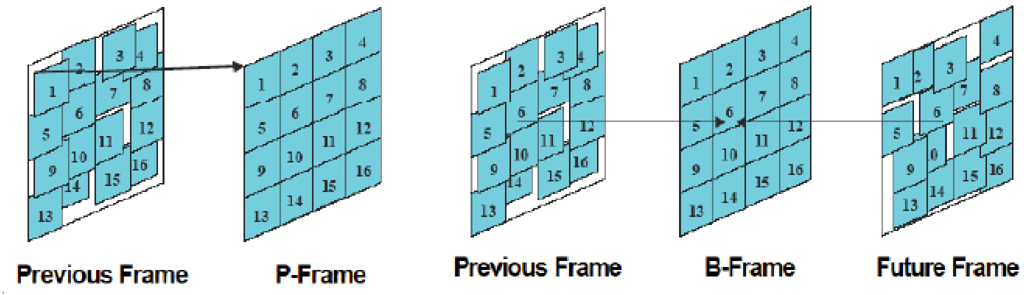


P frames are now computed in a way that for each macroblock in the current frame to be coded, a block is identified in the decoded reference frame (an I frame or another P frame) to be the most similar block. The error block is computed as described before. Now the encoder takes a decision how to actually encode the macroblock based on the error block (e.g. the variance of the block can be compared to that of the original block in the current frame):

- In case the prediction error is too large, the macroblock is compressed in I frame mode using intraframe compression (e.g. this happens in case of a scene change).
- In case motion estimation / compensation was sucessful, the macroblock is compressed in P frame mode using motion compensated prediction and the motion vector field together with the error frame is coded.

Note that this encoding decision is encoded in the macroblock header.

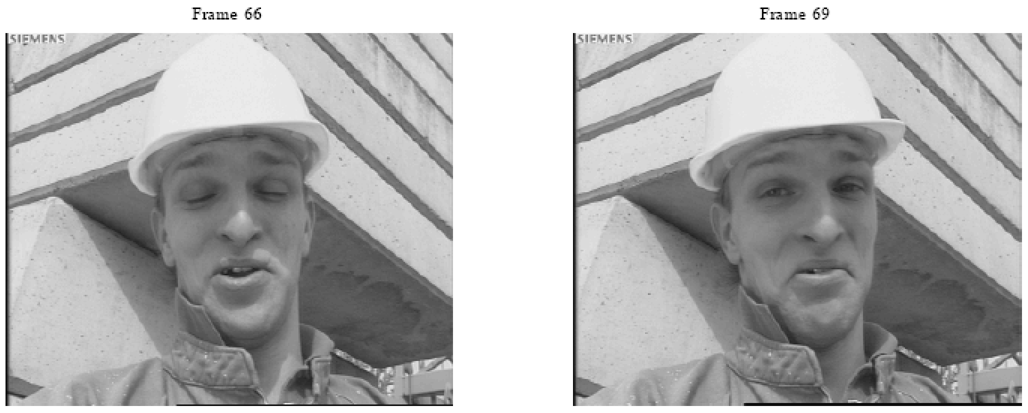
B frames are bi-directionally predicted. For each macroblock in the current frame, two motion vectors are computed, one to a reconstructed reference frame in the past and one to a reconstructed reference frame in the future, both either I or P frames but not B frames. The idea of applying the ME process in both



direction is inspired by the idea that objects can be occluded in the past, the might appear for the forst time in the current frame or, the most extreme case, a scene change happens to be co-located with the B frame. For B frames, there are 4 options for each macroblock how it is coded, the decision is again taken according to the best error block obtained:

- Intra coding in case the ME / MC process was not successful
- Backward predictive coding (P frame mode)
- Forward predictive coding
- Bidrectional predictive coding (here the predicted frame is averaged from the backward and forward prediction and both motion vectors are encoded

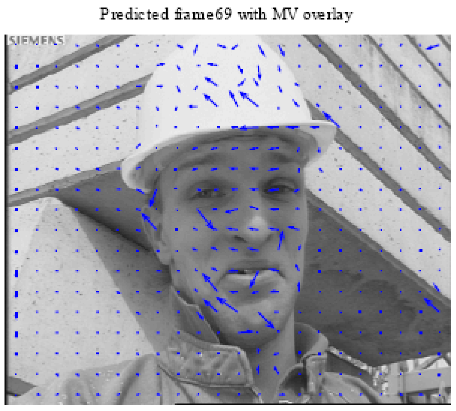
Again, the coding decision is encoded in the header of each macroblock.

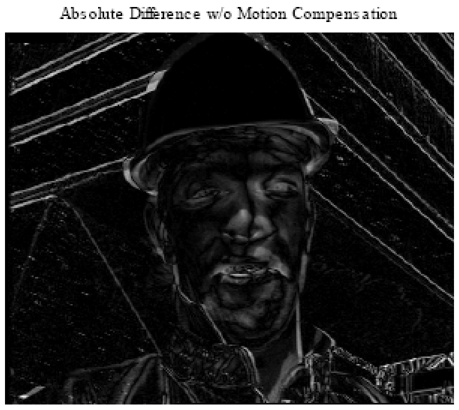


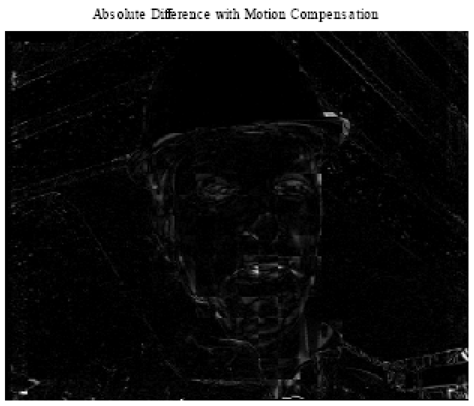
In the figures, examples are given for two original frames from a high motion video test sequence ("foreman") which is frequently employed in test sets for rating video coding algorithms.

It is crearly visible that motion estimation / compensation leads to much better results as compared to simple difference coding since the luminance information of the reference frame is shifted to the appropriate position before computing the difference.

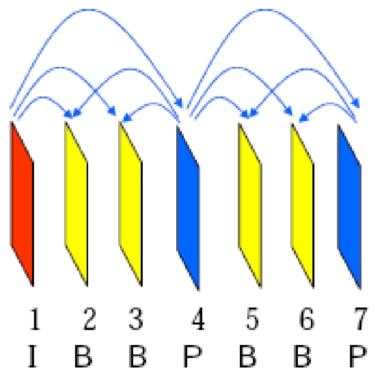
The coding scheme desribed so far has an impact on the ordering of the frames in the bitstream and on the order the frames are decoded. Since for decoding B frames, macroblocks from reference frames before





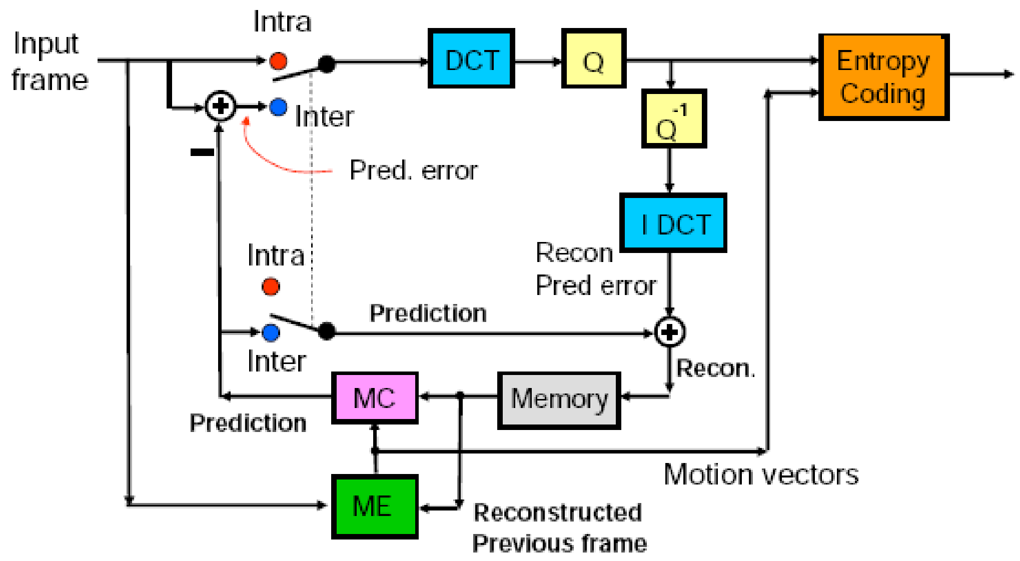


and after a B frame are required for the decoding of the B frame data, these reference data are stored first into the bitstram and also decoded in this order. For correct display, the frames have to be re-ordered again. Note that as a consequence, the ordering of the data in the original video does not correspond to the ordering of the data in the bitstream.



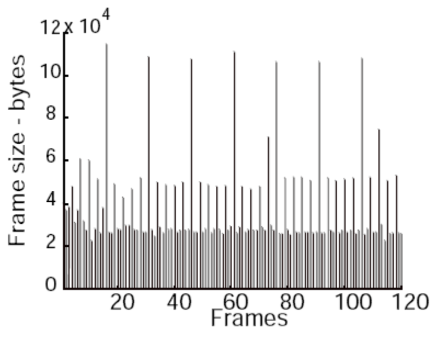
- Encoding order: 1 4 2 3 7 5 6
- Decoding order: 1 4 2 3 7 5 6
- Display order: 1 2 3 4 5 6 7
- Need more buffers
- Need buffer manipulations to display the correct order.

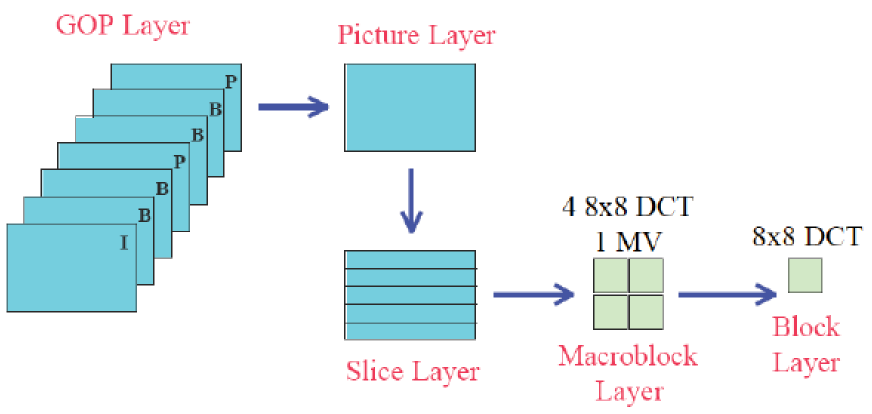
The figure shows a block diagram of a typical hybrid motion-compensated predictive video coding scheme. It has to be noted that due to the necessity to base predictions on reconstructed frames, each encoder needs to include a decoder as well.



Use reconstructed frame in the loop to prevent drifting.
Original input is not available to the decoder.

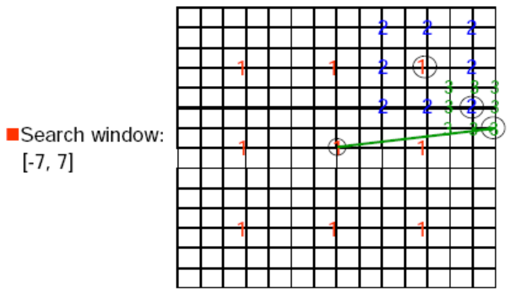
For video coding, we again note the typical tradeoff between achieved compression ratio and invested computational performance: B frames require most computational effort due to the two ME processes conducted on past and future frames. As can be seen in the figure, the size of the corresponding data is also the smallest. P frames are in-between and I frames require by far the highest bitrate of the three frame types. Therefore, for optimising compression ratio, the share of B frames in the GOP structure should be as high as possible. On the other hand, this maximises computational effort and the necessity to buffer frame data which raises coding delay.



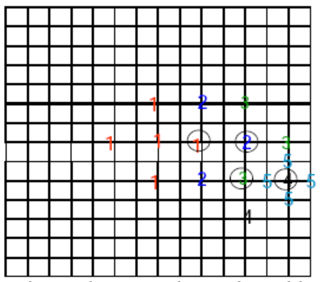


The MPEG-1 bitstream is organised in distinct bitstream layers, each of which is preceeded by a corresponding header information. The lowest layer is the VLC quantised coefficient data of 8×8 pixel blocks. These are collected in the macroblock layer, adjacent macroblocks are organized into slices which are then stored together in the frame layer. The next layer is the GOP layer, on top of the scheme is the sequence layer which has usually one instance and is preceeded by the main header.

As mentioned before, ME is not standardised and can be conducted in various manners. Full search as described before is very costly in terms of computational effort and especially, in software often different techniques are applied, basically reducing the number of blocks in the reference frame search range which are compared to the block in the current frame. This is usually not done by random subsampling but by first using a sparse grid across the entire search window, subsequently refining the grid in the are of the local minimum found on the sparse grid. This strategy dramatically reduces search time, on the other hand the global optimum is not often found but in most cases, matches have sufficient quality. The two examples show two different search patterns (three step search TSS and cross search), a large variety of other patterns has been investigared in literature.



- Initial step: 4. Search 9 positions, find the best MV1: (4, -4)
- Reduce step to 2. Search 8 pos around MV1, find MV2: (6, -2)
- Reduce step to 1. Search 8 pos around MV2, find MV3: (7, -1)
- Can be generalized to n-step search for larger window and sub-pixel.



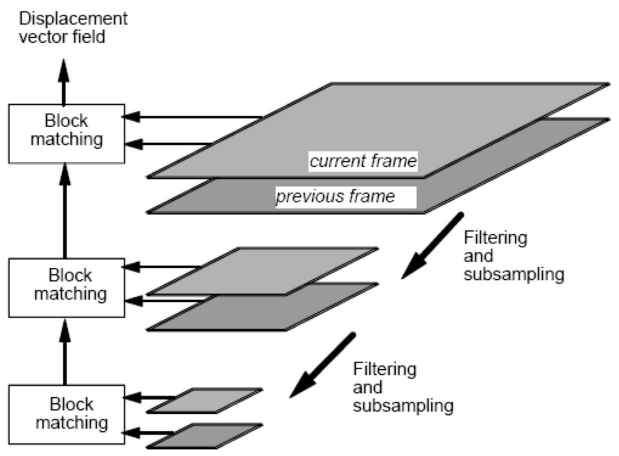
- Search 4 + -shaped or x – shaped positions in each step
- Initial step size: R / 2
- Step size is halved if the best match is at the center of the cross, or near the image boundary.
- Can also halve the step size after each loop.

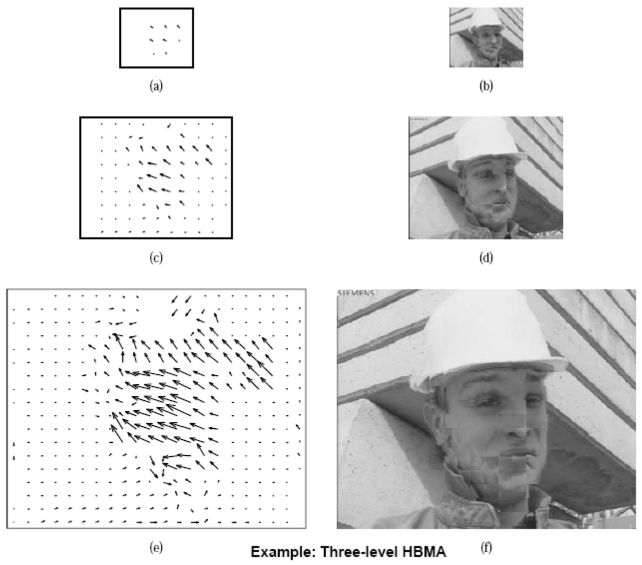
A different approach to improve motion estimation is to apply a hierarchical model. The advantage is that in a lower resolution, the process is less affected by noise and chances are better that actual motion is identified. Additionally, computational complexity is reduced in case that the search result in the lower resolution is taken as local minimum, which is only refinded locally in the higher resolutions.

