The Big SQUEEZE

A look at compression techniques by Mike Bedford

If it wasn't for data compression, many of todays up and coming technologies could never have come to fruition. Without employing compression, for example, it would have been impossible to cram more than a few minutes of high quality video onto a 120mm diameter disk. Yet, as home cinema enthusiasts are well aware, DVDs are capable of storing a full-length movie at broadcast quality. Much the same argument applies to digital TV. Simply digitising an analogue TV signal would result in a data stream that would require a much higher bandwidth than analogue. When that same signal is subjected to data compression, though, it becomes possible to cram half a dozen programmes into the bandwidth that was formerly required for a single TV station. And for reasons of reducing transmission time, reducing RF bandwidth, and/or reducing the cost of distribution media, data compression is also applied on the Web, for software publishing, and in mobile telephony to name just a few of the more obvious areas.

But although data compression is such a familiar term, at first sight it appears rather counter-intuitive. Surely reducing the amount of data by some means reduces the amount of information it carries and this would appear to be unacceptable. In fact, both these assumptions are incorrect in some cases. First of all it is often possible to reduce the amount of data without loosing any of the information it carries. And in other cases, by accepting some loss of information an even greater degree of compression can be achieved and, furthermore, that loss of information can sometimes go unnoticed. This article – an introduction to the technology of data compression – will look at many of the methods in use today and explain how a quart can be crammed into a pint pot and why, even if we don't quite manage it, it doesn't always matter too much.

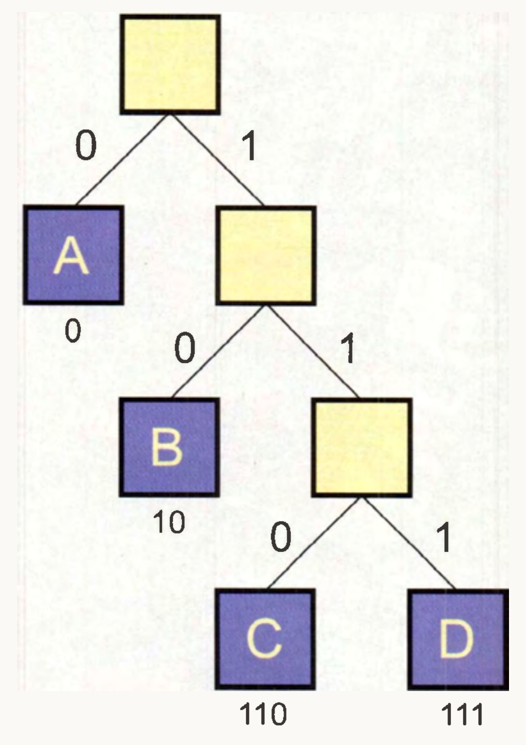


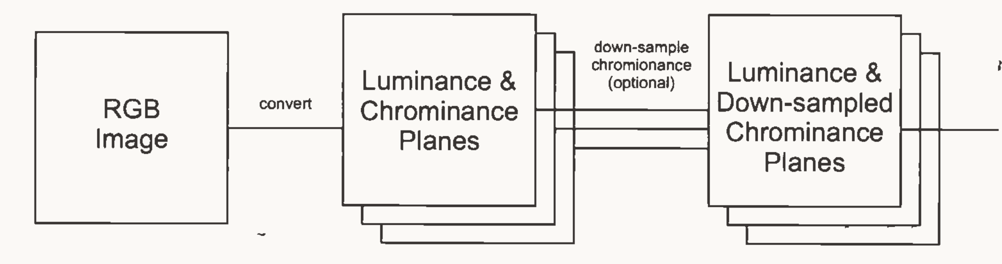
Figure 1.

Lossless Compression

The fact that the volume of data can sometimes be reduced with no loss to the information carried is generally due to the fact that it's represented in an inefficient manner. For example, English text uses the ASCII characters in the range 00 to 7F hexadecimal and it requires seven bits to store any of these characters. For convenience, though, characters are nearly always stored and transmitted as bytes, i.e. eight bits, so one bit in eight is wasted. Clearly, therefore, textual data could be compressed to 87.5% of its original size simply by stripping out these redundant bits. This isn't a practical compression method, of course, because it doesn't achieve an appreciable improvement; I simply give it as an example of how information is often represented inefficiently.

