CHAPTER 10 Digital Representation of Information

All information in today's computers are represented by a sequence of binary digits "0" and "1".

10.1 Binary coding

There are many formal representations for a value. If we represent data in a digital form it can be processed by a machine. Computers use the simplest possible set of symbols, the "binary digits" consisting of only 2 symbols, denoted by "0" and "1". A binary digit is called "bit". If 8 bits are grouped together they are called "byte". The reason for using the simplest possible code lies in the computer hardware (see Section xxxx). Both symbols should not be confused with the decimal digits 0 and 1. However, if we encode the decimal value 0 (res. 1) with the binary symbol "0" (res. "1") it means dual numbers. But we will see soon that we can encode any value or fact with binary digits. Using a binary alphabet is not an invention of the computer age. The Morse alphabet (point and dash) was used long before for encoding messages in the navigation.

Morse code	Letter
.	A
- ...	B

There is a widely used coding for characters and digits. The ISO (International Organization for Standardization) adopted the ASCII (American Standard Code for Information Interchange) as standard ISO 646 for coding letters, digits, and some special characters with 7 binary digits.

You may find that this coding is not really convenient for coding numbers and doing arithmetic operations with it. This observation indicates that we have to look for a more appropriate coding of numbers. Another question is if we can code anything with a binary code.

hex val	0	1	2	3	4	5
0	NUL	DLE	SP	0	@	P
1	SOH	DC1	!	1	A	Q
2	STX	DC2	"	2	B	R
3	ETX	DC3	#	3	C	S
4	EOT	DC4	$	4	D	T
5	ENQ	NAK	%	5	E	U
6	ACK	SYN	&	6	F	V
7	BEL	ETB	’	7	G	W
8	BS	CAN	(	8	H	X
9	HT	EM	)	9	I	Y
A	LF	SUB	*	:	J	Z
B	VT	ESC	+	;	K	[
C	FF	FS	,	<	L	\
D	CR	GS	-	=	M	]
E	SO	RS	.	>	N	ˆ
F	SI	US	/	?	O	_

Figure 10.1 ASCII coding table (the ISO 646 US variant)

10.1.1 Universality of binary coding

We have already shown a possible coding of letters and digits using ASCII. Therefore we can code text and numbers by concatenation. But what’s about data that has a continuous range of values (analog values)?

Let’s assume a physical value that changes over time like the sound of music, or the intensity of light. In this these cases we do not have discrete, but continuous values. We cannot code the situation with a limited number of binary values. But if we define a range of values to have the same code we have an approximation for all values in that range. We call this method *discretization*. As the inaccuracy introduced depends on the size of the range, we can limit the “error” to any predefined value by making the range sufficiently small.

The speed of a car driving from city A to city B is subject to continuous change over time. We can draw a continuous function speed = f(time). We may measure the speed in km/h for each minute by taking a speed sample. Also, we will measure the speed only with a resolution of full 10 km/h. This produces for each one-minute time interval a natural number representing the speed. The natural numbers are easily encoded in the dual system but also any decimal number can be digitally coded as we will see in 10.3.2. By discretization we can always assign a digital code to virtually anything which can be named or represented as a value. The principle of discretization followed by digitizing is used for many applications (see audio, image, and video data)

10.2 Coding of characters

For the coding of characters we need a coding table. This is a discrete mapping function m from the set of characters A to a set of binary values B.

Figure 10.2 Speed over time, discretization, digital coding.

$$\begin{aligned}&\mathrm{m}: \mathrm{A} \rightarrow \mathrm{B}\\ &m(a) = b, \text{ with a} \in \mathrm{A}, \mathrm{b} \in \mathrm{B}\end{aligned}$$

If the length (number of bits) is n for all $b \in B$, we call it an n-bit coding. The ASCII code is a 7 bit code. With an n-bit code 2^n different characters can be coded. In many European countries special characters are used like ä, ç, õ, or even a complete different alphabet. This is why the ASCII code has been extended to 8 bit to include national characters (ISO 8859-1, called Latin-1). But on a global scale this was not sufficient. The 16-bit ISO 10646 standard (Unicode) is able to code 65536 ($= 2^{16}$) different symbols. To be compatible with the former standard, the first 256 codes represent the same symbols as ISO 8859-1.