A final issue not discussed so far but a very important one is that ME and the storage of motion vectors can be done with half pixel accuracy in MPEG-1. This sounds strange at first sight since there seems to be simply no data between pixels. The idea is that the data in the reference frame can be interpolated between integer pixel positions. This turn out to be highly beneficial since

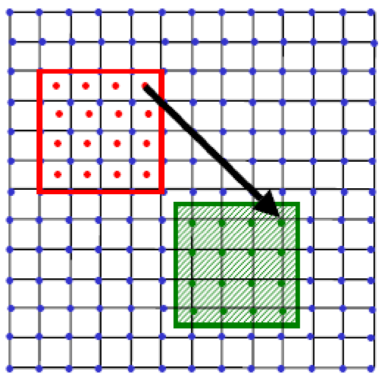
- simply more data is available to find block matches of high quality (compare domain overlap in fractal compression) and
- in case of high motion, visual data may simply not be present due to a sampling rate which is too low for the amount of motion. Here, the missing data is interpolated by fractional pixel accuracy ME / MC.

Again we face a tradeoff between compression ratio and computaitonal effort since the interpolation of data (usually bilinear interpolation schemes as the one shown in the figure) represents a significant computational burden. However, it has turned out that dispite of the higher bitrate required for the motion vectors, fractional pixel ME / MC is highly beneficial and is applied in all future standards not only as an option but as a must.


Displacement vector field
Block matching
current frame
previous frame
Filtering and subsampling
Block matching
Filtering and subsampling
Block matching

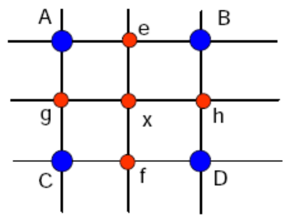


Example: Three-level HBMA



$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} 4.5 \\ 4.5 \end{pmatrix}$$

- Bilinear interpolation:



- Original samples:
  - A, B, C, D
- Half-pixel locations:
  - e, f, g, h, x

$$e = \left\lfloor \frac{A+B}{2} + 0.5 \right\rfloor$$

$$f = \left\lfloor \frac{C+D}{2} + 0.5 \right\rfloor$$

$$g = \left\lfloor \frac{A+C}{2} + 0.5 \right\rfloor$$

$$h = \left\lfloor \frac{B+D}{2} + 0.5 \right\rfloor$$

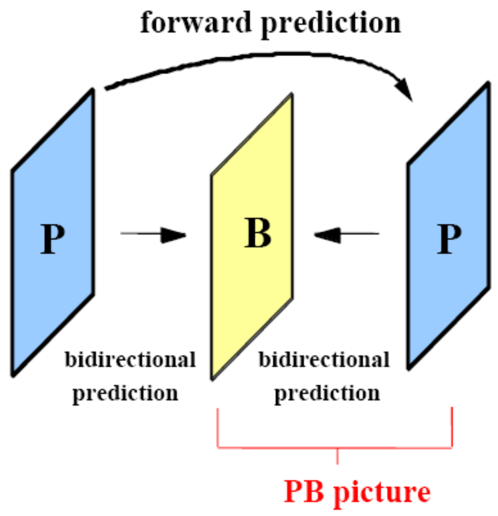
$$x = \left\lfloor \frac{A+B+C+D}{4} + 0.5 \right\rfloor$$

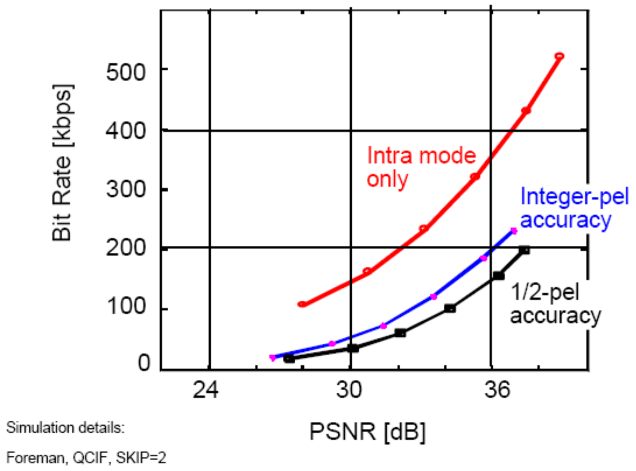
Bitrate control is similar to JPEG (based on scaling quantiser matrices) but there are more options in MPEG-1: spatially adaptive quantisation within a frame and frame dropping (reducing the frame rate) can be applied to B frames since the corresponding data does not serve as reference data for any other frames.

4.3 H.261 and H.263

H.261 is primarily intended for videoconferencing over ISDN channels at rates $p \times 64$ kbits/sec video and audio combined. This standard is very similar to MPEG-1 but does not define B frames. The reason is that symmetry of encoding and decoding and avoiding excessive frame buffering and corresponding coding delay is important for the interactive target applications. A H.261 encoder may skip frames and/or reduce spatial resolution to control rate besides adapting quantisation. Chroma sampling is 4:2:0, only QCIF (176×144 pixels luminance resolution) and CIF (common image interchange format, 352×288 pixels luminance resolution) are supported.

H.263 improves compression efficiency to support videoconferencing with 28.8 Kbits/sec over PSTNs (pulic switched telephone networks). Additional image formats are sub-QCIF (128×96 pixels), 4CIF, and 16 CIF (the latter two doubling the images' side length from CIF in each step). Contrasting to H.261, ME can be done with half pixel accuracy and MV predictive coding is improved from simple difference coding to a predictor based on a median of MV from three neighboring marcoblocks. Arithmetic coding and overlapping motion compensation (reducing blocking artifacts) can be used optionally. Also, an advanced prediction mode can be used optionally based on prediction of 8×8 pixel blocks: the prediction is a weighted average of the MV of the block and the MVs of two neigboring blocks (right and bottom block). Additionally, a resticted form of bidirection prediction can be used (PB frames, only one between two P frames, the unit of P frame – in the future of the B frame – and the B frame is called PB frame).





H.261 belongs to the H.320 familiy of standards where also audio compression, multiplexing, and communication protocols are standardised. The same is true for H.263, which belongs to the H.324 family.

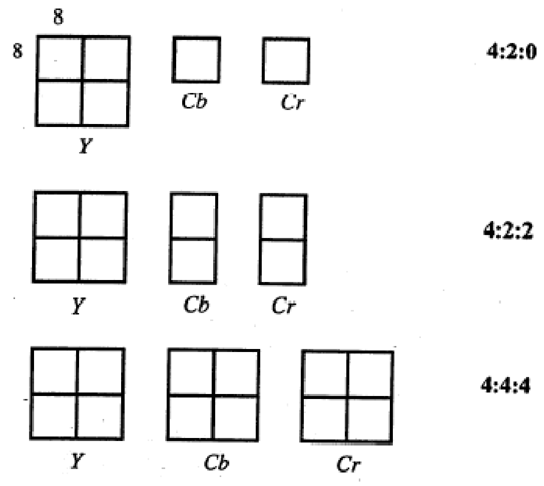
H.263+ and H.263++ exhibit many features as finally standardised in H.264 and never have been accepted widely.

4.4 MPEG-2

MPEG-2 is intended for higher bitrates and higher quality than MPEG-1 (for DVD, BluRay, DVB, HDTV); this is achieved partially by allowing other types of color subsampling: 4:2:2 where only one dimension is subsampled and 4:4:4 without subsampling).

MPEG-3 was originally intended for HDTV but has been incorporated into MPEG-2 as a "profile". Several profiles and levels have been defined supporting different functionalities (profiles) and parameter ranges (level). As a consequence, there is not a single MPEG-2 "algorithm" but many different algorithms for the different profiles. This makes the situation more difficult since a decoder needs to support the profile a bitstream has been generated with at the encoder.

MPEG-2 (ISO 13818) is published in 7 parts: multiplexing, video, audio, conformance testing, technical



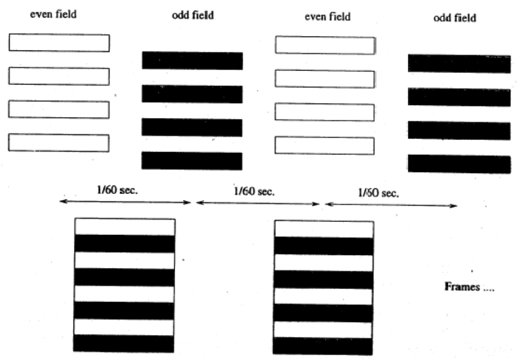
report with software simulation, digital storage media command control (DSM-CC, for VoD applications defining protocols for manipualtion / playback of MPEG bitstreams, and a non-backward compatible audio compression scheme (AAC as known from i-Tunes).

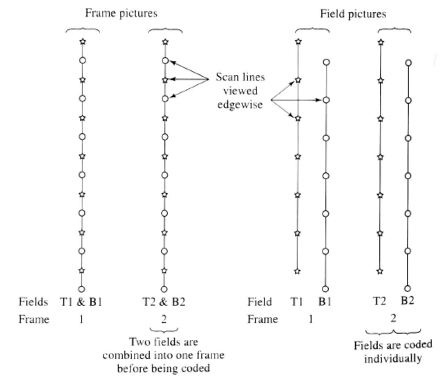
The basic structure and core coding system of MPEG-2 is similar to MPEG-1, and a MPEG-1 video with so-called constrained parameters can be decoded by any MPEG-2 decoder (a flag in the MPEG-1 bitstream indicates this property: $\leq 768 \times 576$ pixels, frame rate maximal 30 fps, bitrate at most 1.856 Mbits / sec (constant bitrate)). However, there are also important differences which will be discussed in detail in the following:

- MPEG-2 is intended for higher quality and higher bitrate (half pixel accuracy for MV is a must, finer coefficient quantisation, separate quantisation matrices for luma and chroma for 4:4:4 and 4:2:2 data),
- introduction of interlaced video (fields instead or frames in progressive video),
- corresponding modes in ME / MC,
- scalable profiles, and
- error resilience and error concealment

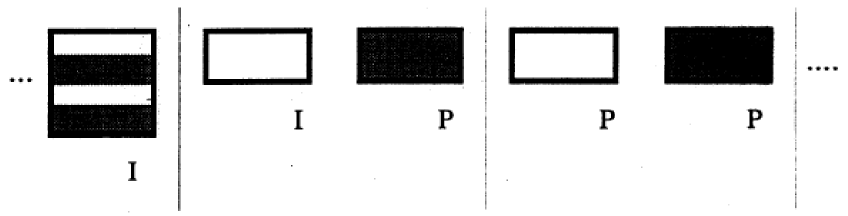
Interlaced video (as opposed to “progressive” video) is composed of fields. Each field consists only of even or odd numbered lines (even and odd filed) and in each field all pixels values correspond to the same point in time, however, two fields are displaced by a certain amount of time. This scheme originates from early TV, where upon arrival of new data (a field), only half of the lines in the image needed to be refreshed. This leads to new data entities with respect to compression as compared to progressive video: interlaced data can be either compressed as “field pictures” (where each field is compressed as an idenpendent unit) or as “frame pictures” (where the corresponding lines of two fields are interleaved and compressed together.

The decision about how to encode pictorial data can be made adaptive to image content and rate-distortion considerations and can be mixed within an image sequence as shown in the figure. This mixture

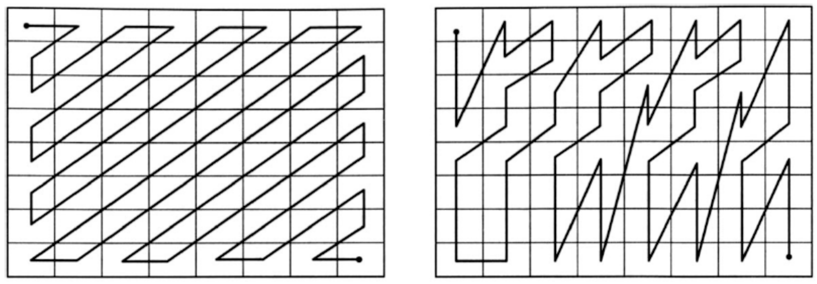



Frame pictures
Field pictures
Scan lines viewed edgewise
Fields T1 & B1
T2 & B2
Frame 1
2
Two fields are combined into one frame before being coded
Field T1 B1
T2 B2
Frame 1
2
Fields are coded individually

is not arbitrary, i.e. if the first field of a frame is a P (B) picture, then also the second field should be P (B). If the first field is I, the second should be I or P.

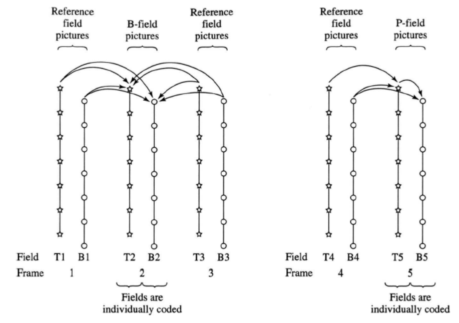


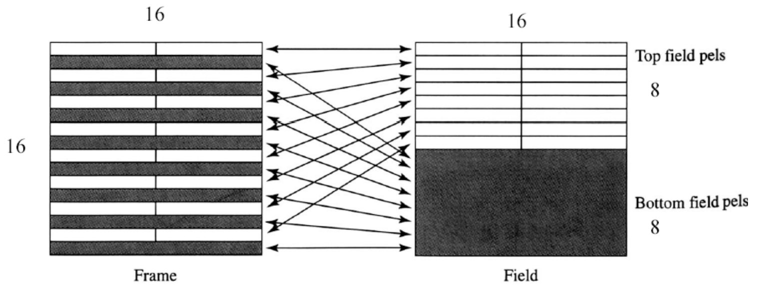
Due to the reduced horizontal correlation present in field pictures (which is due to the reduced resolution in this direction, basically a downsampling by 2 operation is done) an alternate scan pattern is used after transform and quantisation as shown in the figure.



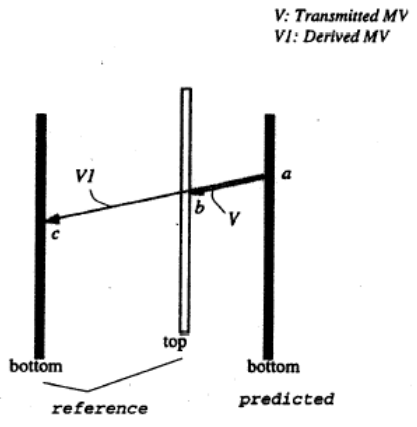
Due to the introduction of field and frame pictures, the motion estimation and compensation process can offer more flexibility and more modes to exploit temporal correlations. For field pictures, field-based prediction can be done by computing a single motion vector per macroblock in each direction or by computing two motion vectors in each direction (two MV for each field using two fields as reference). Dual prime prediction (see below) is different.

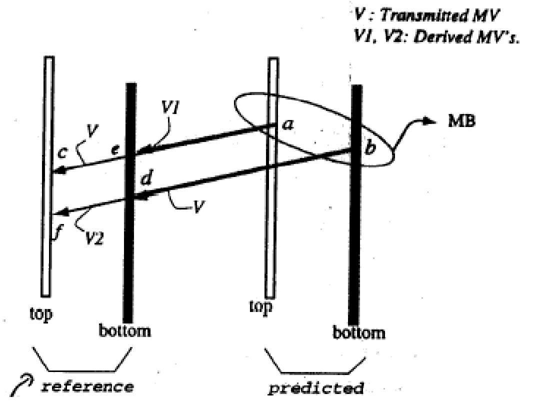
For frame pictures, a frame-based prediction can be done, where one MV is computed per macroblock in each direction (the fields of the reference frame may have been compressed individually as two field pictures or together as a frame picture). Field-based prediction in frame pictures (as shown in the figure) computes two motion vectors per marcoblock for each prediction direction, one MV for each of the fields. Again, the fields of the reference frames can be compressed individually or together.





A specific case of prediction is “dual-prime” prediction. For field pictures, data from two reference fields of opposite parity is averaged to predict a field picture. A single MV is computed for each macroblock of the field. This motion vector V points to the reference field. The second MV, V1, pointing to the field of other parity, is derived from V on the assumption of a linear motion trajectory (i.e. no acceleration). The prediction a is subsequently averaged between b and c. This mode is restricted to P pictures in case there are no B pictures between predicted and refereced fields.