If you're a linguist you'll have noticed that I deliberately referred to English text. And, if we want to be able to write in the common European languages, compressing the data by stripping out the most significant bit won't work since the accented characters which would be required are ASCII characters in the range 80 to FF hexadecimal. Now let's assume that we're writing in French and that, accordingly, an extra handful of characters are required. To use these few extra characters, though, we have to move from seven bits per characters to eight, which seems rather inefficient. But the same argument applies even if we stay with English. The letter J, for example, crops up very infrequently but the ASCII coding scheme assigns it an 8-bit code, just the same as more common letters such as E and T. The problem with ASCII is that it's a fixed length code. When we're dealing with data in which the various symbols have differing frequencies of occurrence, though – and this is common in many types of data, not just text – greater efficiency can be achieved by using a variable length code in which the shorter codes are assigned to the commonest symbols. Compared to a fixed length code, the uncommon symbols will be represented by longer codes but the fact that the very common ones use short codes more than compensates for this and an overall reduction on the amount of data results.

Of course, once we move to a variable length code some method of determining when one code ends and another one starts has to be devised. The most common method is to derive the codes such that no code can start with a bit combination assigned to a shorter code. This is done using a binary tree in which symbols are represented by the leaves of the tree (i.e. the end points) and the code for each leaf is determined by starting at



RGB Image
convert
Luminance & Chrominance Planes
down-sample chromionance (optional)
Luminance & Down-sampled Chrominance Planes

Figure 3.

the top and adding a 1 for each right turn and a 0 for each left turn. Since the symbols are all at the ends of branches, no other symbol can have a route that passes through an already defined symbol that is the property we require. I'm not going to describe how this sort of binary tree is generated but Figure 1 is a portion of a tree for encoding the string "ABBDCAAACBBAABCABAAA". Analysing the string reveals that there are 10 As, 6 Bs, 3 Cs and 1 D so the encoding process must assign the shortest code to A and the longest to D which is exactly what's shown in Figure 1. The string has 20 characters, which if we encoded it in ASCII would occupy 20 bytes. Using this tree, though, the data is reduced to 34 bits or just over 4 bytes. Of course, the definition of the tree must also be placed in the file along with the compressed data and this will reduce the compression ratio somewhat. With long files, though, the overhead imposed by including the tree definition is very small in percentage terms. What I've just described forms the basis of Huffman encoding. This is a common method of lossless compression but it does have a significant disadvantage. Because the data has to be analysed before it can be encoded, it's suitable for encoding a file on disk but not, for example, a stream of data being transmitted via some communication channel.

Another method which achieves much the same sort of compression ratio uses a fixed length code but assigns these codes to differing amounts of data. So for example, one code might represent a single uncommon letter such as J or Z, whereas another code of the same length might represent a whole string of characters that crop up very frequently – " <space> the <space>" for example. This is the method used in the LZW compression algorithm and, unlike Huffman, the coding scheme is built up on the fly so it can be used for compressing data, which can't be analysed in advance. It is used in the common ZIP compression utility and, typically, is able to reduce the size of many types of files by around 50%. Similar compression algorithms have also been built into modems.