10.3 Coding of numbers

In our daily life numbers are represented in the decimal number system. There are 10 digits (the symbols 0, 1, 2, ..., 9) for constructing a natural number. We could use the ASCII code for these 10 symbols.

Example: 15 = 00110001 00110101 (each digit ASCII coded)
The disadvantage is that it is not convenient to do arithmetic computation on this binary code.

10.3.1 Coding of integer numbers

If we choose our binary code for a decimal number n to correspond with the dual number d then arithmetic operations come easy. Like with decimal numbers the most significant bit (MSB) is written first and the least significant bit (LSB) is written last. This is why the binary symbols are mostly written as 0 (zero) and 1 (one).

Definition: (natural number coding)
Let $n \in N$. The dual number d is called a *dual code* for n if $d_{dual} = n_{dec}$.

We can compute an equivalent dual number for any natural number by the following algorithm (Horners Scheme): Let n be a natural number. We divide n repeatedly by 2 and use the remainders (0 or 1) as digits for the dual number in reverse order until we reach 0 for the quotient.

Example: Let 13 be a number in decimal notation.
We divide the number repeatedly by 2.
13 : 2 = 6 remainder 1
6 : 2 = 3 remainder 0
3 : 2 = 1 remainder 1
1 : 2 = 0 remainder 1
Result: $13_{dec} = 1101_{dual}$

The basic arithmetic operations for integer dual numbers are defined by the following tables:

+ (addition)		
0 + 0	=	0
0 + 1	=	1
1 + 0	=	1
1 + 1	=	0 and carries 1 to the next position

* (multiplication)		
0 * 0	=	0
0 * 1	=	0
1 * 0	=	0
1 * 1	=	1

Please note the analogy to the binary logic. If you associate 0 with “false” and 1 with “true” then the addition (+) corresponds to the logical XOR (exclusive OR) and the multiplication (*) corresponds with the logical AND. This means that we only need to implement electronic circuits for XOR and AND functions to enable arithmetic operations. In fact, only their interpretation will be different. In 12 we will see how to do arithmetical and logical operations in detail by the electronic circuits of a computer.

How about negative numbers and subtraction? If we use 1 byte for the binary code $2^8 = 256$ numbers can be represented. It is desirable that the number range is symmetrical to zero and the addition of two binary numbers should be executed in the same way, no matter if a number is negative or not. In particular, the addition of a positive number n with its negative counterpart should yield zero (n + (-n) = 0) without to consider the signs separately. This can be reached with a two's complement coding. The code table looks like this:

Decimal	binary 2's complement
0	00000000
1	00000001
2	00000010
126	01111110
127	01111111
-128	10000000
-127	10000001
-2	11111110
-1	11111111

Now 1 + (−1) is 00000001 + 11111111 = 00000000 and the carry bit is ignored.

You can always test easily if a number is negative in this coding scheme: just look if the first bit (Most Significant Bit, MSB) is set.

10.3.2 Coding of decimal numbers

Beside integer numbers we need to code fraction numbers. We can extend our 2's-complement coding scheme to map numbers with decimal fraction in the following way:

Definition: (decimal number coding)
Let $z \in Q$ (Q denotes the set of all fraction numbers). The dual number d with a possible fraction part is called a *fractional dual code* for z if $d_{dual} = z_{dec}$.

Let $z \in Q$. We decompose z into an integer part n and a fraction part r $(0 \leq r < 1)$, i.e. $z = n + r$. We code n like shown before. The fraction part r is coded with an extended Horners Scheme by stepwise division with the negative powers of 2 while doubling the remainders.