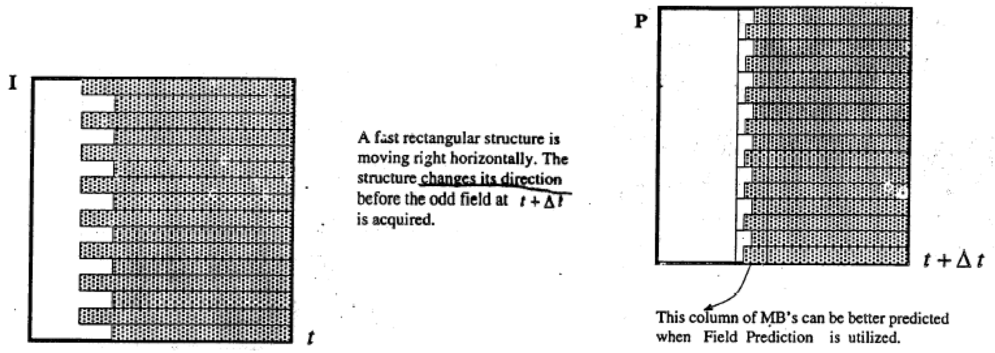




In frame pictures, dual prime prediction is a special case of field-based prediction. A single MV V is computed for each macroblock of the predicted frame picture. Using this vector, each field in the macroblock is associated with a field of the same parity in the reference frame. Motion vectors V1 and V2 are derived from V based on linear motion trajectory assumption again. Predictions for a and b are based on averaging c, e and d, f, respectively.

In frame pictures, it is possible to switch prediction modes on a macroblock basis. For example, the encoder may prefer field-based prediction mode over frame prediction in areas with sudden and irregular motion, since in this case the differences between even and odd field in predicted and reference frame do not match..

The concept of scalability refers to the ability of a decoder to decode only a part of a scalable bitstream

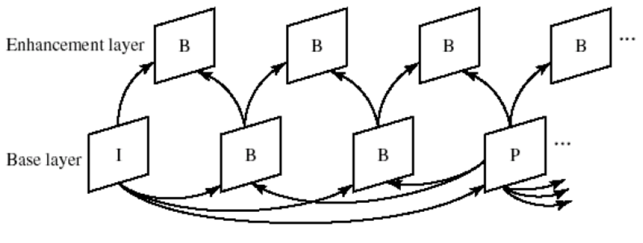

I
t
A fast rectangular structure is moving right horizontally. The structure changes its direction before the odd field at $t+\Delta t$ is acquired.
P
$t+\Delta t$
This column of MB's can be better predicted when Field Prediction is utilized.

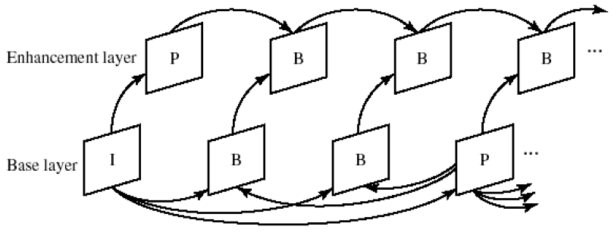
and generate a “useful” video. E.g., a lower resolution video can be obtained without decodeing the entire bitstream first and then lowering the resolution by postprocessing, or without simulcasting two distinct bitstreams corresponding to different resolutions. There are three types of scalability in video coding:

1. Temporal scalbility (different frame rates)
2. Spatial scalability or Resolution scalablity
3. Quality scalability (different levels of SNR / picture quality)

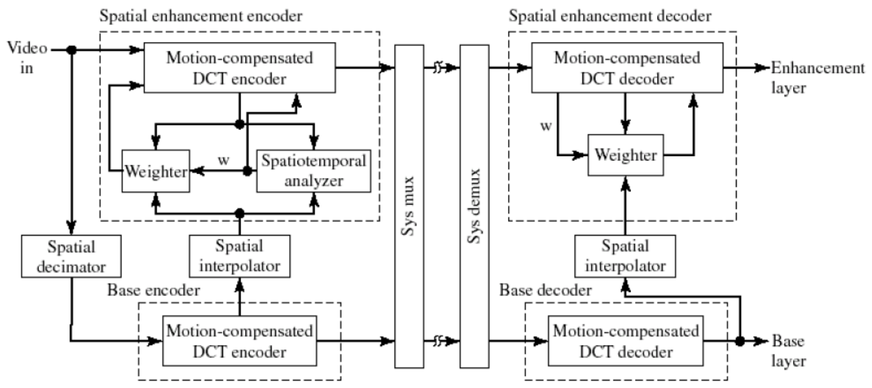
In MPEG-2, scalbility is achieved by the concept of layered compression, i.e. by generating a base layer and additional enhancement layers, which successivley improve the quality of the base layer if decoded together. In all schemes of this type, usually a higher computational demand and reduced compression efficiency is observed as the price of increased functionality.

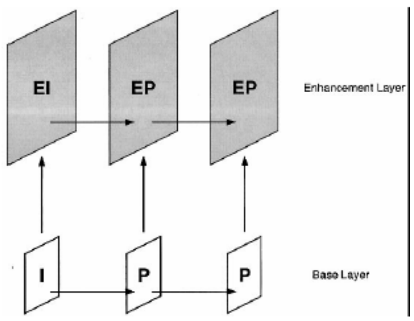
Temporal scalability is achieved very easily by assigning a set of suited B frames to the enhancement layer. In order to keep the compression degradation as small as possible, correlations among the frames in the base layer and those in the enhancement layer need to be exploited. While the first scheme depicted is similar to the classical prediction in a GOP (here prediction is only from base layer to enhancement layer), the second one applies a different prediciton structure (including intra-enhancement prediction).



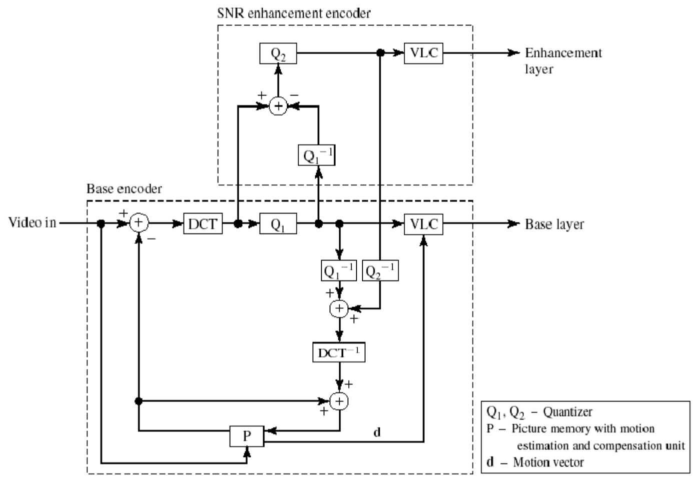


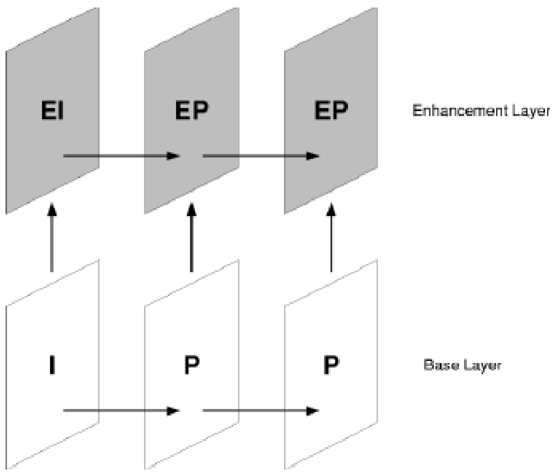
Spatial scalability is accomplished in a similar manner as in hierachical JPEG. Video frames are represented as spatial pyramids, prediction for the enhancement-layer frames is done based on base layer data (interpolated from the lower resolution) and based on temporal prediction in the enhancement layer data itself.


Spatial enhancement encoder
Video in
Motion-compensated DCT encoder
Weighter
w
Spatiotemporal analyzer
Spatial decimator
Spatial interpolator
Base encoder
Motion-compensated DCT encoder
Sys mux
Sys demux
Spatial enhancement decoder
Motion-compensated DCT decoder
Enhancement layer
w
Weighter
Spatial interpolator
Base decoder
Motion-compensated DCT decoder
Base layer



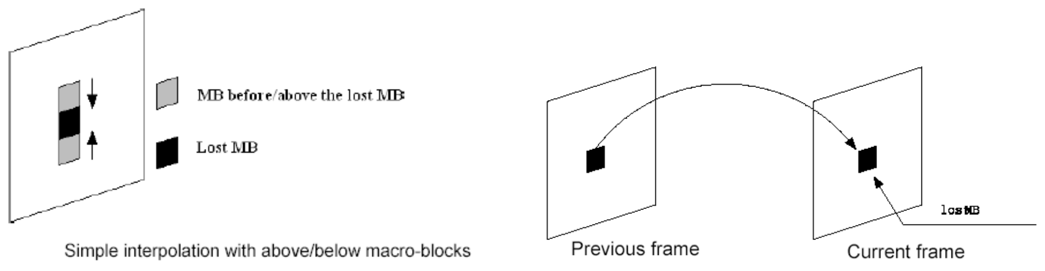
Finally, quality scalability is achieved based on differently quantised version of the video (as opposed to the lower resolution in resolution scalability). The reconstructed lower quality version is used as the prediction for the enhancement layer as well as motion compensated data from the enhancement layer itself is used. The scalability profiles in MPEG-2 (see below) were not very successful since when using a few layers, compression performance is drastically reduced at the price of significant complexity increase.



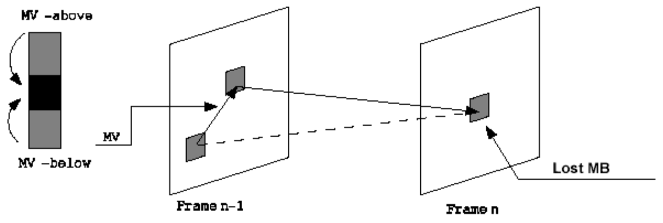


Since MPEG-2 is also intended for broadcast applications where transmission errors are to be expected (e.g. DVB-T, DVB-S, ...), several techniques how to cope with these errors have been incorporated into the standard. In particular, “error concealment” strategies have been suggested for the decoder. In P and B frames, a lost marcoblock is substituted with the motion compensated macroblock of the previously decoded frame. For motion vectors, the MV belonging to the block above the lost macroblock is used. If the top macroblock used in concealment is intra coded, there are two options:

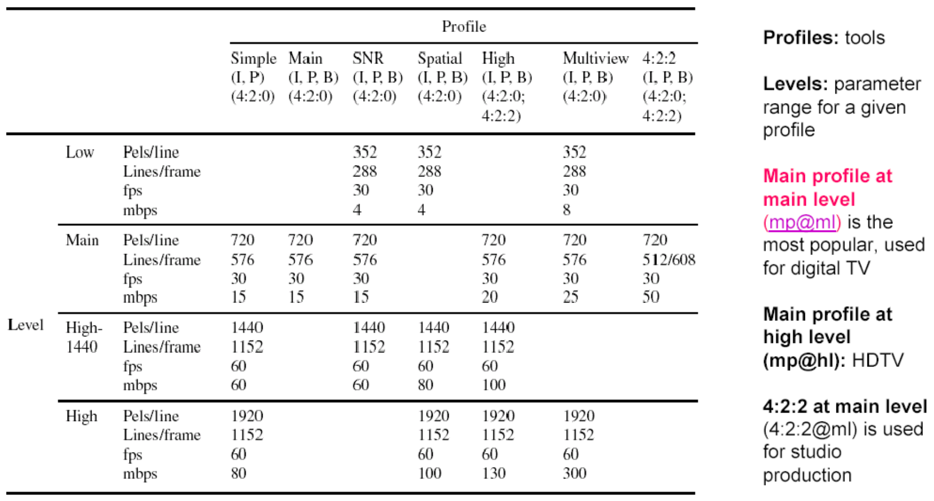
- If a MV for the intra block has been included (the standard allows to do this for concealment purposes), the concealment is done like described before.
- Otherwise, the concealment is done using the MV of the macroblock of the block in the previous frame (assuming zero motion). For the macroblocks in the first line in a slice, it is done in the same manner.


MB before/above the lost MB
Lost MB
Simple interpolation with above/below macro-blocks
Previous frame
Current frame
lost MB

In I-frames, error concealment is done in the same manner. The scheme can be improved by using two adjacent macroblocks, in case enough MV are retained, motion-compensated concealment can be done to approximate the lost macroblock.



Different functionalities are supported by different profiles of the standard, which are basically different algorithms. The different levels correspond to different parameter ranges which are supported (e.g. frame resolution, frame rate, etc.). A MPEG-2 decoder does not need to support all profiles and levels, in contrast, usually, a given decoder supports a specific profile and level combination (see the figure for examples).



			Profile						
			Simple (I, P) (4:2:0)	Main (I, P, B) (4:2:0)	SNR (I, P, B) (4:2:0)	Spatial (I, P, B) (4:2:0)	High (I, P, B) (4:2:0; 4:2:2)	Multiview (I, P, B) (4:2:0)	4:2:2 (I, P, B) (4:2:0; 4:2:2)
Level	Low	Pels/line			352	352		352	
		Lines/frame			288	288		288	
		fps			30	30		30	
		mbps			4	4		8	
	Main	Pels/line	720	720	720		720	720	720
		Lines/frame	576	576	576		576	576	512/608
		fps	30	30	30		30	30	30
		mbps	15	15	15		20	25	50
	High-1440	Pels/line	1440		1440	1440	1440		
		Lines/frame	1152		1152	1152	1152		
		fps	60		60	60	60		
		mbps	60		60	80	100		
	High	Pels/line	1920			1920	1920	1920	
		Lines/frame	1152			1152	1152	1152	
		fps	60			60	60	60	
		mbps	80			100	130	300	

Profiles: tools

Levels: parameter range for a given profile

Main profile at main level (mp@ml) is the most popular, used for digital TV

Main profile at high level (mp@hl): HDTV

4:2:2 at main level (4:2:2@ml) is used for studio production

4.5 MPEG-4

MPEG-4 (ISO/IEC 14496) currently consists of 27 (!) parts out of which we will describe Part 2 (Visual) and later also Part 3 (Audio). MPEG-4 Part 10 is commonly referred to as MPEG-4 AVC (advanced video coding) or H.264. MPEG-4 is targeted to applications in the multimedia area. Since interactivity is an important aspect of many multimedia applications, providing this functionality is one of the main aims of the standard. Due to the frame-oriented processing in former standards, interactivity was very limited. MPEG-4 uses the concepts of (arbitrary shape) video objects instead of frames to enable object-based interactivity.

MPEG-4 fuses ideas from the following (successful) fields:

- Digital TV
- Interactive graphics applications (e.g. computer games)
- Interactive multimedia applications (e.g. WWW)

The aim is to improve the applicability for users, service providers, and content creators:

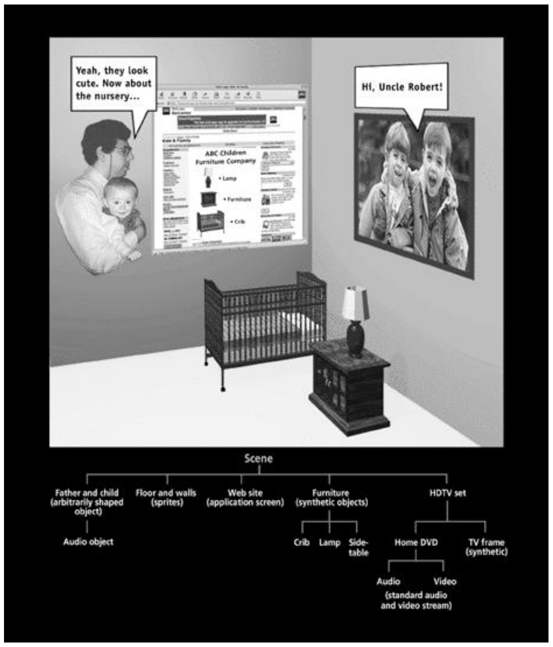
- Content creators: due to object-based representation, parts of the video data can be kept and re-used. Also copyright can be protected in a more focussed manner.
- Service providers: network load can be controlled better due to MPEG-4 QoS descriptors.
- Consumers: facilitating interactivity with multimedia data, over high-bandwidth networks or mobile networks.