Figure 2 shows a small portion of a bitmapped image. You can't make out any features in the image, admittedly, but this is much the same sort of thing you might see if you took a very small portion of any bitmapped image and expanded it like this. Normally, this sort of image would be represented in a file as a string of values, one per pixel, indicating the colour of that pixel. In this case we'll assume that there are 256 possible colours and, accordingly, each pixel has to be represented by a single byte. If we assume that the code for black is 0, the code for red is 63, for blue is 127 and for white it is 255 it should be clear that the data which represents this image is 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 255, 63, 63, 255, 255, 255, 255, 255, 255, 63, 63, 127, 255, 255, 255, 255, 255, 63, 63, 127, 127, 127, 127, 127, 127. Now clearly, although I've stated that there are 256 possible colours, only four are actually used and this suggests scope for data compression. However, let's forget about the sorts of methods we looked at in the previous section and think, instead, of a method that would work even if most of those 256 colours were represented in an image. A method which is often used for compressing bitmaps is called run-length encoding and takes advantage of the fact that if one pixel is a particular colour then successive pixels will often be in the same colour. The following is the same data represented by run-length encoding: 255, 25, 0, 14, 255, 1, 63, 2, 255, 6, 63, 2, 127, 1, 255, 5, 63, 2, 127, 6. The figures are actually in pairs – the first figure in each pair is the colour and the second figure is the number of times that colour appears in succession. In this case the data has been compressed to 31.25% of its original size. In real world applications much greater compression ratios can often be achieved.

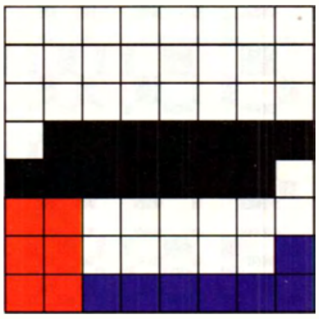
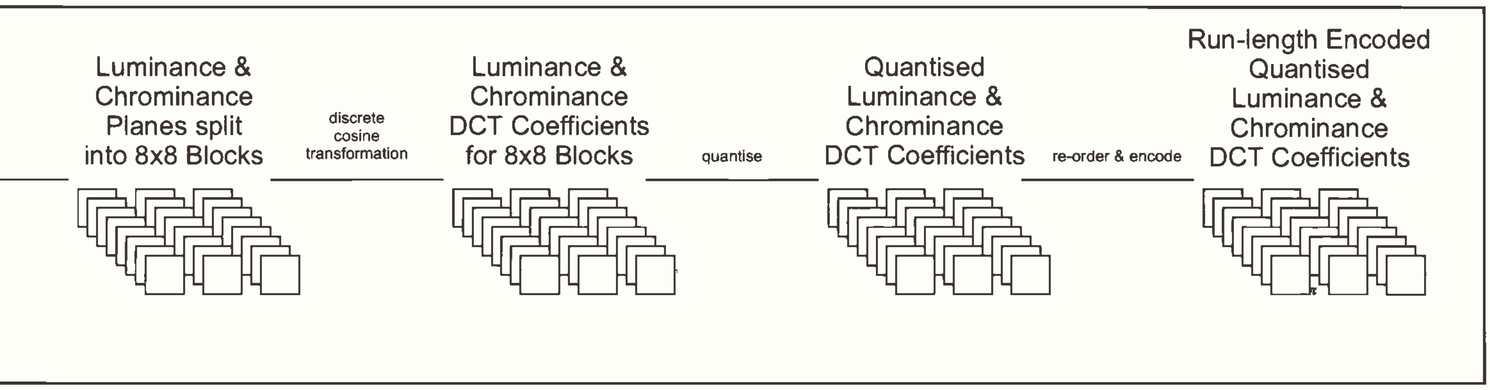


Figure 2.

So far we've looked at three methods of data compression. All three methods have the potential for reducing the amount of data even though none of them will actually loose any of the information that is represented by the data. As such these methods are referred to as lossless. Any method of compression is lossless if the data remains the same when subjected to compression followed by decompression. And for many applications, lossless compression is absolutely essential. Software distribution is such an application. If the compression process causes even a single bit in a program to change then there's potential for that program to crash when it's run. But although the lossless methods are the only ones available in certain areas, the amount of compression achievable tends to me modest. Although there are exceptions – run-length encoding a bitmap consisting of nothing more than a single black dot in the middle of a white background, for example – lossless compression rarely achieves a compression ratio of much more than 50%. To go beyond this sort of performance, lossy compression has to be employed. This is a compression method in which information is lost by compression followed by decompression. At first sight this might seem totally unacceptable. However, if the data represents an image or audio information, for example, the human senses may fail to notice a certain degree of information loss. And even when the loss is noticeable – perhaps as a reduction in the sharpness of a picture – it could be that this is a price worth paying for the reduction in the volume of data.