Example: Let 13.375 be a fraction number in decimal notation. We decompose the number into 13 and 0.375. The integer part is codes as above.
$13_{dec} = 1101_{dual}$
The fraction 0.375 is coded as follows:

0.375	:	1/2	=	0 remainder 0.375
75	:	1/2	=	1 remainder 0.0.25
5	:	1/2	=	1 remainder 0

Resulting in: $0.375_{dec} = 0.011_{dual}$. Finally we have $13.375_{dec} = 1101.011_{dual}$

Non-periodic decimal fraction numbers may have a periodic dual number coding. This will lead to rounding errors.

Exercise: Calculate the dual number coding of the decimal fraction 0.1.

The disadvantage of the above coding scheme is not only the periodic fraction part but also the possible unwanted precision of big integer values. This is often the case in scientific and experimental computing. The instruments

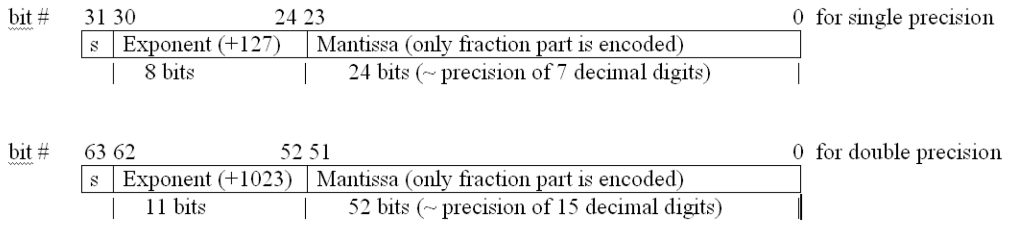

bit # 31 30 24 23 0 for single precision
s
Exponent (+127)
Mantissa (only fraction part is encoded)
8 bits
24 bits (~ precision of 7 decimal digits)
bit # 63 62 52 51 0 for double precision
s
Exponent (+1023)
Mantissa (only fraction part is encoded)
11 bits
52 bits (~ precision of 15 decimal digits)

Figure 10.3 Layout of ANSI/IEEE 754 single and double precision floating point.

deliver often very big numbers with only 3 or 4 digits of precision. It would be wasted storage if we had to code it in 2's complement. As genius way for coding a huge range of numbers values with a limited precision is the floating point coding which has been inspired by the scientific number representation.

Definition: (floating point number coding)
Let $z = m2^p$ be a rational number. We call m the "mantissa" and p the "exponent". If $1 \leq m < 2$ we call the representation "normalized". Let n be the number of bits we want to use for our coding. Then we reserve 1 bit for the sign, p bits for the mantissa and q bits for the exponent ($n = 1 + p + q$). With this ratio the precision will be p binary digits and the number value range is defined by the 2^q. The more bits we assign to the mantissa, the higher the precision, and the more bits we assign to the exponent, the greater the value range will be (10 bits correspond to 3 decimal digits approximately).

A disadvantage is that the number n is limited so that not every decimal number is exactly represented as floating point number. There are two cases in which "rounding errors" may occur. First, the decimal number like 0.1 leads to a periodic dual number which cannot be represented in p bits no matter how large p is chosen. Second, the decimal number z has more digits than can be represented with a given mantissa. So, for example, z and z+1 will be coded with the same bit-pattern.
Due to the "rounding error" problem you should never compare two floating point numbers for equality, like f1 = f2. It is better to check if the difference of both numbers is relatively small, like |(f1-f2)/f1 |< e, where e is sufficiently small.

There is the widely accepted standard ANSI/IEEE 754 who exploits a 32-bit and a 64-bit floating point representation in an optimal manner. The 32-bit floating point number dedicates 1 bit for the sign, 8 bits for the exponent, and 23 bits for the mantissa.

