2.1. General Purpose

There are many popular general purpose lossless compression techniques, that can be applied to any type of data.

2.1.1. Run Length Encoding

Run Length Encoding is a compression technique that replaces consecutive occurrences of a symbol with the symbol followed by the number of times it is repeated.

Example

the string 1**11110000003355** could be represented by **15063252**. Clearly this compression technique is most useful where symbols appear in long runs, and thus can sometimes be useful for images that have areas where the pixels all have the same value, cartoons for example.

the string **ABCCCCCCCCDEFGGGGHHH** could be represented by **ABC!8DEFG!4HHH** in which the **!** is a flag and a run of less than 4 is not encoded since we would not gain. .

2.1.2. Relative Encoding

Relative encoding is a transmission technique that attempts to improve efficiency by transmitting the difference between each value and its predecessor, in place of the value itself . Thus the values **15106433003** would be transmitted as **1+4-4-1+6-2-1+0-3+0+3**. In effect the transmitter is predicting that each value is the same as its predecessor and the data transmitted is the difference between the predicted and actual values. Differential Pulse Code Modulation (DPCM) is an example of relative encoding.

2.1.3. Tokanization or Pattern Substitution

Tokanization is the replacement of commonly occuring sequences with a special one byte code which does not occur naturally in the message.

Example

Replace **BEGIN, END, IF** by single bytes codes.

2.1.4. Diatomic Encoding

Substitute a single byte for commonly occuring pairs in a language.

Example

TH IN E_ T_ _A S_ RE HE

2.1.5. Huffman Coding

Huffman coding is a popular compression technique that assigns variable length codes (VLC) to symbols, so that the most frequently occurring symbols have the shortest codes . On decompression the symbols are reassigned their original fixed length codes. When used to compress text, for example, variable length codes are used in place of ASCII codes, and the most common characters, usually space, e, and t are assigned the shortest codes. In this way the total number of bits required to transmit the data can be considerably less than the number required if the fixed length representation is used. Huffman coding is particularly effective where the data are dominated by a small number of symbols.

Example 1

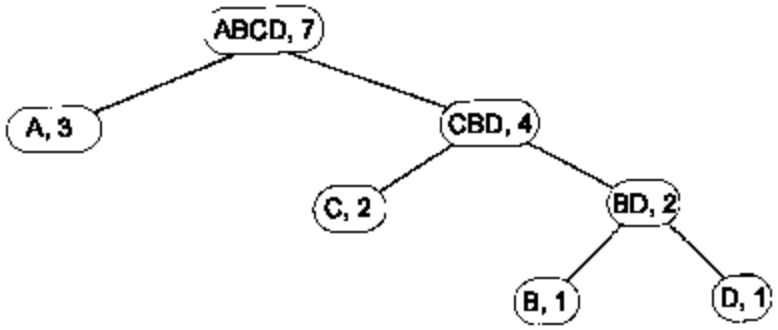
Fixed Length Codes (n=4)	
SYMBOL	**CODE**
A	00
B	01
C	10
D	11

Then the message **ABACCDA** would be encoded as **00010010101100** = 14 bits

Now suppose we use the following code instead:

Variable Length Codes (n=4)	
SYMBOL	**CODE**
A	0
B	110
C	10
D	111

Then the message **ABACCDA** would be encoded as **0110010101110** = 13 bits



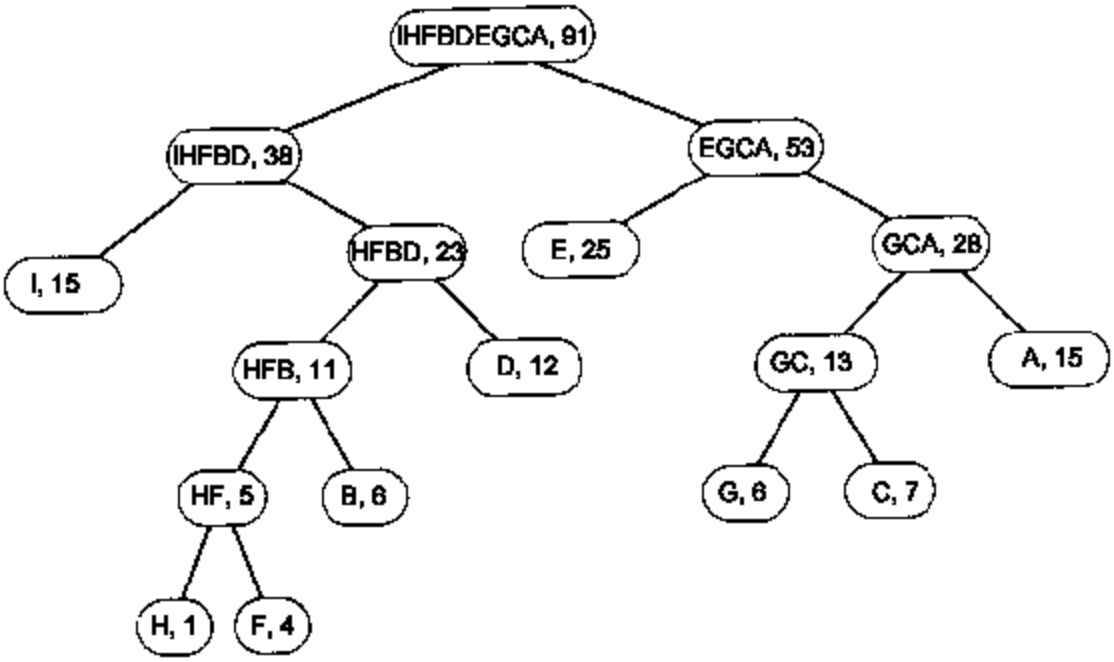
Huffman Tree for Message = ABACCDA

Example 2

Assume a message with the following frequency table:

Variable Length Codes (n=9)		
SYMBOL	**FREQUENCY**	**CODE**
A	15	111
B	6	0101
C	7	1101
D	12	011
E	25	10
F	4	01001
G	6	1100
H	1	01000
I	15	00

Which would lead us to construct the following tree:



2.1.6. Arithmetic Coding

Although Huffman coding is very efficient, it is only optimal when the symbol probabilities are integral powers of two. Arithmetic coding does not have this restriction and is usually more efficient than the more popular Huffman technique. Although more efficient than Huffman coding, arithmetic coding is more complex.

2.1.7. Lempel-Ziv Coding

Lempel-Ziv compressors use a dictionary of symbol sequences. When an occurrence of the sequence is repeated it is replaced by a reference to its position in the dictionary. There are several variations of this coding technique and they differ primarily in the manner in which they manage the dictionary. The most well known of these techniques is the Lempel-Ziv-Welch variation.

2.2 Intraframe

Intraframe compression is compression applied to still images, such as photographs and diagrams, and exploits the redundancy within the image, known as spatial redundancy. Intraframe compression techniques can be applied to individual frames of a video sequence.

Sub-sampling

Sub-sampling is the most basic of all image compression techniques and it is equivalent to intelligently discarding data. Sub-sampling reduces the number of bits required to describe an image, but the quality of the sub-sampled image is lower than the quality of the original. Sub-sampling of images usually takes place in one of two ways. In the first, the original image is copied but only a fraction of the pixels from the original are used. Alternatively, sub-sampling can be implemented by calculating the average pixel value for each group of several pixels, and then substituting this average in the appropriate place in the approximated image. The latter technique is more complex, but generally produces better quality images.

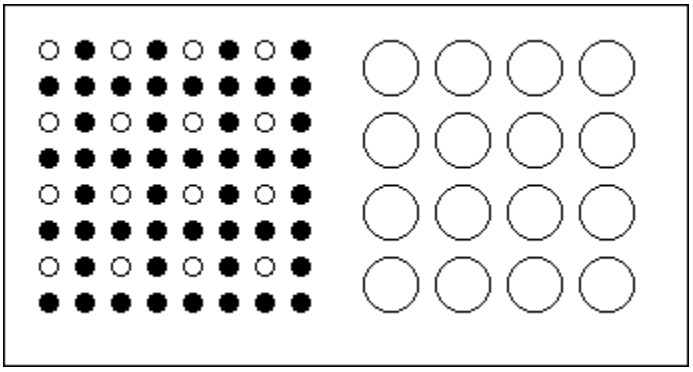


Figure 2.2 2x2 sub sampling with pixel doubling.

For example, an image might be sub-sampled by 2 in both the x and y directions, thus every second line and every second column of the image is completely ignored, as in Figure 2.2. When the sub-sampled image is displayed at the same size as the original image, the size of the pixels is doubled as in Figure 2.3. This is known as *pixel doubling*.

Figure 2.3. Original Lisbon image (left) and a sub-sampled version (right)

When coding colour images, it is common to sub-sample the colour component of the image by 2 in both directions, while leaving the luminance component intact. This is useful because human vision is much less sensitive to chrominance than it is to luminance, and sub-sampling in this way reduces the number of bits required to specify the chrominance component by three quarters.

Sub-sampling is necessarily lossy, but relies on the ability of human perception to fill in the gaps. The receiver itself can also attempt to fill in the gaps and try to restore the pixels that have been removed during sub-sampling. By comparing adjacent pixels of the sub-sampled image, the value of the missing in-between pixels can be approximated. This process is known as *interpolation*. Interpolation can be used to make a sub-sampled image appear to have higher resolution than it actually has and is usually more successful than pixel doubling. It can, however, result in edges becoming blurred.