DCT Compression

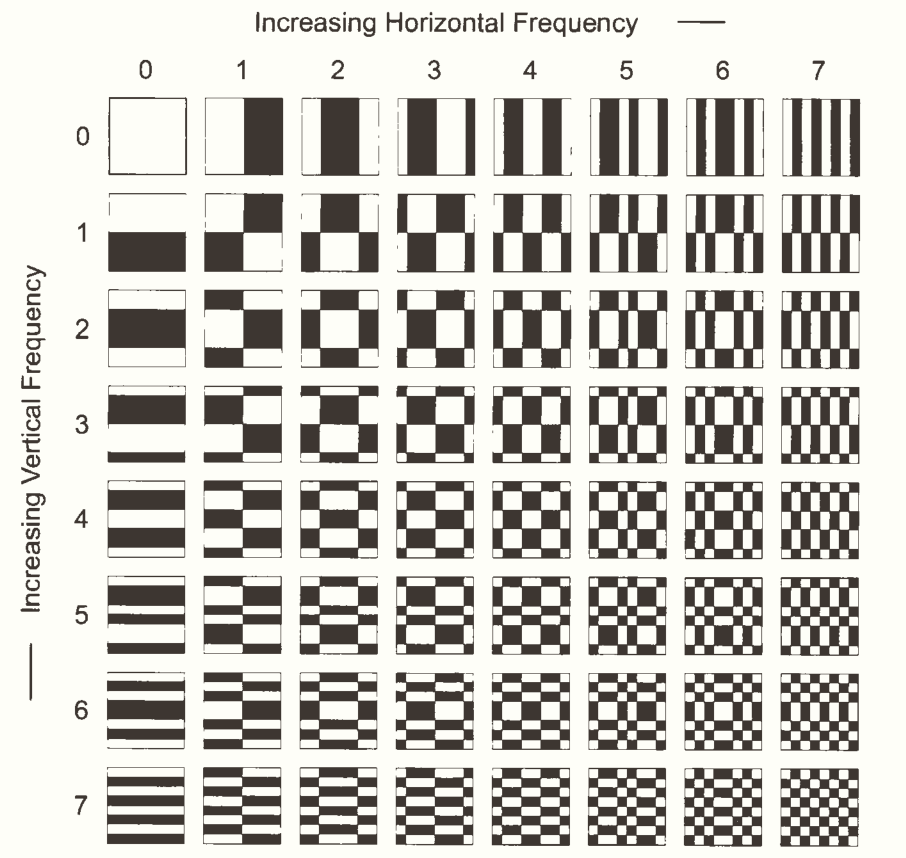
The first lossy compression scheme that we'll look at makes use of something called the discrete cosine transformation. It's integral to the JPEG file format which is used for representing images on the Web


Luminance & Chrominance Planes split into 8x8 Blocks
discrete cosine transformation
Luminance & Chrominance DCT Coefficients for 8x8 Blocks
quantise
Quantised Luminance & Chrominance DCT Coefficients
re-order & encode
Run-length Encoded Quantised Luminance & Chrominance DCT Coefficients

and it also forms as part of the MPEG family of compression algorithms which are used for compressing moving images.

The Fourier Theorem states that any waveform can be expressed as the sum of sine waves of differing frequencies and amplitudes. To represent some waveforms accurately an infinite number of sine waves would have to be added together but a waveform could be approximately just by giving the amplitudes of the first few harmonics. A Fourier Transformation is the analysis of a waveform to determine these amplitudes. The Discrete Cosine Transformation (DCT) is a close relative of the Fourier Transformation which is used to split a waveform into frequency components and is at the heart of many methods of compressing photographic images. Figure 3 is a block diagram of the process – this should clarify the following verbal description.