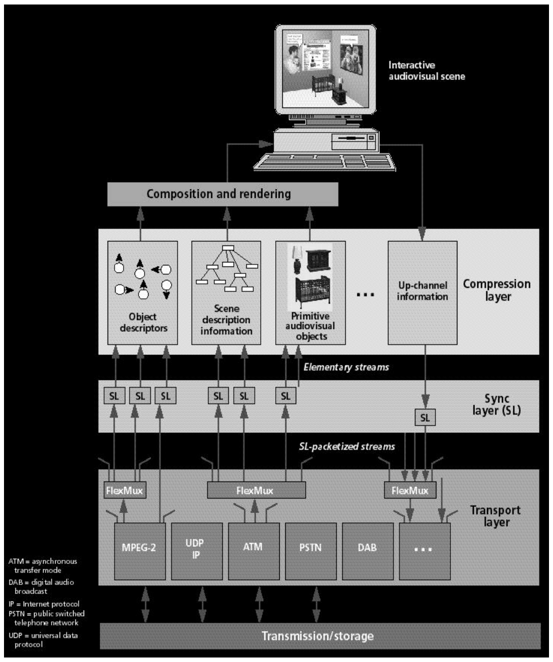
As a consequence, MPEG-4 does not only provide an improvement with respect to compression but mainly is focused on improved functionality. This is achieved by standardisation in the following areas:

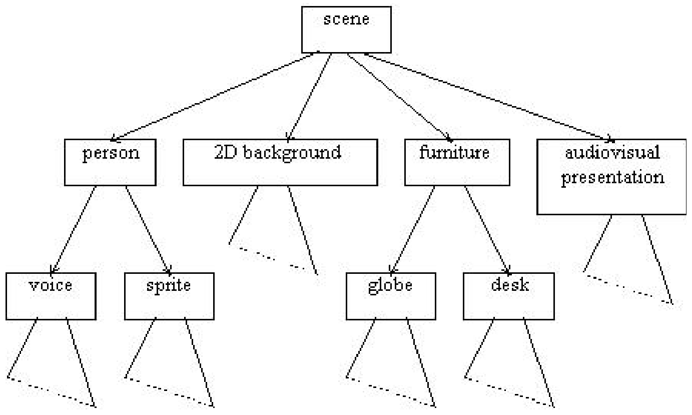
- Representation of "media objects" which are units of visual or audio (or both) data, which can be of natural or synthetic origin.
- Composition of media objects to an audiovisual scene.
- Multiplexing and synchronisation of data corresponding to media objects for network transfer according to respective QoS requirements.
- Interaction between user and audiovisual scene (note that this requires an uplink which does more as FFW or RWD).

MPEG-4 audio-visual scenes are composed of several media objects which are organised in an hierarchical fashion (tree). The leaves of the tree are "primitive media objects" like still images, video data, audio objects, etc. Additionally, special objects are defined like text, graphics, synthetic heads and faces with associated text to synthetise speech in a synchronised manner to head animation (talking head), synthetic sound, etc.

Scene description is standardised (features similar to computer graphics and virtual reality – VRML) and supports the following functionalities:

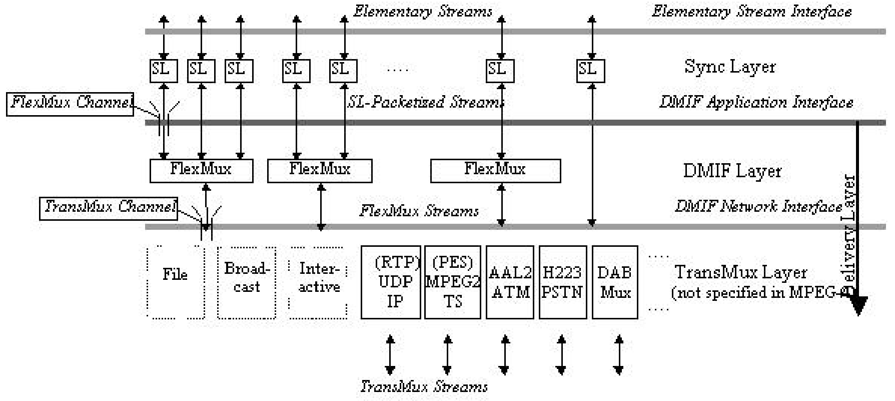

Yeah, they look cute. Now about the nursery...
Hi, Uncle Robert!
ABC Children Furniture Company
Scene
Father and child (arbitrarily shaped object)
Floor and walls (sprites)
Web site (application screen)
Furniture (synthetic objects)
HDTV set
Audio object
Crib
Lamp
Side-table
Home DVD
TV frame (synthetic)
Audio
Video
(standard audio and video stream)


Interactive audiovisual scene
Composition and rendering
Object descriptors
Scene description information
Primitive audiovisual objects
...
Up-channel information
Compression layer
Elementary streams
SL
Sync layer (SL)
SL-packetized streams
FlexMux
Transport layer
MPEG-2
UDP IP
ATM
PSTN
DAB
...
ATM = asynchronous transfer mode
DAB = digital audio broadcast
IP = Internet protocol
PSTN = public switched telephone network
UDP = universal data protocol
Transmission/storage



- media objects can be positioned at any given point in a coordinate system.
- media objects can be transformed thereby changing its geometrical, optical, and acustical appearance.
- primitive media objects can be fused into a larger single media object.
- streaming data can be "applied" to media objects to change their characteristics (e.g. a sound, a moving texture, animation parameter for a face, ...).
- the viewing and listening position can be changed.

Synchronised transmission of streaming data with different QoS over a network is provided employing a synchronisation layer using a two-layer multiplexer.



The first layer of the multiplexer is implemented corresponding to the DMIF (delivery multimedia integration framework) specification. DMIF is a session protocol for multimedia data similar to FTP. Using the MPEG FlexMux tool, elementary streams with identical or similar QoS requirements are treated together. TransMux is used to implement the actual QoS requirements for the fused streams. The standard only covers the interface to TransMux, the actual deployment has to be carried out using the network protocol.

Interactivity (uplink !) is facilitated by e.g. changing viewing and/or listening angel, remove or translate objects from / in a scene, trigger a cascade of events by a mouseclick (e.g. playing a video, an animated graphics), selection of suited language or selection of the desired view from a multiview video.

4.5.1 Compression of synthetic video objects

To give two examples, we describe face animation and general purpose mesh-coding with texture mapping.

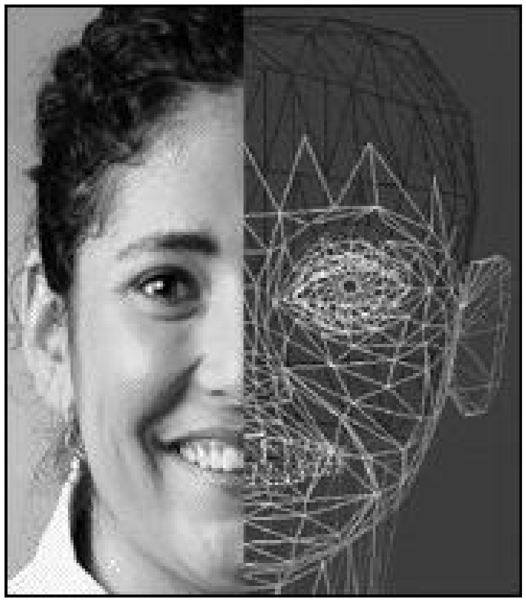
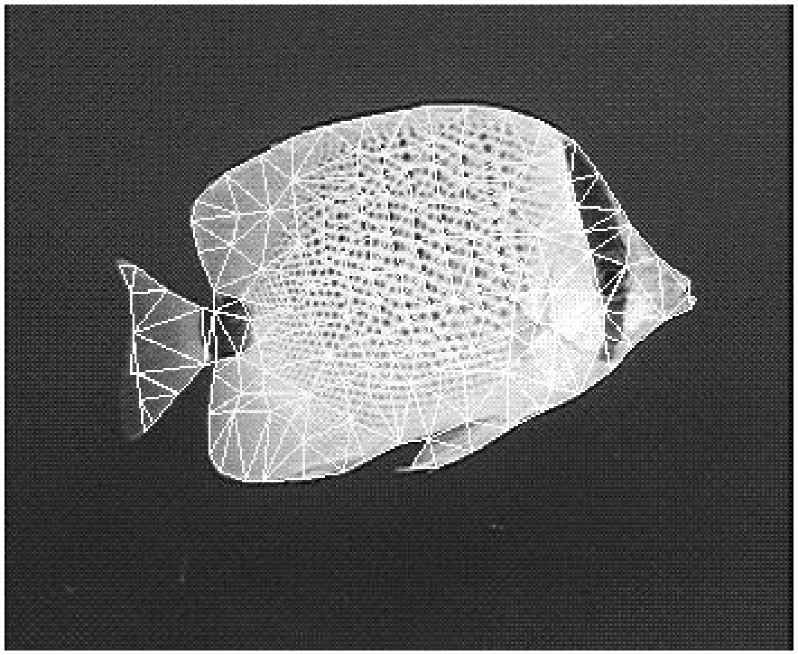
A two-dimensional mesh is a partitioning of a 2-D planar region into polygonal parts. The corner points are denoted as "node points" or "vertices", MPEG-4 only supports a triangulation. A 2-D dynamic mesh describes the movement of the vertices in a certain amount of time. Efficient coding (based on predictive coding of course) is defined for those data. In order to map texture onto a 2-D mesh, the triangles of the original mesh are deformed and the texture is adjusted to the resulting parts by a technique called "warping". In case of restricting the vertices movements to affine mappings, the movement can be stored very efficiently.

A "facial animation object" is a generic 2-D mesh of a face which can be used to generate an animated synthetic face. Shape and appearance can be described by "Facial Description Parameters" (FDP). "Facial Animation Parameters" (FAP) are used to generate facial expressions by manipulating specific mesh points.

MPEG-4 standardises techniques how to efficiently store FDP and FAP and also how to interpolate them in case of data loss. As in the case of generic 2-D meshes, facial texture (an image of a specific person) can be mapped onto the wireframe model. MPEG-4 defines JPEG2000 as the technique to compress the textures used for mapping in a scalable manner. The same ideas are used for a parameterised description and animation of the entire human body.

Mesh-representation enables interesting features:

- Augmented reality: mixture of computer-generated data and natural data.

- Replacement of naüral data by synthetised data.
- Improved compression @ low bitrates: the texture is compressed only once plus additional movement parameters.
- Improved representation of object boundary @ lower quality: polygons instead of bitmaps

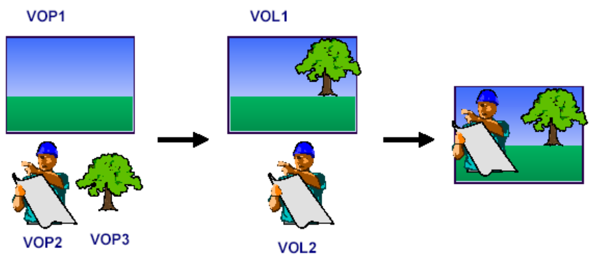
4.5.2 Compression of natural video objects

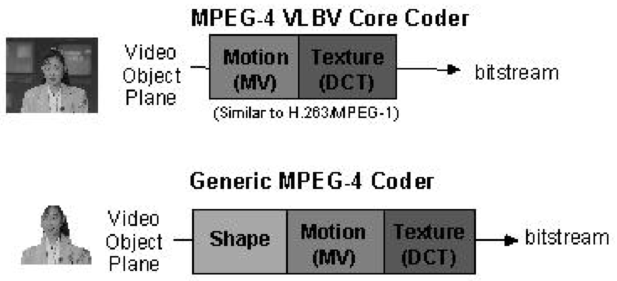
The arbitrary shape of the video objects leads to the question how video data can be generated, which offers object-based segmentation. MPEG-4 standardisation does not cover this issue (which is a very critical one !). The following options could be considered:

- Fully automatic segmentation of existing video material: is only possible in case additional knowledge is available or in case of simple sequences (e.g. head-and-shoulder)
- Semi-automatic segmenation of existing video material: keyframes are segmented manually, object borders are temporally tracked.
- Geberation of new video material with techniques like blue-box (we know it from the weather forecast).

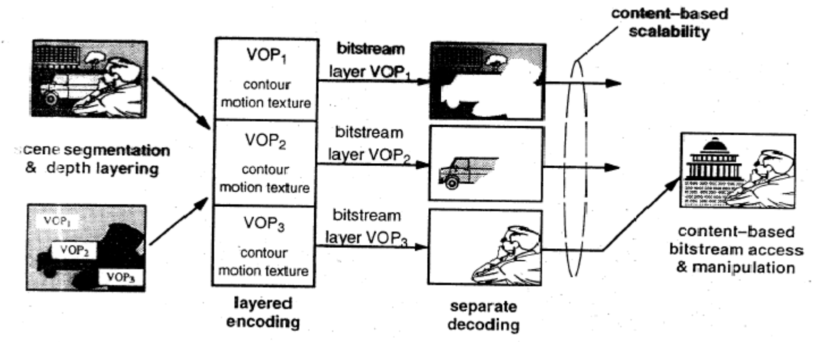
The bitstream hierarchy is as follows: a comple (hierarchical) description of a scene is called “Visual Object Scene (VS)” which contains 2-D or 3-D objects in base- and enhancement layers. A “Video Object (VO)” corresponds to an actual object in the scene, in the simplest case a rectangular frame are an arbitrarily shaped object, which can be foreground or background. The next level is the “Video Object Layer”, the content of which depends if the object is represented in scalable or non-scalable form. A “Video Object

Plane (VOP)" is a temporal sample of a VO, several VOPs can be grouped to a "Group of Video Object Planes (GOV)".



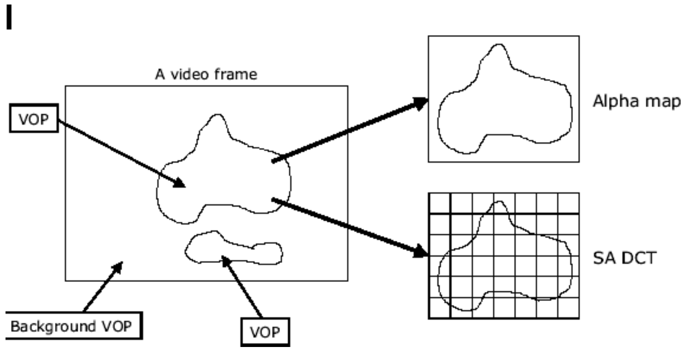


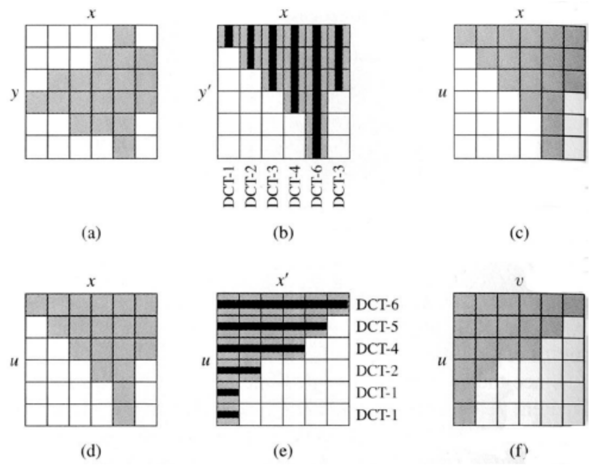
VOPs can be compressed in different manners, the classical mode a VOP contains shape information, motion parameters, and texture information. As an alternative, a VOP can be represented as "sprite" (see below). Single VO are processed and compressed indenpendent of each other.


content-based scalability
VOP1 contour motion texture
VOP2 contour motion texture
VOP3 contour motion texture
bitstream layer VOP1
bitstream layer VOP2
bitstream layer VOP3
scene segmentation & depth layering
layered encoding
separate decoding
content-based bitstream access & manipulation

Coding of shape information: In case of binary shape information, shape is defined by a binary matrix which covers the objects' shape. The matrix is tiled into 16x16 blocks, the binary alpha blocks BAB, values outside of the object are set to 0, the other values are set to 1. In addition to BABs at the border of the object, two special types of blocks have all identical values 0 – transparent or 1 – opaque which are encoded by a single symbol. BABs are encoded by motion compensation, arithmentic encoding and DPCM of motion vectors. In case of storing transparency information, gray scales with 8bpp are used instead of binary values and DCT-based coding is done instead of AC.

Motion compensation: beside a technique for global motion compensation which computes a MV for each pixel based on warping a grid the classical ME / MC concept of MPEG is used, only adapted to the VOP concept. ME is done only for macroblocks, which are situated in a rectangle surrounding the shape of the object. In case of blocks situated entirely inside the object, there is small difference to MPEG-2 (overlapped ME / MC can be used and MV can be computed for each 8x8 block). For macroblocks being positioned only partially inside the VOP, "modified block (polygon) matching technique" is used, which means that only pixel values inside the VOP are used for evaluating the difference; if the reference block is outside the VOP, pixel values are filled according to a fixed scheme.