Example: (13.375 as ANSI/IEEE 754 floating point coding)
$13.375_{dec} = (1.337510^1)_{dec} = (1.718752^3)_{dec}$,
$1.71875_{dec} = 1.10111_{dual}$,
$3_{dec} = 11_{dual}$
The standard only codes the fraction part of the normalized mantissa and

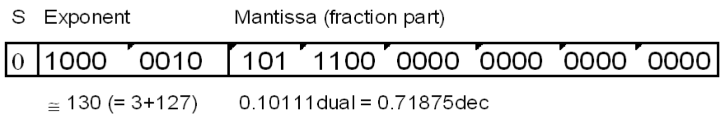
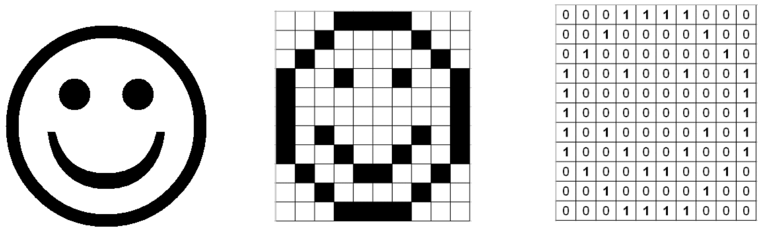


Figure 10.4 Layout of floating point number 13.375



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

Figure 10.5 Simple b/w image (smiley) and its rasterized form.

the exponent with an offset of -127. As 3 = 130 - 127 the exponent will be 130 = 10000010_{dual}. The layout of this coding is shown in figure 10.4

The arithmetic operations on IEEE 754 floating point numbers are implemented in the hardware of a modern computer to achieve a higher performance than with software.

10.4 Coding of images

Images represent analog data. In the simplest case, a black-and-white (b/w) image has information about the "color" (black or white in this case) of a continuous area of the image. To simplify the situation the image is divided into *pixels* (picture element). This is done like in figure 10.5. First the image is dissected (rasterized) into a finite number of rectangular locations, the pixel areas. For each pixel one bit is chosen to represent white (bit 0) or black (bit 1). The more pixels for an image are used, the more details of the image will be retained, but also the more data that is required to represent the image.

If shading or more gradation is desired, we can use a grey scale between black and white. This requires more bits for the coding of the different grey values of each pixel. Normally one byte is used to display 256 grey shadings. For color images the "true color" representation is widely used. Each pixel area has 24 bits assigned to represent the color and luminosity. In the RGB (red-green-blue) model the three colors are intensity coded with one byte each. From figure 10.6 we see that the resulting color is an additive color mixing where (255, 255, 255) stands for white, (0, 0, 0) stands for black, (128, 128, 128) stands for grey, and (255, 0, 0) stands for the primary color red, etc. This color model naturally maps to cathode ray tubes, liquid crystal displays, or plasma displays used for television or computer monitors because these devices use