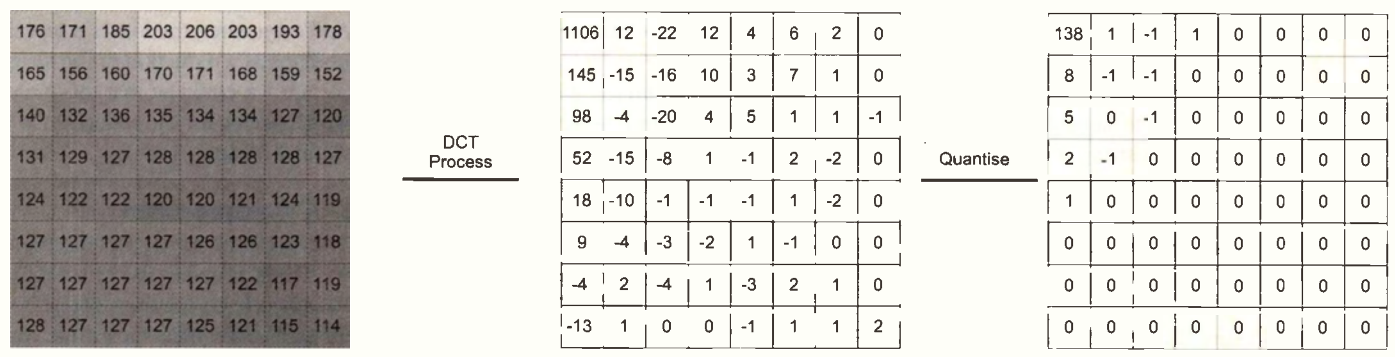
For a colour photograph the first step is to convert the image from the usual RGB form to luminance and chrominance values like those which are used in TV transmission. The luminance contains the basic monochrome image whereas the two chrominance values provides information on the intensity of two of the primary colours (the intensity of the third primary colour is the luminance value minus the two chrominance values). Since the eye is less sensitive to chrominance than luminance information, an optional first step is to re-sample the chrominance data using a coarser grid. Generally a two-fold reduction here goes unnoticed. The other reason for representing the data in this form is because the later compression steps work better on a luminance and chrominance data than on RGB data. It also allows a greater degree of compression to be applied to the chrominance data in these later stages. The next step is to split the luminance and each of the chrominance bitmaps into 8 x 8 pixel blocks and perform a DCT on each of these blocks. Since a



Increasing Horizontal Frequency
0
1
2
3
4
5
6
7
Increasing Vertical Frequency
0
1
2
3
4
5
6
7

Figure 4.

bitmap is two-dimensional, though, this is somewhat more complicated than performing the transformation on one-dimensional data like an audio signal. The output of this stage is a set of DCT coefficients relating to the amplitude of the various combinations of horizontal and vertical frequencies. A verbal description of this is, perhaps, not the easiest thing to understand but Figure 4 should, hopefully, clarify things. Any pattern that can be represented in an 8x8 pixel block can be expressed as the amplitudes of each of these 64 patterns. Therefore, storing or transmitting each of these 64 coefficients can reconstitute the original image. It's important to note, though, that just converting bitmaps into DCT coefficients doesn't result in data compression. However, it is an important first step.



176	171	185	203	206	203	193	178
165	156	160	170	171	168	159	152
140	132	136	135	134	134	127	120
131	129	127	128	128	128	128	127
124	122	122	120	120	121	124	119
127	127	127	127	126	126	123	118
127	127	127	127	127	122	117	119
128	127	127	127	125	121	115	114

8 x 8 block of pixel intensities

1106	12	-22	12	4	6	2	0
145	-15	-16	10	3	7	1	0
98	-4	-20	4	5	1	1	-1
52	-15	-8	1	-1	2	-2	0
18	-10	-1	-1	-1	1	-2	0
9	-4	-3	-2	1	-1	0	0
-4	2	-4	1	-3	2	1	0
-13	1	0	0	-1	1	1	2