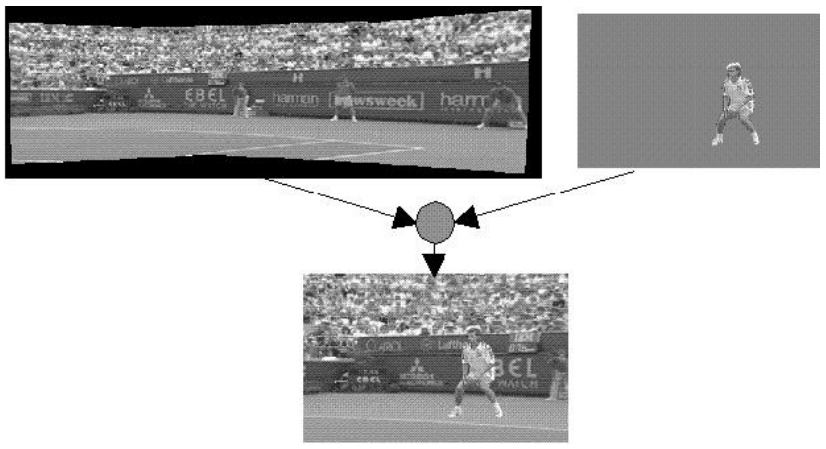


Texture coding: For texture blocks inside the VOP, compression is conceptually identical to MPEG-2. Blocks positioned at the border of the VOP are pasted with values: for B or P-VOPs, 0 is used for pasting, for I-VOPs the mean of the VOP (with some slight modifications) is used. Alternatively, a shape adaptive DCT can be used, providing better compression (no pasted values !) but causing more complex computations (hardware support is difficult to achieve). DC and large AC coefficients can be used to predict the coefficients of the block to be encoded. There are three scan variants (following the prediction of the coefficients): zig-zag, alternate-horizontal and alternate-vertical.

Sprite coding: a sprite is a large VO which is present during an entire scene and which is usually larger than the current field of view of a single frame. A classical example is a background sprite. A VO like this is transmitted / decoded at the start of a scene and modified according to the needs of the current frames. Coding a sprite is done as for I-VOPs.

In extending error concealment capabilities of MPEG-2, MPEG-4 is also aimed towards mobile environments and their even more problematic transmission errors:

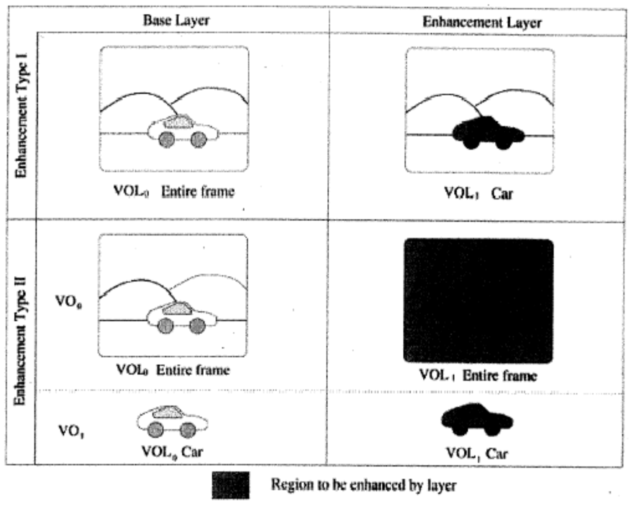
- Resynchronisation: unique markers are embedded into the bitstream indicating synchronisation points from which decoding can be restarted.
- Data partitioning: Motion vector data and texture data are separated in order to facilitate more efficient error concealment. In case one of the two data types is lost, the dislocated other type can be



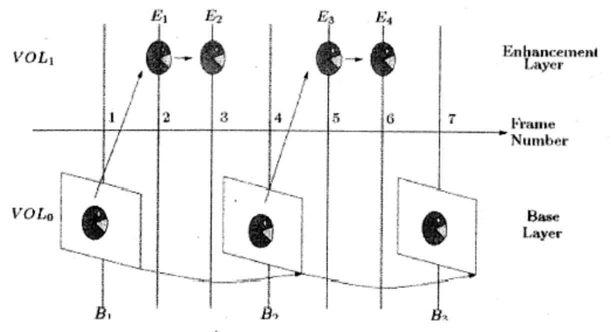
used for concealment.

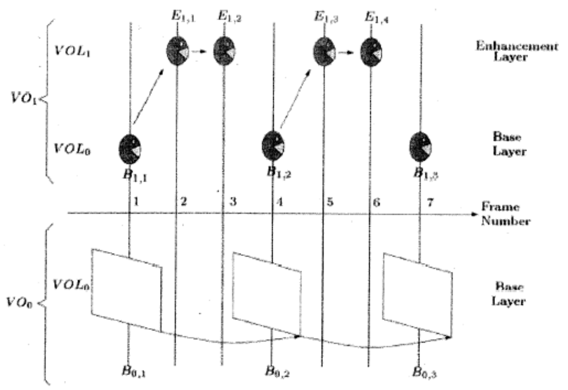
- Header extension Code: additional redundant header information is inserted into the bitstream.
- Reversible VLC: codewords have been developed, which can be decoded from both sides. in case the front part is lost, decoding can start from the end.

Scalability is adapted to the VOP concept. Scalability is implemented by multiple VOL. Scalability on an object basis is achieved by the VOP concept itself in a natural manner. For temporal scalability, two types are supported:



- Enhancement layer improves temporal resolution of a part of the base layer only (the important one)
- Enhancement layer improves temporal resolution of the entire base layer



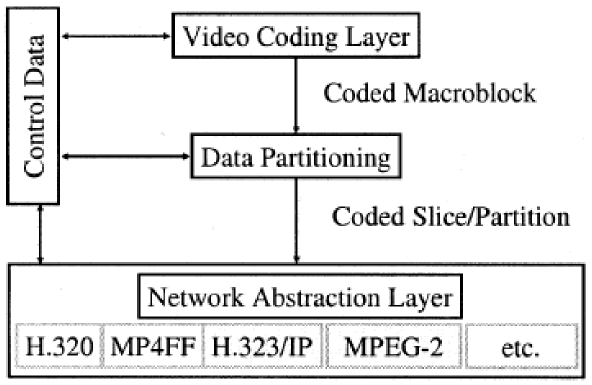


The most well known implementations of MPEG-4 Part 2 are DivX and Xvid, however, they only support rectangular VOP. There are no implementations and applications which support all functionalities. It seems that object-based interactivity is not really required for video consumption.

4.6 H.264 / MPEG-4 AVC

An inofficial experimental coding scheme was named "H.26L" which is still seen in some result charts and papers. H.264 follows a two layer design: a network abstraction layer (NAL) and a video coding layer (VCL). NAL is designed in order to provide "network friendliness" in the sense of enabling the use of the VCL for a broad variety of systems (e.g. RTP/IP, file formats (ISO MP4), H.32X, DVB-X). The video is organised in NAL units, each of which is a packet of data preceeded by a header byte indicating the type of data. Data is interleaved with emulation prevention bytes to prevent the "start code prefix" to be generated. This definition is used for data transport in both packet-oriented and bitstream oriented transport systems.

4.6.1 NAL



For byte-stream format (e.g. video conferencing H.320), the data is transported as an ordered stream of bytes within which the location of the NAL unit boundaries is identifiable from the start code prefix. In other systems (e.g. RTP) the coded data is carried in packets that are framed by the transport protocol, so identification of the boundaries of the NAL units can be established without the the start code prefix patterns. NAL units are classified into VCL units (containing actual video data) and non-VCL units (containing header information like parameter sets or timing information). A parameter set is supposed to contain information that is expected to change rarely and can be used to decode a significant number of VCL NAL units, sequence and picture parameter sets are distinguished. Each VCL NAL unit contains an identifier to the relevant sequence and picture parameter sets.

An access unit composed of several NAL units can be decoded to one entire frame – called a primary coded picture. These data consists of “slices” that represent samples of the video. Additional redundant representations of some areas of the same frame can be included for error concealment purposes – redundant coded pictures. Decoders are not required to decode these data. If the frame is the last of a coded video sequence (a group of picture that is indenpendently deocidable and refers to a single sequence parameter set) an “end of sequence” NAL unit may be present; if it is the end of the entire NAL unit stream, an “end of stream” NAL unit may be present.

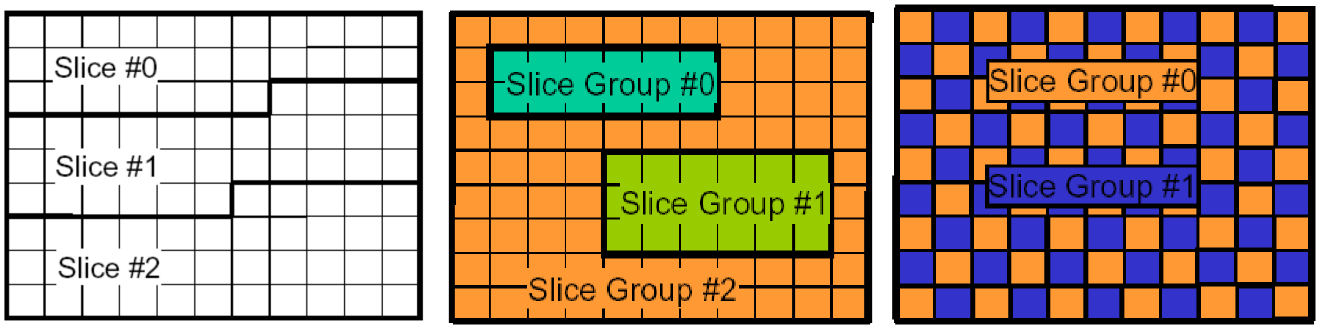
A coded video sequence consists of a series of access units, at the beginning of which an “instantaneous decoding refresh (IDR)” access unit is found. This IDR access unit contains an intra frame – a frame which can be decoded without reference to any other frame in the NAL unit stream. There are no references in the sequence to frames before the IDR access unit. A NAL unit stream may contain one or more coded video sequences.

4.6.2 VCL

The basic idea is identical to former standards since it follows the idea of block-based hybrid video coding, where the video is represented by squared units of data, the macroblocks, consisting of 16x16 samples luminance with the additional associated chroma samples. Of course there is support for progressive and interlaced video as it is for MPEG-2,4. It has to be noted that improved coding efficiency is not caused by just few single new techniques, it is the sum of many smaller enhancements that lead to the gain in R/D performance.

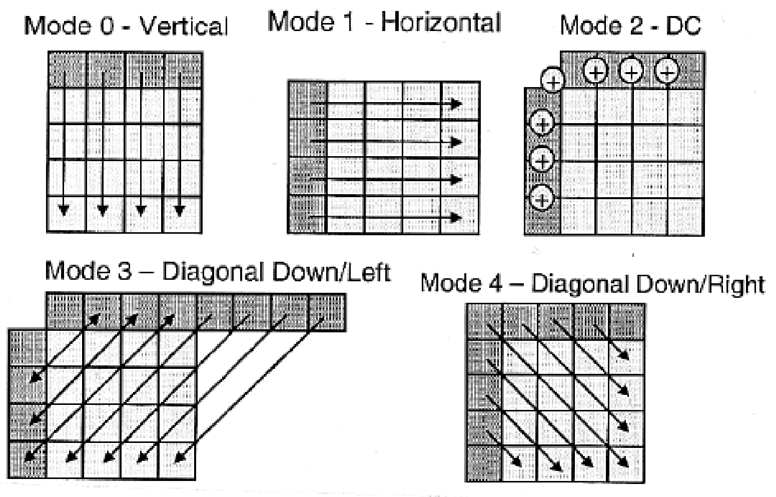
Slices are sequences of macroblocks which are processed in raster scan order (except if FMO is used, see below). A frame can be split into one or several slices as shown in the figure. Slices are self-contained in the sense that in case of available picture and sequence parameter sets their syntax elements can be parsed from the bitstream and the pixel data can be correctly decoded without the use of data from other slices (given that correct reference frames are available).

Flexible macroblock ordering (FMO) modifies the way frames are partitioned into slices and macroblocks by using the technique “slice group”. Each macroblock can be assigned to any slice independent of its scan-order position based on the “macroblock to slice group” map (which contains a slice group ID for each macroblock). The purpose of doing this can be a region of interest coding type or, like in the example of the checkerbord pattern, a tool for error concealment if these two slice groups are transmitted in separate packets. In addition to the classical I, P, and B slices, SP and SI slices are defined: an SP slice is a so-called switching P slice which enables efficient switching between pre-coded pictures of different bitrate. An SI slice is used to enhance an SP slice for a perfect macroblock match to enable random access.



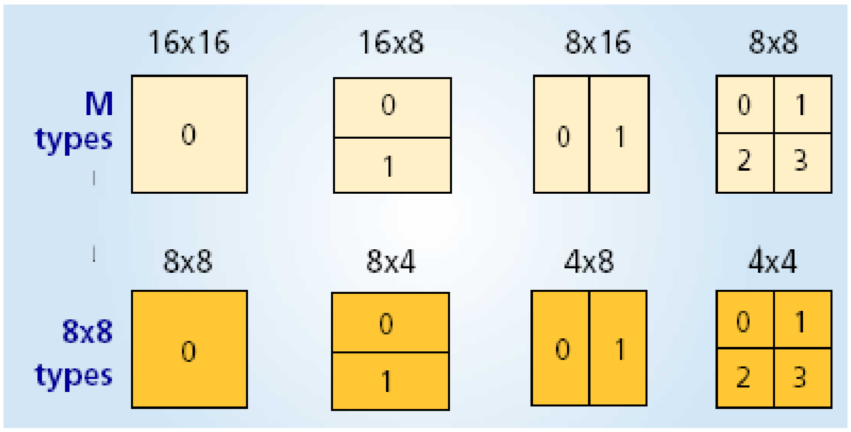
With respect to interlaced video, we again have frame pictures, field pictures, or an adaptive mixture, where for two vertically adjacent macroblocks the decision can be taken how to process them (interleaved or non-interleaved). The choice between the three options can be made adaptively for each frame in the sequence – the choice between the first two is called picture adaptive frame/field (PAFF) coding. The third option is called macroblock adaptive frame/field coding (MBAFF) and is applied in case a frame consists of areas with high motion and other areas with low motion content. The distinction between PAFF and MBAFF frames leads also to different scan oders, motion vector coding etc. for the two modes. While in general MBAFF is more effieicent in terms of compression due to its adaptivity, for some rare cases like rapid global motion, scene change etc. PAFF can be superior.

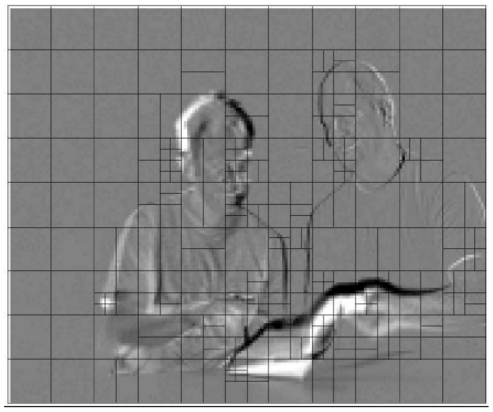
Intra-Frame Prediction: In all slice coding types, three different types of intra coding are supported. Intra_4x4 mode, Intra_16x16 mode, and I_PCM mode. I_PCM mode bypasses prediction and transform coding and directly encodes the values of the encoded samples (for lossles coding, representation of anomalous picture content without expansion). Intra-prediction is done in the spatial domain.



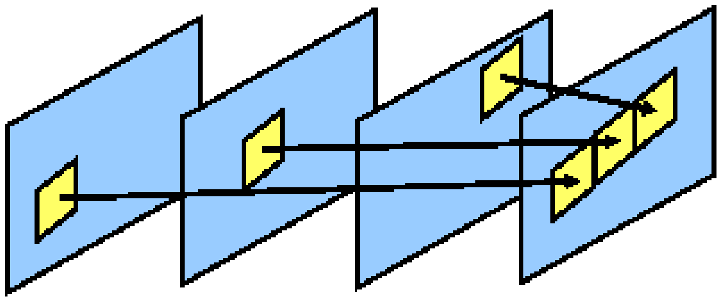
When using Intra_4x4 mode each 4x4 pixels block is predicted from spatially neighbouring samples. One out of nine prediction patterns is employed – the way how to choose the most appropriate one is not standardised and is subject to current research. The aim is to exploit the redundancy of edge-like structures. When utilising Intra_16x16 mode (full macroblock prediction), the whole luma component of a macroblock is predicted. Four prediction modes (vertical, horizontal, DC, planar) are defined similar to the Intra_4x4 mode but in this case 16 neighbours on each side of the block are used for prediction. The planar mode fits a parametric surface to the pixel content. Chroma samples are predicted similar to the Intra_16x16 mode since chroma data is usually more smooth. There is no intra prediction across slice boundaries.