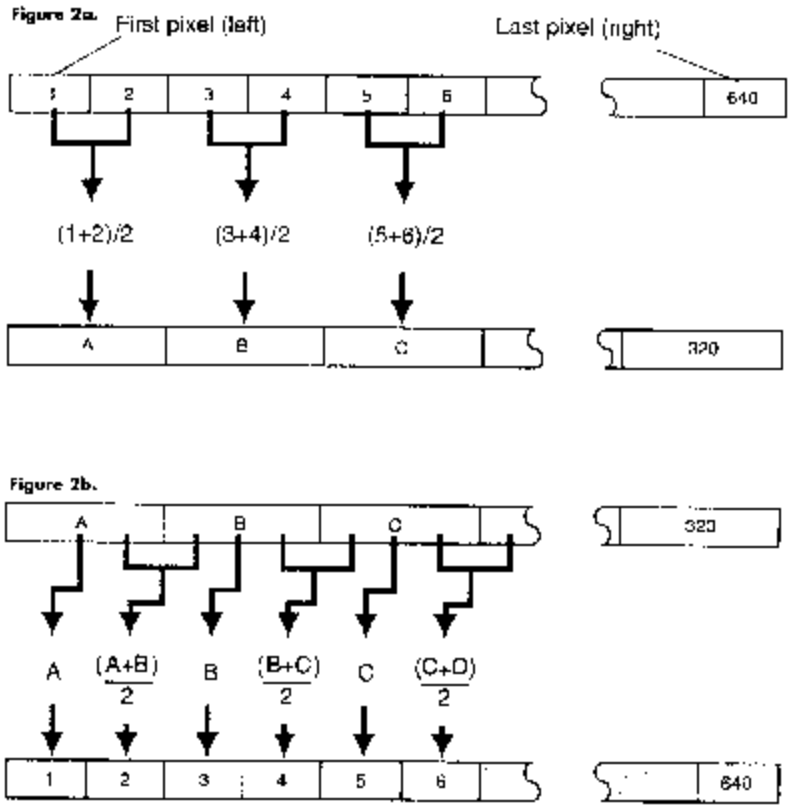


Figure 2. With Horizontal Interpolation enabled during capture (a), adjacent pixels are averaged and stored 320 rather than 640 per scan line. During playback (b), the 320 pixels are spread out over 640 in a way that minimizes loss of detail.

2.2.2. Coarse Quantization

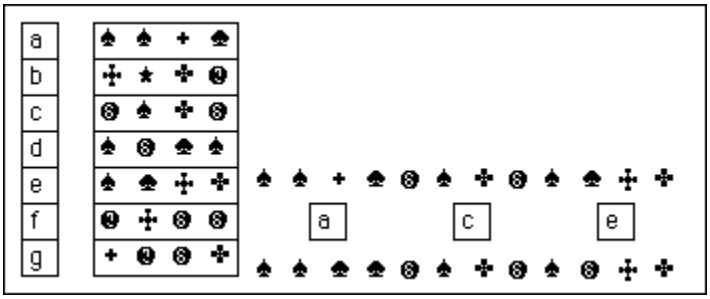
Coarse quantization is similar to sub-sampling in that information is discarded, but the compression is accomplished by reducing the numbers of bits used to describe each pixel, rather than reducing the number of pixels. Each pixel is reassigned an alternative value and the number of alternate values is less than that in the original image. In a monochrome image, Figure 2.4a for example, the number of shades of grey that pixels can have is reduced. Quantization where the number of ranges is small is known as *coarse quantization*.

[2.4a | 2.4b | 2.4c]

Figure 2.4 Images with different numbers of pixel quantization levels. Figure 2.4a uses the most bits per pixel and Figure 2.4c the least.

Photographic quality images typically require pixels of 24 bits, but can be reduced to 16 bits with acceptable loss. Images with 8 bits, however, are noticeably inferior to those with 16 bits per pixel. Coarse quantization of images, often called bit depth reduction, is a very common way of reducing the storage requirements of images.

2.2.3. Vector quantization



[2.5a | 2.5b]

Figure 2.5 Vector quantization. A sequence of symbols (Figure 2.5b bottom) is divided into blocks of four symbols and the blocks are compared to those in a table (Figure 2.5a). Each block is assigned the symbol of the table entry it most resembles (Figure 2.5b middle) which results in an approximation of the original sequence when decoded (Figure 2.5b top).

Vector quantization is a more complex form of quantization that first divides the input data stream into blocks. A pre-defined table contains a set of patterns for blocks and each block is coded using the pattern from the table that is most similar. If the number of quantization levels (i.e. blocks in the table) is very small, as in Figure 2.5, then the compression will be lossy. Because images often contain many repeated sections vector quantization can be quite successful for image compression.

2.2.4. Transform Coding

Transform coding [Penn93] is an image conversion process that transforms an image from the spatial domain to the frequency domain. The most popular transform used in image coding is the Discrete Cosine Transform (DCT). Because transformation of large images can be prohibitively complex it is usual to decompose a large image into smaller square blocks and code each block separately.

Instead of representing the data as an array of 64 values arranged in an 8x8 grid, the DCT represents it as a varying signal that can be approximated by a collection of 64 cosine functions with appropriate amplitudes. The DCT represents a block as a matrix of coefficients. Although this process does not in itself result in compression, the coefficients, when read in an appropriate order, tend to be good candidates for compression using run length encoding or predictive coding.

The most useful property of DCT coded blocks is that the coefficients can be coarsely quantized without seriously affecting the quality of the image that results from an inverse DCT of the quantized coefficients. It is in this manner that the DCT is most frequently used as an image compression technique.