DCT Coefficients

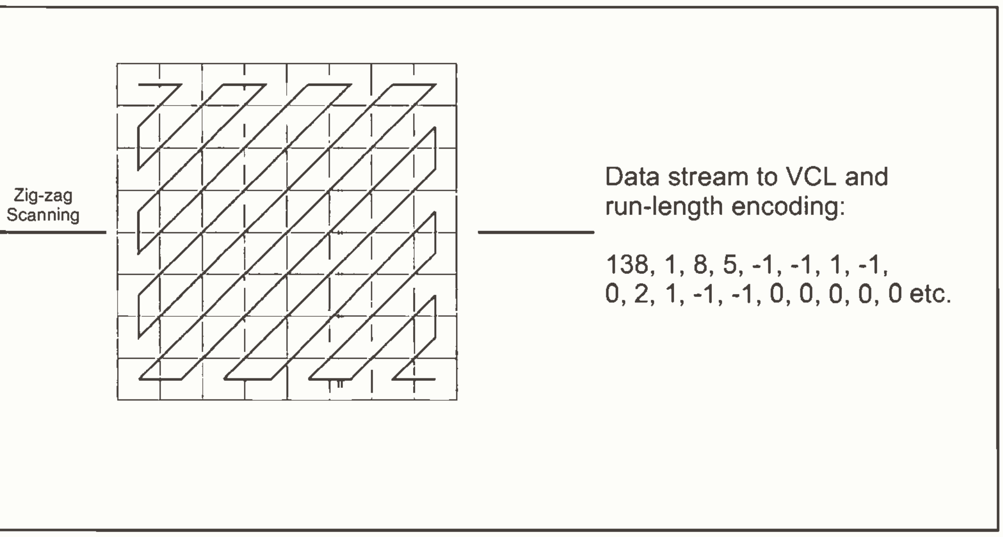
138	1	-1	1	0	0	0	0
8	-1	-1	0	0	0	0	0
5	0	-1	0	0	0	0	0
2	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantised DCT Coefficients

Figure 5.

What does achieve a reduction in the volume of data is the quantisation of each of these samples using different quantisation coefficients for each DCT coefficient. The rationale here is that the eye is more sensitive to low frequency than to high frequency information so the coefficients toward the top left are assigned the most bits and the quantisation becomes more coarse as we move toward the bottom right. The final step is to run-length encode the quantised coefficients but this only achieves a further reduction in the amount of data if the coefficients are read out in a zig-zag manner starting at the top left and working toward the bottom right. By re-ordering the coefficients in this way, and since those toward the bottom right tend to be zero, the zero coefficients are generally consecutive so run-length encoding is especially efficient. Figure 5 shows this process on one 8x8 pixel block of luminance information.

This method of data compression takes account of the fact that the eye is more sensitive to some parts of the overall data stream than others. In this instance, having re-coded the image into the frequency domain, the less noticeable high frequency components can be de-emphasised by using a coarser quantisation coefficient. This sort of reasoning is common in lossy data compression algorithms. However, it also raises the question 'at what point can we safely assume that data can be discarded without having an adverse affect on the final quality?'. This question has resulted in much heated debate, for example, in drawing up the specification for DVD-audio, the up-and-coming replacement for the audio CD. However, it also raises the possibility of having variable compression ratios with a corresponding variation in quality. So, for example, you could choose a low compression ratio and, thereby achieve a high quality result, or you could pick a higher compression ratio and, in so doing, accept that the quality will suffer. JPEG offers just such a range of compression ratios and Web designers have to choose between file size (and hence download time) and quality.


Zig-zag Scanning
Data stream to VCL and run-length encoding:
138, 1, 8, 5, -1, -1, 1, -1, 0, 2, 1, -1, -1, 0, 0, 0, 0, 0 etc.

Frequently, for Web use, a high compression ratio is selected even though this will result in a noticeably inferior image. Figures 6, 7 and 8 show an image – in each case with an inset enlargement of part of that image – compressed using JPEG compression ratios of 2, 127 and 255 respectively. The reduction in image quality with increasing compression ratio is clear to see.