Inter-Frame Prediction: Macroblocks can be partitioned into different tiles which are the basic elements for the ME / MC process. For the partitioning into 8x8 tiles in P slices, an additional syntax element is stored indicating a possible further decomposition. A maximum of 16 motion vectors can be used for a single macroblock, the approach facilitates an adaptive structuring of the ME / MC process depending on video content.





Accuracy of ME / MC is quater pixel, where half pixel accuracy is obtained by bilinear interpolation, quater-pixel accuracy averages two half-pixel samples. MVs can extend over frame boundaries, in this case, the boundary samples are extrapolated. No ME / MC computations cross slice borders.

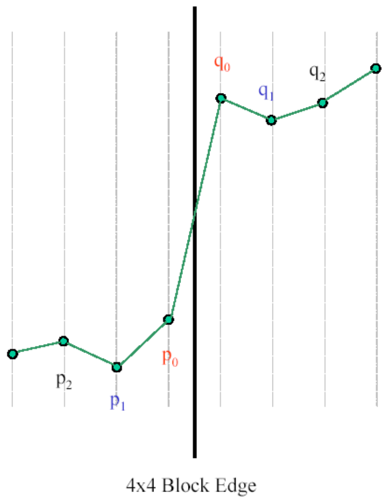


H.264 supports the use of multiple reference frames for a single slice. This means that different macroblocks can be predicted by macroblocks originating from different frames. In order to accomplish this, encoder and decoder store the reference frames for inter prediction in a multipicture buffer. The index of the reference frame used for a macroblock in the multipicture buffer needs to be stored for each macroblock. In an 8x8 region, all blocks need to reference the same index in the buffer. Additionally, a P_Skip prediction can be done where no residual error and no motion vector is encoded but the reconstruction is done using the frame at index 0 with neighbourhood averaged motion information. In this manner, large areas with no change or constant motion can be represented with few bits. For B slices, a generalisation is done since corresponding blocks may serve as prediction for other blocks (contrasting to earlier standards). B slices use two different multipicture buffers. In B slices, 8x8 blocks are not further subdivided and B_Skip is similar to the corresponding P slice mode.

Transform: H.264 employs transform coding of the prediction residual (note that also I-slice blocks are encoded in predictive manner !). The transform is applied to 4x4 blocks and an integer-operation based transform similar to the DCT is used instead of the DCT. This leads to faster execution and the avoidance of rounding errors in the inverse transform. For Intra_16x16 prediction, the DC coefficients of the 16 blocks are transformed a second time (recall the similarity to JPEG XR), also for the 2x2 chroma DC coefficients a dedicated smaller transform kernel is provided. The smaller blocksize of the transform can be justified by the lower amount of spatial correlation present in residual errors due to improved prediction. Additionally, blocking artifacts are less pronounced and the transform is simply faster.

Entropy coding can be done in two ways: Context-adaptive VLC (CAVLC, which chooses codeword tables according to the statistics of already encoded values in a process similar to JPEG XR coding) and Context-adaptive binary arithmetic coding (CABAC), where the latter is more expensive but improves compression performance significantly. Coding of other syntax elements in headers is done with fixed length coding or a simple universal VLC (UVLC) code table.

Typical artifacts in block-based video coding are found at the edges of the blocks. To compensate this, H.264 defines an adaptive in-loop deblocking filter, where the strength of the filtering is controlled by sample values. The principle is illustrated in the figure: whether p_0 and q_0 as well as p_1 and q_1 are filtered is determined using quantisation parameter (QP) dependent thresholds. The idea is that if a large difference between samples close to the edge is observed, it is likely that this results from blocking artifacts. However, if the difference is so large that it cannot be explained by the coarseness of QP, the edge is probably the result of actual image content and is not smoothed. So blockiness is reduced while maintaining sharpness of the image (contrasting to global smoothing at blockborders).



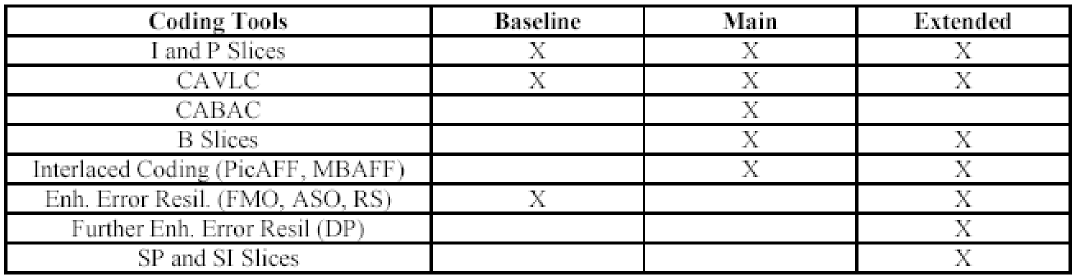
Similar to MPEG-2,4 profiles and levels have been defined for various application scenarios. The Baseline profile supports all features except the following two feature sets:

- Set 1: B slices, weighted prediction, CABAC, field coding, picture or macroblock adaptive switching between frame and field coding.

- Set 2: SP/SI slices and slice data partitioning.

Set 1 supported by the Main Profile, which on the other hand does not support FMO, ASO (arbitrary slice ordering), and redundant pictures which is supported by baseline. Thus, not all baseline video sequences can be decoded by a Main Profile decoder. The Extended Profile supports baseline features, Set 1 and Set 2 features, but not CABAC. All 15 levels can be used for each profile. Typical application areas are:

- Baseline Profile: low latency applications below 1 Mbps, like H.320 video conferencing, 3GPP conversational services, H.323 with best effort IP/RTP protocols.
- Main Profile: entertainment video applications with moderate latency in the are of 1 to 8 Mbps, like DVB-C,T,S, DVD, BluRay, Video on demand.
- Extended Profile: streaming applications with latency of 2s and more, like internet streaming using IP/RTP and RTSP



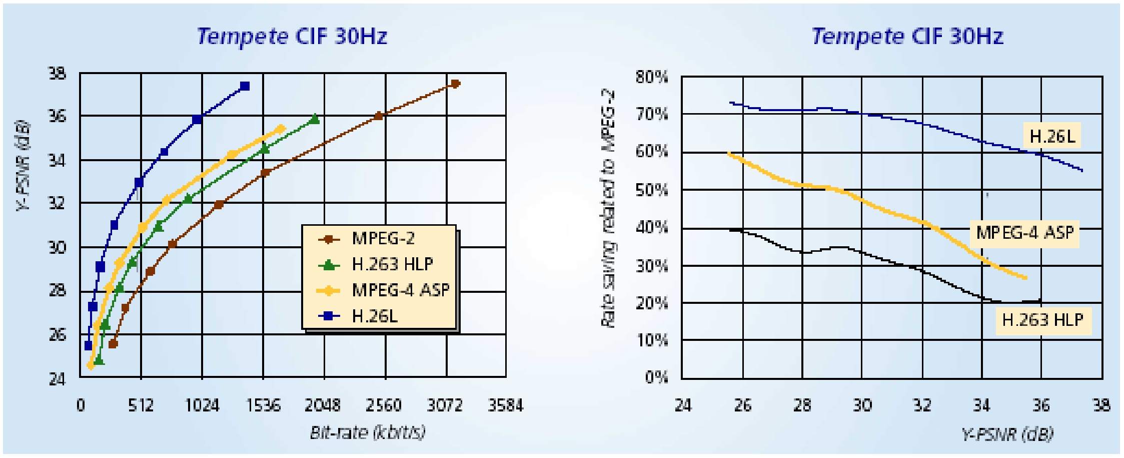
Coding Tools	Baseline	Main	Extended
I and P Slices	X	X	X
CAVLC	X	X	X
CABAC		X	
B Slices		X	X
Interlaced Coding (PicAFF, MBAFF)		X	X
Enh. Error Resil. (FMO, ASO, RS)	X		X
Further Enh. Error Resil (DP)			X
SP and SI Slices			X

In 2004, the fidelity range extensions have been standardised as Amendement 1 (FRExt). Besides providing support for higher bitdepth (up to 12 bit) and for chroma sampling up to 4:4:4, several additional functionalities have been introduced:

- Adaptive block size in the transform (now also 8x8 blocks are supported); transform block size is the same or smaller than the block size of prediction;
- 8x8 luma prediction in intra-prediction, technically similar to 4x4;
- Encoder-specified perceptual-based quantisation scaling matrices;
- Efficient lossless compression using prediction and entropy coding for specific regions (enhancing the PCM raw mode);
- Losselss color transform;
- More colour spaces;

In the figure, we see an example for the improvement in terms of rate-distortion performance as compared to older standards, which is impressive given the fact that many people thought that after MPEG-4 visual, no more significant improvements will be possible.

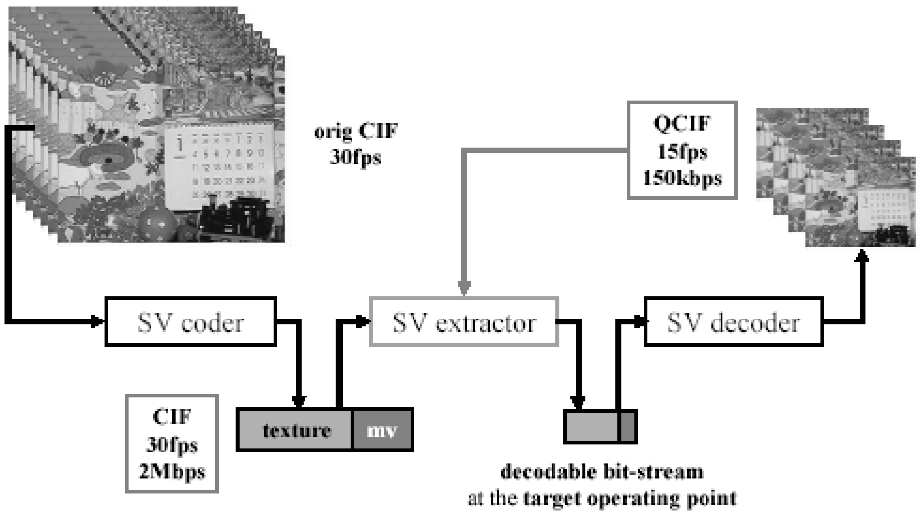
Coding Tools	High	High 10	High 4:2:2	High 4:4:4
Main Profile Tools	X	X	X	X
4:2:0 Chroma Format	X	X	X	X
8 Bit Sample Bit Depth	X	X	X	X
8x8 vs. 4x4 Transform Adaptivity	X	X	X	X
Quantization Scaling Matrices	X	X	X	X
Separate Cb and Cr QP control	X	X	X	X
Monochrome video format	X	X	X	X
9 and 10 Bit Sample Bit Depth		X	X	X
4:2:2 Chroma Format			X	X
11 and 12 Bit Sample Bit Depth				X
4:4:4 Chroma Format				X
Residual Color Transform				X
Predictive Lossless Coding				X

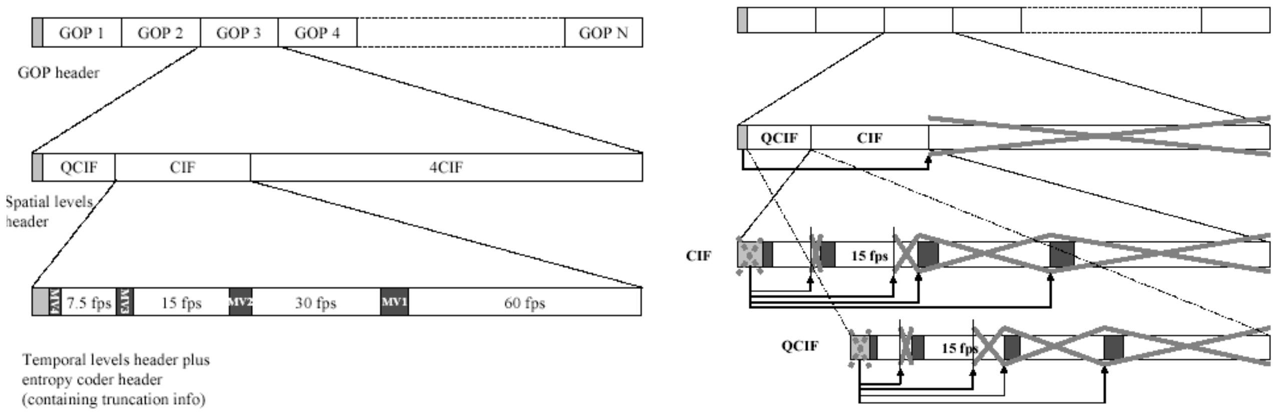


4.7 Scalable Video Coding

Although scalable video coding has been defined in the corresponding Scalable Profiles of MPEG-2 and MPEG-4, these techniques have hardly been adopted due to a dramatic decrease of compression performance in case of more enhancement layers being applied. The basic idea of scalable video coding (SVC) is (again) illustrated in the figure, where an SV extractor is applied to generate a much smaller bitstream out of the larger one which can be decoded by the SV decoder to a full video at reduced resolution and reduced temporal rate (spatial and temporal scalability).

The idea is that the organisation of the video file enables a fast extraction of video data with lower temporal, spatial and quality "resolution". The example of a bitstream based on GOPs shows a scalable organisation of data in each GOP, where successive quality layers with respect to spatial resolution can be seen. In each spatial layer, several temporal layers are organised in a hierarchical manner. It is also shown which parts of the data are required for decoding at CIF resolution @ 15 fps.

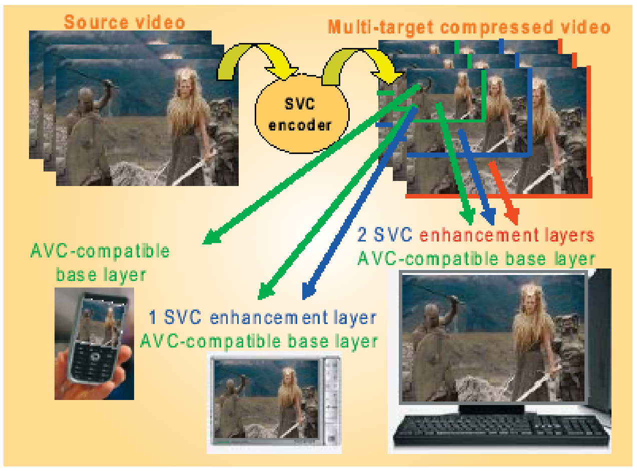


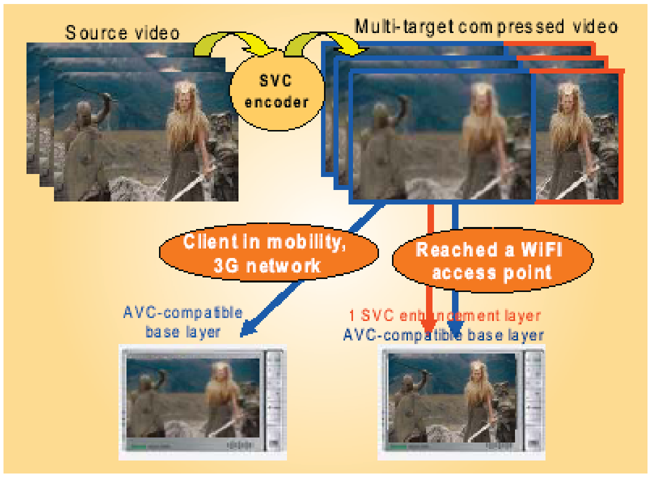


4.7.1 H.264 SVC

This video codec is the youngest child of the JVT (Joint Video Team, a common pool of people from the MPEG and ITU-T video standardisation groups). During the standardisation, a majority of proposals was based on scalable wavelet video coding (see next section), only two H.264 based schemes have been submitted. Close to the date where the final candidate algorithm should have been fixed, still the wavelet-based schemes have been considered the best candidates. At this time, a team from the Berlin Heinrich Hertz Institute (HHI) improved their submitted H.264 variant significantly in a tremendously fast development process which finally lead to the current H.264 SVC standard (in short SVC). One major argument for the decision was that the baselayer of the entire scheme uses H.264 which is a very attractive feature of course enabling backwards compatibility and the possibility to (partially) decode a SVC video on each H.264 decoder.

The following example shows sensible application context for resolution and quality scalability. Resolution scalability is required in case we want to support a heterogeneous pool of display devices with screens supporting various resolutions, while quality scalability perfectly supports transmission (to identical displays) over a heterogeneous network structure.