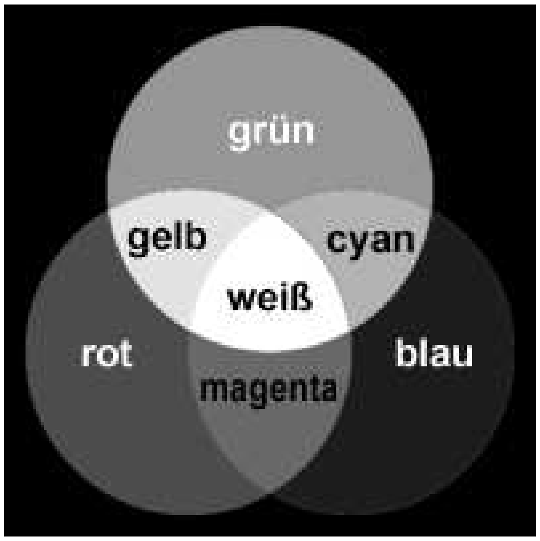


Figure 10.6 Color space for additive color mixing

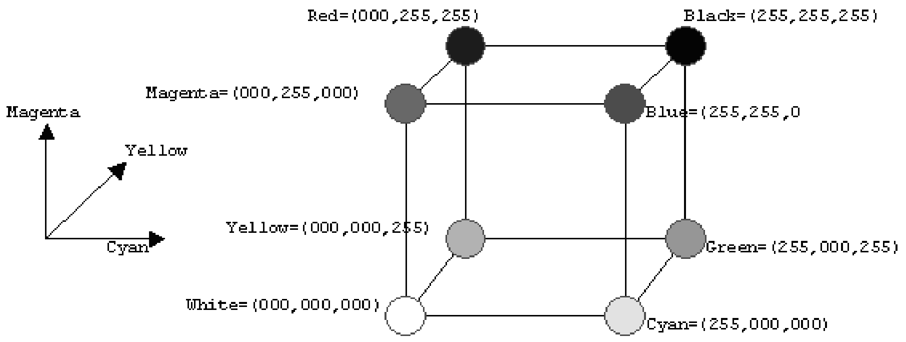


Figure 10.7 CMY-color space

additive color mixing. The device independent bitmap (file suffix .bmp) uses RGB coding with 1, 4, 8, 16, 24 or 32 bits per pixel.

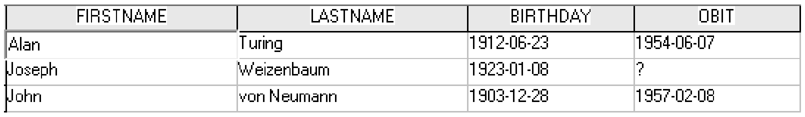
For color printing a subtractive model like CMYK (C=cyan, M=magenta, Y=yellow, K=key plate (black)) is more adequate. The color models RGB and CMYK are inverse to each other, even if a image on screen may not print out in exactly the same colors because neither one represents an absolute color space. The "nominal" conversion RGB to CMYK is:

C = 255 - R; M = 255 - G; Y = 255 - B;

The forth component K (black) is redundant, but the contrast and details will enhance if it is used.

Using 24 or more bits per pixel leads to enormous amount of data used for a detailed image. A digital image of 5 Mega pixels (the typical digital camera resolution of the year 2005) results in $5*10^6 * 24$ bits = 15 Mega byte of data. As the color does not change completely for every pixel compression is generally used to reduce the amount of data.

We can distinguish two classes of compression techniques: *Lossless compression*



FIRSTNAME	LASTNAME	BIRTHDAY	OBIT
Alan	Turing	1912-06-23	1954-06-07
Joseph	Weizenbaum	1923-01-08	?
John	von Neumann	1903-12-28	1957-02-08

Figure 10.8 Example of digitizing a video using difference encoding and image compression

which is used if we need to reconstruct the image exactly. Higher compression rates are achieved with a *lossy compression* if we tolerate to ignore some details in the image. The loss of information is mostly invisible under normal circumstances. The concepts and some details of compression techniques are explained in Section 10.6)

GIF (Graphics Interchange Format) is a code for image formats with lossless compression that is widely used on the World Wide Web, both for still images and for animations. GIF images are limited to 256 colors which prohibit its use for high quality imaging. Each pixel refers to a color defined in a *color table*. The resulting set of color references is compressed by a dictionary based code.

The optional interlacing feature, which stores the pixels out of order in such a fashion that even a partially downloaded image is somewhat recognizable, allowing a user to form an opinion on the image before it is completely retrieved giving the user a chance to abort the loading if it is not the wanted

An example of a highly efficient, but lossy compression is the JPEG (Joint Photographic Expert Group) which can achieve compression rates of 1:60 with nearly unnoticeable loss of detail.

If the compression is high JPEG produces visible square monochrome areas (*artifacts*) where adjacent squares have noticeably different colors thus leading to discontinuous color gradation (see figure 10.12).

10.5 Coding of audio and video signals

A typical movie uses 25 images per second which explains that compression techniques are extremely important for videos. Here not only each image may be compressed but as the content of consecutive pictures in general does not change much, coding only the difference is very efficient for reducing the amount of data. Only a few pictures are usually completely encoded, for all others only the changes between two following images are coded (*difference encoding*). The resulting difference image can effectively further compressed by run length encoding (see figure 10.8). The *MPEG-2 (Moving Pictures Expert Group)* coding which is used for the digital television is an example of such a video compression code.