Source video
Multi-target compressed video
SVC encoder
AVC-compatible base layer
2 SVC enhancement layers
AVC-compatible base layer
1 SVC enhancement layer
AVC-compatible base layer


Source video
Multi-target compressed video
SVC encoder
Client in mobility, 3G network
Reached a WIFI access point
AVC-compatible base layer
1 SVC enhancement layer
AVC-compatible base layer

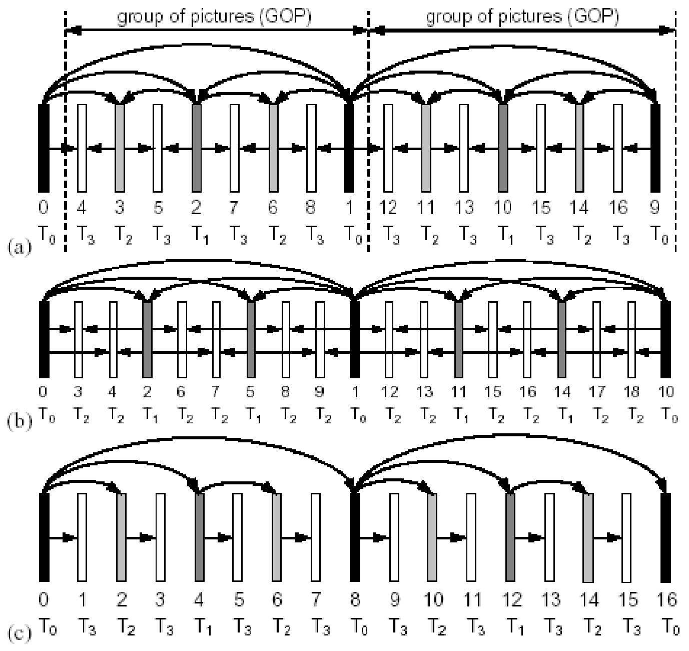
Temporal scalability is provided when the set of corresponding access units can be partitioned into a temporal base layer and one or more temporal enhancement layers. Temporal layers are indentified by an identifier T which starts at 0 at the base layer and is incremented for each additional enhancement layer. In general temporal scalability can be achieved by restricting motion-compensated prediction to reference frames with a temporal identifier that is less or equal to the identifier of the frame to be predicted. While all older MPEG standards offer temporal scalability to some degree, SVC offers more flexibility due to the reference picture memory control, which allows the coding of sequences with arbitrary temporal dependency. Therefore, H.264 did not need to be changed for this approach, only the NAL signalling has been extended.

Temporal scalability with dyadic temporal enhancement is efficiently provided by hierarchical B frames as shown in the upper example in the figure. Enhancement layer frames are typically coded as B frames, where the reference frame lists 0 and 1 are restricted to the temporally preceding and succeeding frame, with a temporal layer identifier less the the identifier of the predicted picture. The set of frames between two frames at the temporal base layer is called group of pictures (GOP) here.

The reference picture lists can contain more than a single frame in each direction and they can also contain frames with the same layer identifier as the predicted picture. Further, also non-dyadic hierarchical prediction structures are possible as shown in the middle example in the figure, which provides two independently decodeable subsequences with 1/9th and 1/3rd of the full frame rate. The chosen temporal prediction structure does not need to be constant over time, i.e. it can be made adaptive to video content.

The third example in the figure shows a structure without coding delay where the coding order of the frames corresponds to the "natural order in the video". In order to guarantee an optimal quality, the fidelity of the base layer has to be the highest one, since it is the base for all predictions. There are several heuristics how to adjust quantisation among layers but this can also be done with costzly rate-distotion optimisation.

Spatial scalability is achieved in the same way as it is done in the older standards based on multi layer coding where each layer corresponds to a supported spatial resolution and is referred to by a dependency identifier D (again 0 for the base layer and incremented for each successive enhancement layer). In each spatial layer, inter-frame prediction and intra prediction is done as for single layer coding. But in order



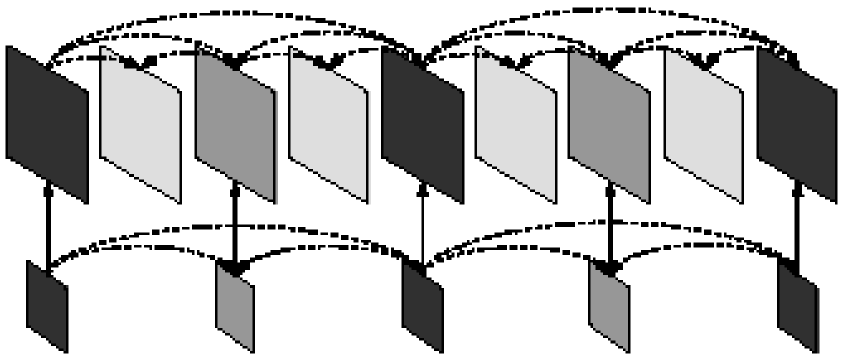
to improve coding efficiency as compared to simulcasting different spatial resolutions, inter-layer prediction techniques are introduced as shown in the figure.

In order to limit memory requirements, the same coding order has to be used in all layers. The representations with different spatial resolutions at a given time instance form an access unit, but as can be seen in the figure, lower layer frames do not need to be present in all access units, thus enabling the combination of temporal and spatial scalability. In order to optimize coding efficiency it is a must to use as much information from the lower statial resolution in the prediction process in addition to exploitation of redundancy within a layer. An SVC encoder can mix these strategies in a flexible way.

The are basically three modes supported to do inter-layer prediction (which are indicated in the corresponding header information):

- Inter-layer motion prediction: the macroblock partitioning and associated motion vectors for the enhancement layer are derived from the reference layer, only a residual is encoded. Also scaled MV from the reference layer can be used as predictors for conventional macroblock types.
- Inter-layer residual prediction: residual data from the reference layer is upsampled and used as prediction for the residuals of the predicted layer data.
- Inter-layer intra prediction: the corresponding signal from the reference layer is upsampled and used in the intra-prediction process as it is done within a single frame. This is only allowed in case the reference layer data is coded in intra-prediction mode to limit complexity of the decoder. Note that this is similar to the spatial scalability modes in MPEG-2 and MPEG-4, however here this is neatly integrated with the H.264 intra-prediction scheme.

There is no restriction to dyadic resultion steps and there is also a mode where the enhancement layer

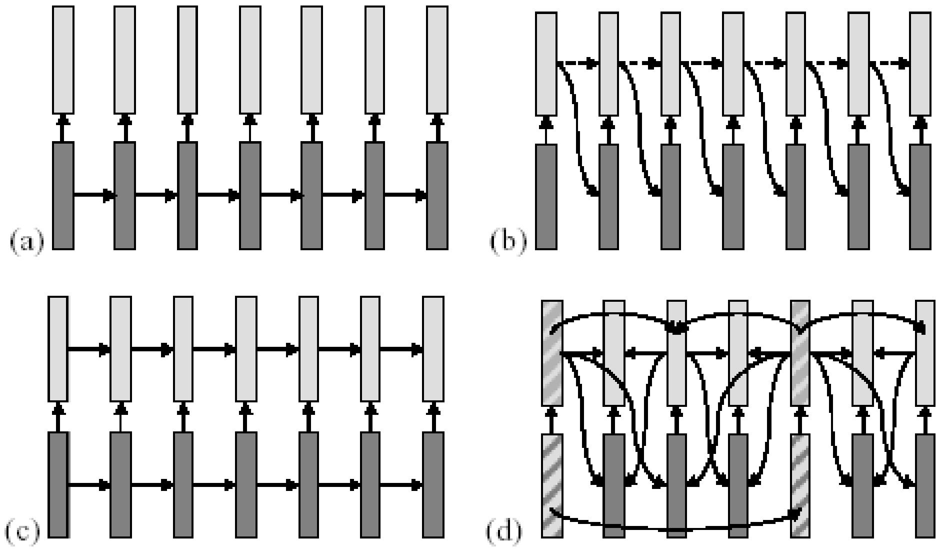


only contains higher resolution data for a selected part of the base layer, which can even be modified from frame to frame ("enhancement layer cropping").

Quality scalability can be achieved in two ways. "Coarse grain quality scalable coding (CGS)" uses almost the identical concept of spatial scalability except for the fact that the different layers exhibit an identical resolution. Apart from that, very similar techniques for inter-layer prediction are used in addition to the intra-layer prediction techniques. Refinement of texture information is usually obtained by re-quantising the residual in the enhancement layer with a smaller quantiser step size as is being used in the preceding CGS layer. The number of supported rate-points is limited by the number of layers and increasing the number of layers decreases compression performance. Therefore, the second mode, "medium-grain quality scalability (MGS)" is included in the standard. In MGS, any enhancement layer NAL unit can be discarded from the stream thus leading to a packet-based quality scalable coding scheme.

A critical issue in packet-based quality-scalable coding is the design of the motion-compensated prediction process, since it determines the trade-off between enhancement layer coding efficiency and "drift". Drift describes the effect of desynchronisation of the motion-compensated prediction loops at the encoder and the decoder due to missing enhancement layer data which is used in the prediction process. The figure illustrates different ways how to cope with this problem.

In the figure, (a) corresponds to MPEG-4 fine-grained scalability (FGS) where drift is completely omitted since only base layer information is used in the prediction process. The drawback is reduced coding efficiency. The approach as used in the Scalable Profile of MPEG-2 shown as (b) in the figure is the other extreme since always the highest quality reconstruction is used for motion-compensated prediction. This is highly efficient, since only a single reference picture is used at a single time instance. On the other hand, any loss of enhancement layer data causes drift. The approach shown as (c) in the figure is similar to spatial scalable coding in MPEG-2 and MPEG-4 and uses two motion compensation loops. While the base layer is not influenced by missing enhancement layer data, the enhancement layer is. SVC employs a strategy illustrated as (d) in the figure which is based on "key frames". For such frames, both the base layer as well as the enhancement layer reconstructions are potentially available for motion compensated prediction and a flag indicates which one is used. All frames of the coarsest temporal resolution are coded as key frames and only for these frames the base layer reconstruction is inserted in the reference frame buffer. Thus, no drift is introduced in this lowest layer. All temporal refinement layers use the highest available quality for reference, which enables high coding efficiency for these frames. Drift propagation is limited to neighbouring frames of higher temporal layers.



The tradeoff between drift and coding efficiency can be controled by the GOP size and the number of hierarchy stages. Note that both the quality scalability structure of MPEG-2 (no frame is coded as key frame) and the MPEG-4 FGS (all frames are coded as key frames) basically represent special cases of the SVC key frame concept.

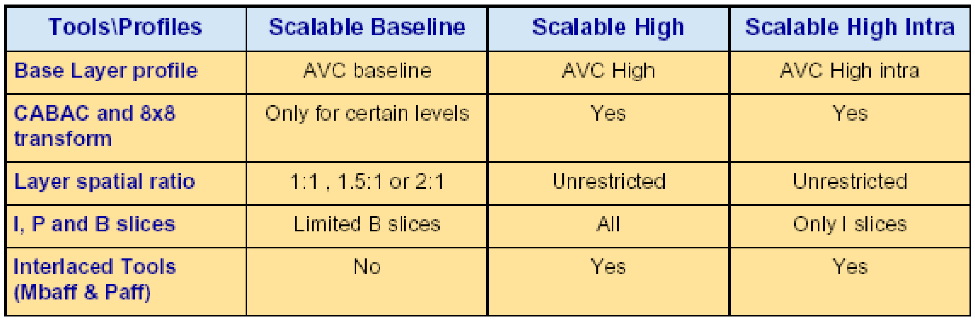
In SVC, the basic concepts of the three scalabilty modes can be combined, but a SVC stream does not need to support all types of scalability. As the support of spatial and quality scalability comes with a loss in coding efficiency, the amount of scalability needed for a target application should be carefully judged when emplyoing SVC. The overall coding structure is organised in "dependency layers D" corresponding to spatial resolutions. Quality refinement layers inside dependency layers are identified by a quality identifier Q. If a reference layer for a dependency layer contains different quality representations, that one used for inter-layer prediction needs to be signaled. For quality refinement layers Q, always the next lower identifier Q-1 is used for inter-layer prediction. Switching between different dependency layers is only supported at specified switching points, whereas for quality layers this can be done at any access unit. All slice data NAL units for a time instance together with no or more non-VCL NAL units form an access unit. All slices of an access unit share the same temporal layer label T. In this way all three scalability modes can be combined.

SVC supports a fourth scalability mode based on Region-of-Interest coding, which is implemented based on the concept of slice groups, however, the shape of the RoI is restricted to patterns that can be represented as a collection of macroblocks.

In order to support efficient bitstream manipulation (i.e. the extraction of partial bitstreams with a reduced spatio-temporal resolution and / or bitrate), all NAL units not required for decoding should be removed from the bitstream. For this purpose, parameters like the dependency identifier D, the quality identifier Q, and the temporal identifier T need to be known for each coded slice NAL unit. Furthermore, it needs to be known which NAL units are required for inter-layer prediction of higher layers.

For enabling this functionality the 1-byte NAL header of H.264 is extended by additional three bytes for SVC NAL unit types. In addition to encoding parameter D, Q, T, and the inter-layer prediction information, also a priority identifier P is encoded, which signals the importance of the enclosed information with respect to rate-distortion performance. For the H.264 compliant base layer, standard NAL units are employed, which do not the include the extended SVC NAL unit header. Since an SVC decoder needs to utilise some of the related information for decoding, prefix NAL units precede all non-SVC VCL NAL units and contain the SVC NAL unit header extensions.

Three profiles for SVC have been specified as shown in the figure. The Baseline profile is intended for conversational and surveillance applications requiring low decoding complexity and tools for interlace coding are not supoorted. Support for spatial scalable coding is restricted to resolution ratios of 1.5 or 2 between successive layers in both directions and cropping is restricted to marcoblock-aligned patterns.



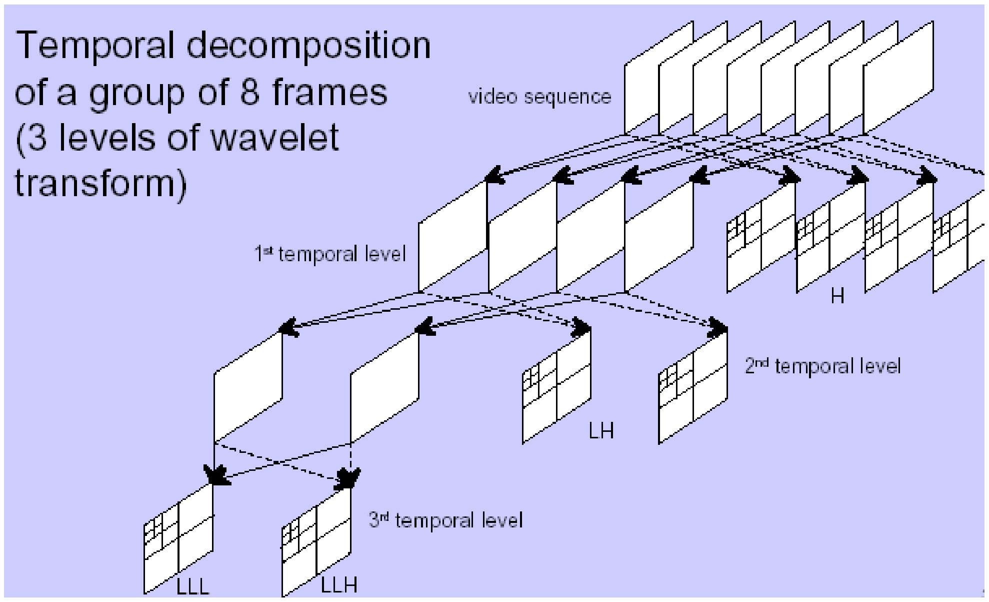
Tools\Profiles	Scalable Baseline	Scalable High	Scalable High Intra
Base Layer profile	AVC baseline	AVC High	AVC High intra
CABAC and 8x8 transform	Only for certain levels	Yes	Yes
Layer spatial ratio	1:1 , 1.5:1 or 2:1	Unrestricted	Unrestricted
I, P and B slices	Limited B slices	All	Only I slices
Interlaced Tools (Mbaff & Paff)	No	Yes	Yes

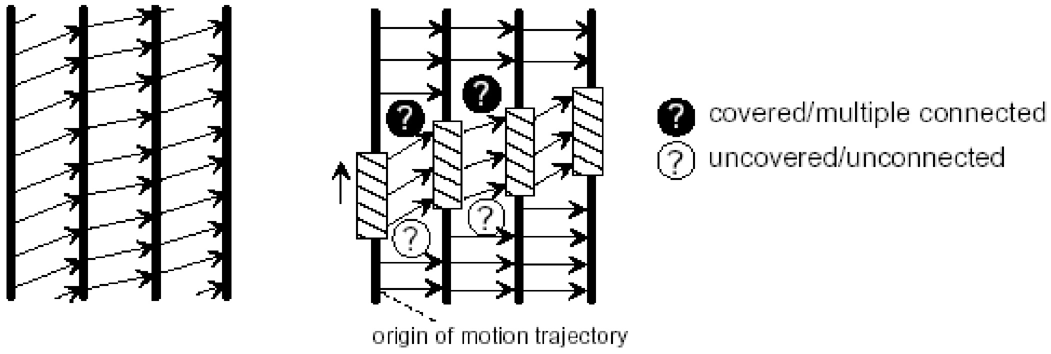
For the High Profile, targeted towards broadcast, streaming, storage, the described restrictions are removed. For the Baseline and the High profiles, the included H.264 base layer is compliant to the Baseline and the High profile of H.264, respectively. Bitstreams conforming to the High Intra Profile contain only IDR frames for all layers and the same set of coding tools as for the High Profile are included. This profile is intended for professional applications.