For the encoding of audio signals the time dimension is disretized to small time intervals, usually twice the actual bandwidth. A sample of the audio signal is taken during each interval. The more bits for each sample are used the higher the resulting precision of the measure will be. With 16 bits per sample the result is a high fidelity audio signal. The smaller the time interval is chosen, the higher the frequency that can be sampled, thus extending the possible frequency range (e.g. 44.1 kHz for CD quality).

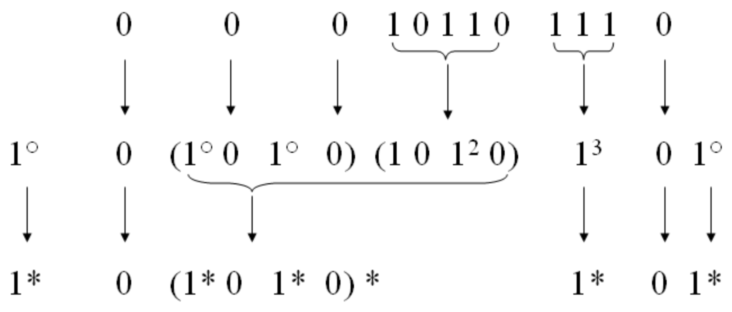


Figure 10.9 MP3 encoder flowchart (source: Fraunhofer-Gesellschaft [2005])

compression factor	coding layer and data rate
1:4	Layer 1 (corresponds to 384 kbps for a stereo signal)
1:6 - 1:8	Layer 2 (corresponds to 256..192 kbps for a stereo signal)
1:10 - 1:12	Layer 3 (corresponds to 128..112 kbps for a stereo signal)

Table 10.1 MP3 compression factor and data rates

The MPEG-1 Audio Layer 3 (called MP3) was developed jointly by the Fraunhofer Institut Integrierte Schaltungen (Germany) and the University of Erlangen (Prof. Dieter Seitzer). This coding standard provides different compression rates delivering different sound quality. The coding is especially efficient because it ignores frequencies that are too high or sound that is to low in volume to be heard. To eliminate these signals a hybrid audio filter is used followed by a discrete cosine transformation. As consequence the MP3 is a lossy compression algorithm.

The perceptual model used for the filter determines the quality of the encoding process. If the quantization noise can be kept below an audible threshold, then the compression results should be indistinguishable from the original signal. In the next coding step only the difference of both stereo channels is used, reducing further the redundancies of the stereo signal.

Quantization of an audio sample is done via a power-law quantizer. In this way, larger values are automatically coded with less accuracy. The quantized values are entropy coded by an Huffman algorithm (see entropy encoding in Section 10.6). As this is a lossless encoding no noise is added to the signal.

With the help from the perceptual model the optimum gain, scale factors, and bit-rate are found by an iterative process. The resulting code is cut into frames with header. An frame consists of 1152 samples and has an equivalent length of 26 milliseconds

The bit stream multiplexer is responsible to include the code from all relevant frequencies. If an audio code does not use all bits for a frame, this free space can be used for signals of an other frame (reservoir technique).

Table 10.1 shows how the compression rate is increasing with each layer of encoding and the corresponding bit rate is falling.

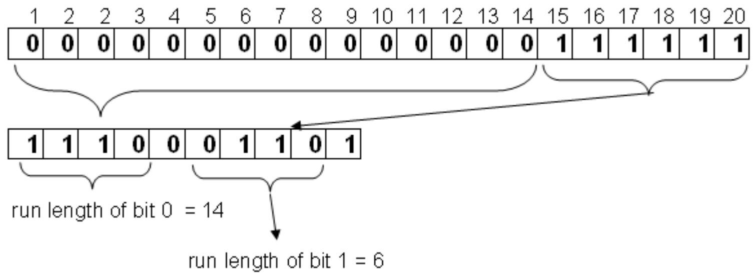


Figure 10.10 Example of lossless compression: coding with a maximum run-length of 15 (= 2^4 - 1).