4.7.2 3D Scalable Wavelet Video Coding

The competitors of SVC during the standardisation process have been based on a three dimensioal wavelet transform, which means that 1-dimensional transforms are applied to the temporal direction in addition to two spatial orientations. The transform into the third dimension can be seen als the replacement strategy for the ME / MC process as used in all the standards described so far. These techniques have their strengh in a better combined scalability and much finer scalability granularity, however, for low granularity SVC has better coding efficiency and a H.264 compliant base layer.

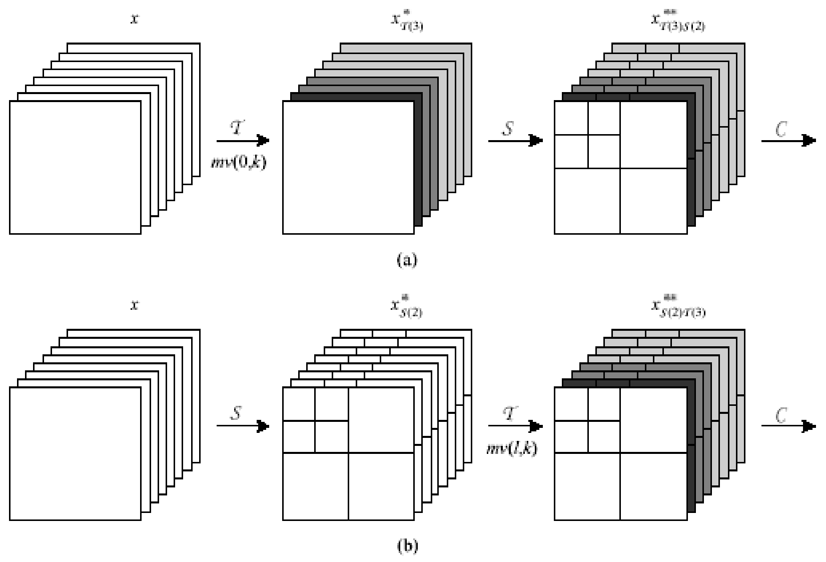
However, when applying the temporal transform in the direction simply orthogonal to the spatial orientations, this would correspond to a MC assuming zero motion between frames. All movements (be it object movements or camera movements) reduce the temporal correlation and thus, the temporal transform can no longer efficiently exploit the correlations present in the data. This is where the key-concept ot motion-compensated temporal filtering (MCTF) comes in. Based on a ME process, for each pixel a “motion trajectory” is determined, which indicates the displacement of a pixel during successive frames. Virtually these temporally connected pixels are fed into a 1-D vector onto which the temporal decomposition is applied. MCTF can make use of ME information in one direction only (the Haar transform is then applied), but also techniques for longer filters involving ME information in both directions have been developed. In this case, MV coding is more expensive.


Temporal decomposition of a group of 8 frames (3 levels of wavelet transform)
video sequence
1st temporal level
H
2nd temporal level
LH
3rd temporal level
LLL
LLH



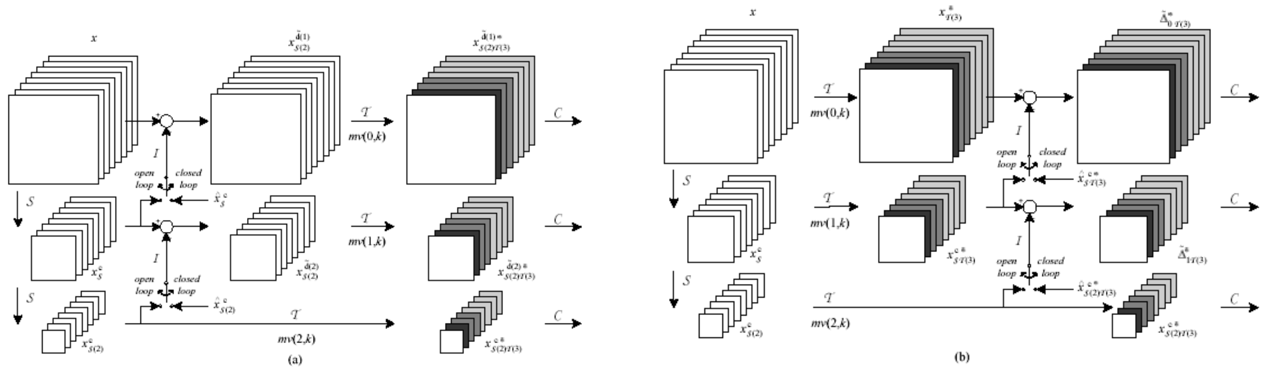
Of course, some problems arise related to “unconnected pixels” which are due to occlusions or suddenly appearing objects. Motion compensated prediction is used locally to “filll the gaps” arising from this phenonemon. There are two basic architectures supporting this idea which are differentiated depending on in which stage of the process ME and the temporal decomposition is conducted. In **t+2D** schemes, ME is applied in the spatial domain followed by the MCTF process. Subsequently, the resulting temporal frames are decomposed in a 2-D wavelet transform for further decorrelation. In this approach it is very easy to integrate 2-D wavelet coding schemes since these can be directly applied to the temporal frames (e.g. JPEG2000 and other scalable schemes can be applied here).

The second approach **2D+t** is based on first applying a spatial 2-D wavelet transform to the frames of the original video. Subsequently, ME is done in the wavelet domain and the temporal decomposition is applied along motion tranjectories filled with wavelet transform coefficients. Both approaches have some disadvantages: t+2D cannot support spatial scalability well, since the motions vectors derived from full resolution are not optimal for the lower spatial resolution (this problem is tackled by using scaled MV for example). On the other hand, 2D+t suffers from the fact the ME cannot be done easily in the wavelet domain due to the lack if shift invariance in decimated schemes. Therefore overcomplate representations



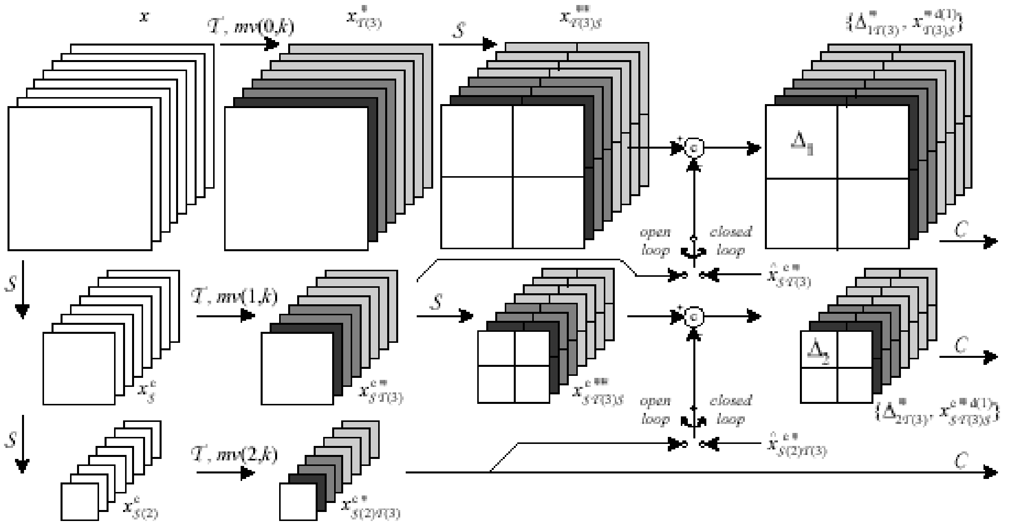
without downsampling need to the applied for the ME process. There have been proposals suggesting to combine both schemes in an adaptive process based on the nature of the video content.

A further improvement has been done by incorporating multi-scale pyramids in coding schemes of this type which are usually called “2D+t+2D” schemes. Here, spatially scaled versions of the video are used while the coding scheme uses inter-scale prediction (ISP) in order to exploit multi-scale representation redundancy. As it was the case before, two different architecures have been designed which can be differnetiated depending on where the pyramidal residue information is generated, before or after the MCTF.



One of the solutions discussed in SVC standardisation is similar to the latter technique, however, the ISP technique is slightly different due to an additional wavelet transform being conducted to the temporal frames before ISP.

An additional argument against this type of techniques during standardisation was the higher memory requirements compared to SVC which makes hardware implementations more cost intensive. Also, is was found to be questionable if the superiority in terms of flexibility with respect to scalability is actually needed in applications. As we have seen in JPEG2000 or MPEG-4, not all capabilities of standardised codecs are


x
$\mathcal{T}, mv(0,k)$
$x^*_{\mathcal{T}(3)}$
$\mathcal{S}$
$x^{**}_{\mathcal{T}(3)\mathcal{S}}$
$\{\Delta^*_{1\mathcal{T}(3)}, x^{*d(1)}_{\mathcal{T}(3)\mathcal{S}}\}$
Δ_1
open loop
closed loop
C
$\hat{x}^{c*}_{\mathcal{S}\mathcal{T}(3)}$
$\mathcal{S}$
$\mathcal{T}, mv(1,k)$
$x^c_{\mathcal{S}}$
$x^{c*}_{\mathcal{S}\mathcal{T}(3)}$
$x^{c**}_{\mathcal{S}\mathcal{T}(3)\mathcal{S}}$
Δ_2
$\{\Delta^*_{2\mathcal{T}(3)}, x^{c*d(1)}_{\mathcal{S}\mathcal{T}(3)\mathcal{S}}\}$
$\hat{x}^{c*}_{\mathcal{S}(2)\mathcal{T}(3)}$
$\mathcal{T}, mv(2,k)$
$x^c_{\mathcal{S}(2)}$
$x^{c*}_{\mathcal{S}(2)\mathcal{T}(3)}$

actually application relevant.

4.8 Non-standard Video Coding

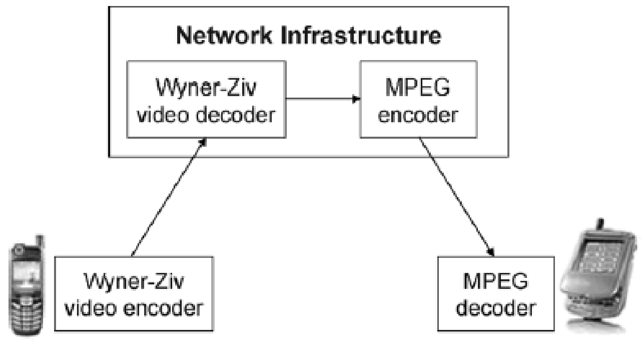
Many video coding algorithms have been developed which never made it into standardisation due to various reasons, since standardisation is always a political issue involving major companies and their corresponding interests also with respect to patent issues. Therefore, superior technology is only one argument during standardisation.

Besides the multitude of 2-D schemes with ME/MC based on block-based DCT, a similar scheme based on 2D global wavelet transform has been developed by the BBC (the DIRAC codec) which is also available as open source and does a reasonable well job. This is just to show that it is also possible in principle to use other transforms instead of the DCT with good results.

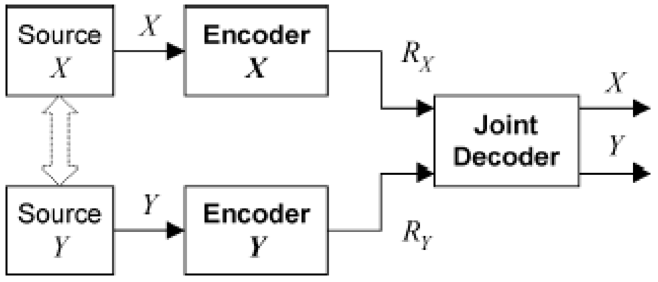
3D video coding schemes have been proposed applying a wide variety of techniques (Wavelet, Fractal, DCT) among which 3D fractal video coding is again interesting from a scientific point of view since here, 3D block self similarities are exploited and decoding is conducted on arbitrary 3D blocks by applying the stored transformations. In all of these schemes, the high computational encoding complexity and high momory requirements make them unsuited for general purpose standardisation despite of sometimes excellent coding performance.

4.8.1 Distributed Video Coding

All video coding schemes discussed so far are highly assymetric in terms of computational complexity since encoding is much more demanding (usually orders of magnitude) as compared to decoding. While this is no problem in common streaming and retrieval-based applications, more recent application contexts like sensor networks or mobile capturing devices exhibit weak capturing and encoding devices which makes the algorithms available so far entirely unsuited for these environments. Distributed video coding aims at **distributing** the computational load in different manner between encoder and decoder thus enabling schemes with lightweight encoding but heavy decoding. In the figure a scheme including network based transcoding is shown which results in lightweight encoding and decoding in mobile capturing and display devices.



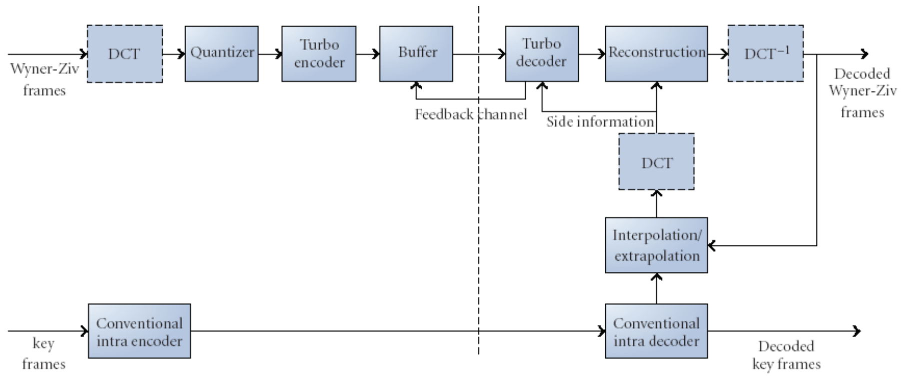
The basic facts behind this recent technology are based on the Slepian-Wolf (lossless) and Wyner-Ziv (lossy) coding principles. The fundamental idea is that correlated source signals X and Y can be compressed independently – in case they are jointly decoded based on side information derived from their correlation the same rate distortion performance can be achieved as in case of joint encoding.



When talking about video coding, the correlated signals X and Y represent temporally correlated frames of a video. In classical video coding, they are jointly encoded using ME / MC to exploit the temporal correlation. In distributed video coding the main idea is to shift ME to the decoder side and using the generated motion information is the side information required for joint decoding.

The two classical architectures have been developed at Stanford and UC Berkeley, repspectively. We briefly describe the former. So-called key frames are encoded using a conventional intra-frame encoding scheme which are passed to the decoder. The Wyner-Ziv frames (which are the ones where the distributed coding principle is applied) are subject to (optional) DCT transform and quantisation. The coefficients are fed into a Turbo encoder (efficient error correcting coding scheme) on a bitplane basis, the resulting data is fed into a buffer. In the decoder, the Turbo decoder starts decoding the MSB of the coefficients. Based on the side information coming from the decoded key frames it can be determined after some more requested bit layers from the buffer, what a good quality reconstruction will be (ME is done based on the coefficients approximation coming from the Turbo decoder and the decoded key frames and the higher bitplayer information is taken from the found match in the key frames).

To enable a more effective ME in the decoder, it has been suggested to transfer also additional (robust) hash values from the encoder to the decoder. Hash values can be generated quickly and only generate a small bitrate overhead. They can be used to aid ME since when hashing is also applied to the key frames, corresponding hash values between those sent by the encoder (generated from the Wyner-Ziv frames) and those computed by the decoder indicate a possible match between key frame block and Wyner-Ziv block.



While the overall concept is appealing, it has to be stated that compression performance is far below that of classical MC-based schemes used in standardisation, however, clearly improving upon compression performance of purely intra-coding scheme. Therefore, for specific application scenarios this technology is an interesting alternative – significant research efforts are still going on in this area.

Kapitel 5

Audio Coding