10.6 Data Compression

In the previous sections we have shown that image, audio and video data produce huge amount of data. In order to save storage space and reduce transmission costs in the case of accessing multi-media data via the internet it is essential to use strong compressing techniques. We have to distinguish two classes of compression techniques:

Lossless compressions reduce the amount of bits used without any loss of information by using an injective mapping. This means that the original coding can be reconstructed with an inverse mapping from the compressed coding.

Lossy compression reduces the amount of bits used by using a non injective mapping. For instance this results in an irreversible loss of accuracy or details in an image.

Images often have lager areas of the same color. The color coding leads in this case to the same data for adjacent pixels, which is a so called *run of data* i.e. a sequence of the same data. This situation is used for a very simple concept to reduce the amount of data as shown in the Figure 10.10.

Definition: (run length coding)
Let p be a data value and let P(n) be a *run of data* of length n with data values p. Coding P(n) as the pair (n,p) is called *run length coding*.

Another technique assigns short code to bit patterns with a high probability to occur. The compression gain depends on the regularity (*entropy*) of the image. These type of codes were developed and studied by Claude E. Shannon, Robert Fano, and have been improved by David A. Huffman as his Ph.D. thesis in 1952 and are therefore known as Shannon-Fano or Huffman codes.

The process of *entropy coding* consists of two stages:

- modeling, and
- coding.

During the modeling phase probabilities are assigned to the different symbols, and in the second stage the coding produces a bit sequence from these probabilities. The higher the probability i.e. the more frequent a symbol is, the fewer bits are used. In order to achieve a good compression rate, the exact probability estimation is needed.

Definition: (entropy coding)

Let p be a data value and let w(p) its relative frequency (probability). If the length (number of bits) of the code for data value p is the reciprocal of w(p) (i.e. length(code(p) = 1/w(p)) we call this an *entropy coding*

Since the model is responsible for the probability of each symbol, modeling is one the most important tasks for this coding concept. As example lets assume we have an image consisting of 2 million colors where the probability is like in Figure 10.11. Normally the coding would need 21 bits for each pixel. Thus resulting in 21*200*200=840000 bits for a 200x200 pixel-image. Using the entropy coding we have 2*200*200*0.9 + 24*200*200*0.1 = 168000 which is a reduction factor of 4 (needing only 20% of the normal storage space).

A more general purpose lossless compression technique is used for GIF (Graphics Interchange Format) and PKZIP (acronym for Phil Katz's ZIP) codes which use the dictionary based algorithm LZW (named after its inventors Abraham Lempel, Jacob Ziv, and Terry Welch). GIF became popular as image code in the Web because it is more efficient than the run-length encoding, and because it supports animated images.

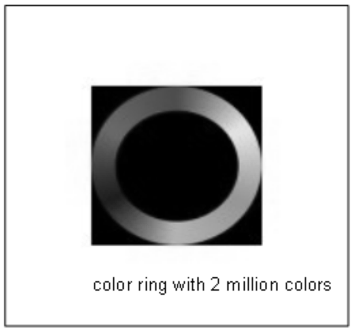
The concept of the method is that it is possible to automatically build a dictionary of previously seen patterns (bit-strings) in the data being compressed. Then each occurrence of the bit-string is replaced by its dictionary value which is much shorter than the original data. The dictionary itself does not have to be included in the compressed data, since the decompressor can build it up the same way the compressor does, and if encoded correctly, will have exactly the same strings that the compressor dictionary had at the same point in the text.

The dictionary starts off with 256 entries, one for each possible character (single byte). Every time a data sequence is not already in the dictionary, it is stored in the dictionary together with the following data element, and the first element from the dictionary is output. This continues until all data is processed.

The output consists of integer indices into the dictionary. These initially are 9 bits each, and as the dictionary grows, can increase to up to 16 bits. A special symbol is reserved for "flush the dictionary" which takes the dictionary back to the original 256 entries, and 9 bit indices. This is useful to make the dictionary adaptive to the data

This use of variably increasing the index size is one of Welch's contributions. Another was to specify an efficient data structure to store the dictionary.

The JPEG 2000 proposed by the "Joint Photographic Experts Group" (JPEG) uses as lossy compression technique the *discrete wavelet transformation* which may result in a compression ratio of 1 : 60 with nearly unnoticeable loss of information. The older JPEG 1992-Format transforms a RGB signal to the color difference scheme YUV (Y = intensity, U = blue-yellow difference, V = red-green difference), then a lossy *low-pass filtering* is applied, the pixels are grouped 8x8, processed with a discrete cosine transformation, quantized (again



color	probability = area in %	code #bits	value*
white	60	2	10
black	30	2	11
all other colors together	10	24	00 + (3*7bits)

*) The first bit position acts as an indicator (1 = 2 bit code, 0 = 24 bit code)

Figure 10.11 Example of lossless compression: entropy coding

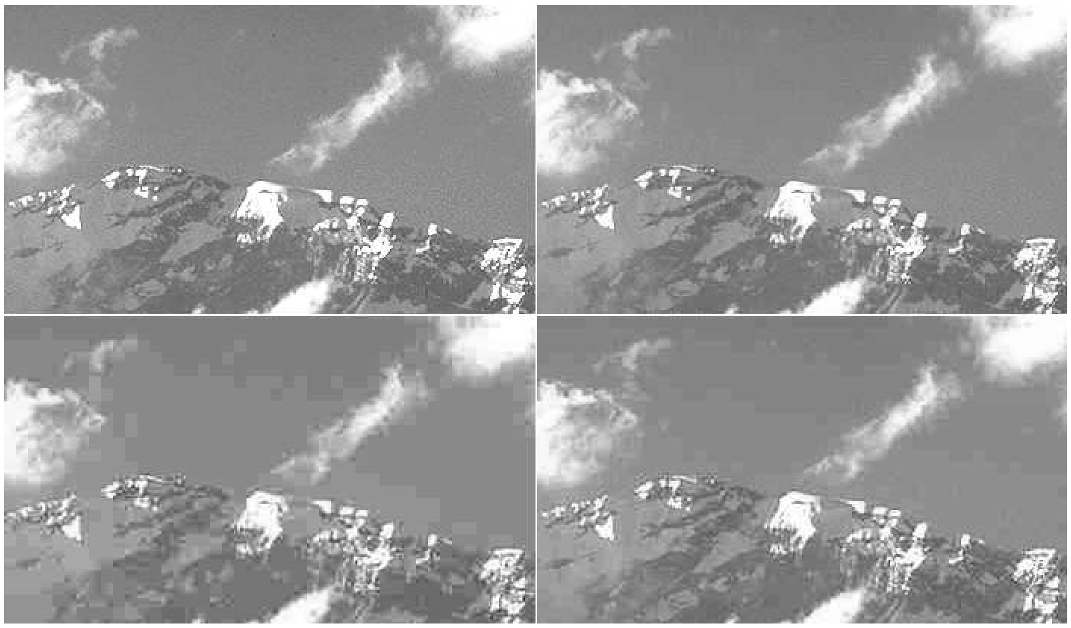


Figure 10.12 Examples of clockwise increasing JPEG compression rates

lossy), rearranged, and newly coded using less bits for patterns with higher occurrence (*entropy coding*). The image is “tiled” into groups of 8x8 pixels for the coding which results in square areas with color distortion and loss of detail. This compression effect isch called an (compression) artifact. The actual JPEG 2000 standard uses wavelet transformation instead of 8x8 pixel grouping and discrete cosine transformation, thus reducing the risk of compression artifacts.
.

10.7 Summary

Computer Science is ...

Exercises

10.1 What is the meaning of life?

Bibliographical Notes