MPEG Compression

The next compression scheme we're about to take a look at is MPEG that is used for moving images. The original MPEG was used for transmitting Web-based video but the newer MPEG-2 is key both to digital TV and to DVD so is at the heart of many of today's high profile consumer products. MPEG employs two quite different forms of compression. The first is referred to as intra-frame compression and involves compressing individual frames so this is much the same as compressing still images. As such, the method used is very similar to the DCT compression algorithm that we looked at in the previous section. I'll say little more about it here, therefore, and move on to the aspect which is unique to compressing moving images – inter-frame compression.

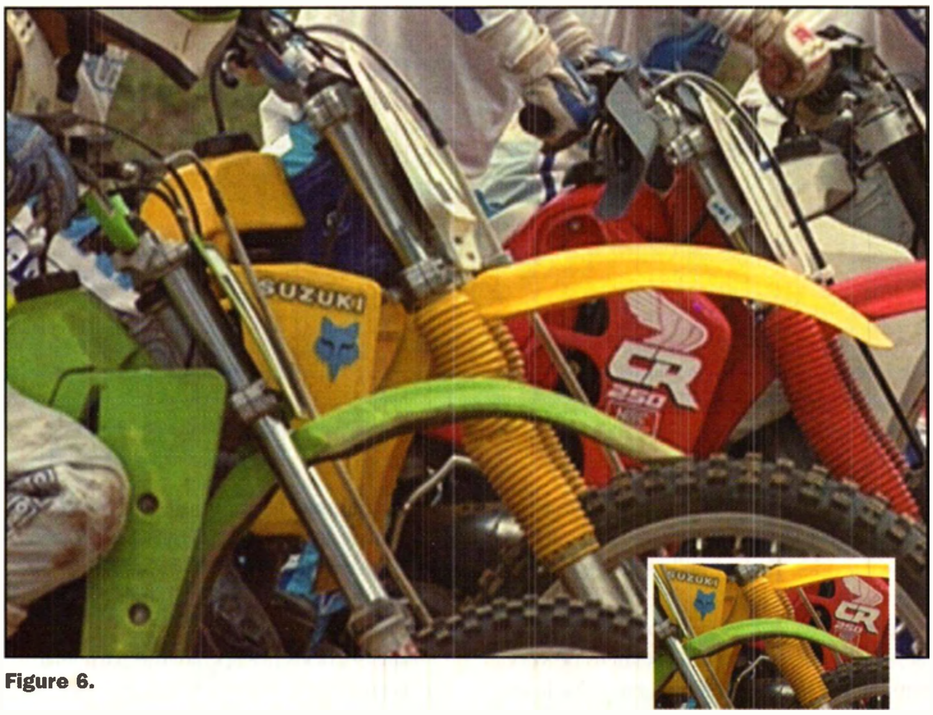


Figure 6.

The key to inter-frame compression is that, in general, one frame is very similar to the previous frame which represents the state of play a fiftieth of a second earlier. And if very little has changed it's clearly inefficient to duplicate the data. Security recording equipment is an extreme case and one in which MPEG compression is highly efficient. Here a CCTV camera captures an image, which is recorded to tape, conventionally an analogue tape. But unless a burglar walks into the frame, nothing changes at all and every frame is virtually identical to the previous one. Yet an analogue tape will keep on running storing the information for that unchanging over and over again. Clearly if the initial frame can be stored and following this only the changes are recorded, a vast reduction in the overall amount of data can be achieved. But this is an extreme case – movies and TV programs aren't as static as this – so let's look in a bit more detail at how MPEG copes with more typical video footage.

To set the ball rolling a complete frame is encoded. In an ideal world, from this point onwards only changes need be stored but it's not hard to see why this doesn't work in practice. For a start, viewers need to be able to switch on the TV at any time or start watching a movie on DVD part way through. To allow this, complete frames have to be transmitted periodically. But whereas a complete frame every 100 frames may be adequate to allow this – since this would result, at the most, in a two second delay – they actually tend to be transmitted more frequently than this. This is to eliminate cumulative errors in the decoding process. Any dropped bit, for example, will carry forward until the next complete frame is received so these need to be comparatively frequent. In MPEG encoding, these complete frames are referred to as I-frames since they're compressed only by the intra-frame compression techniques.

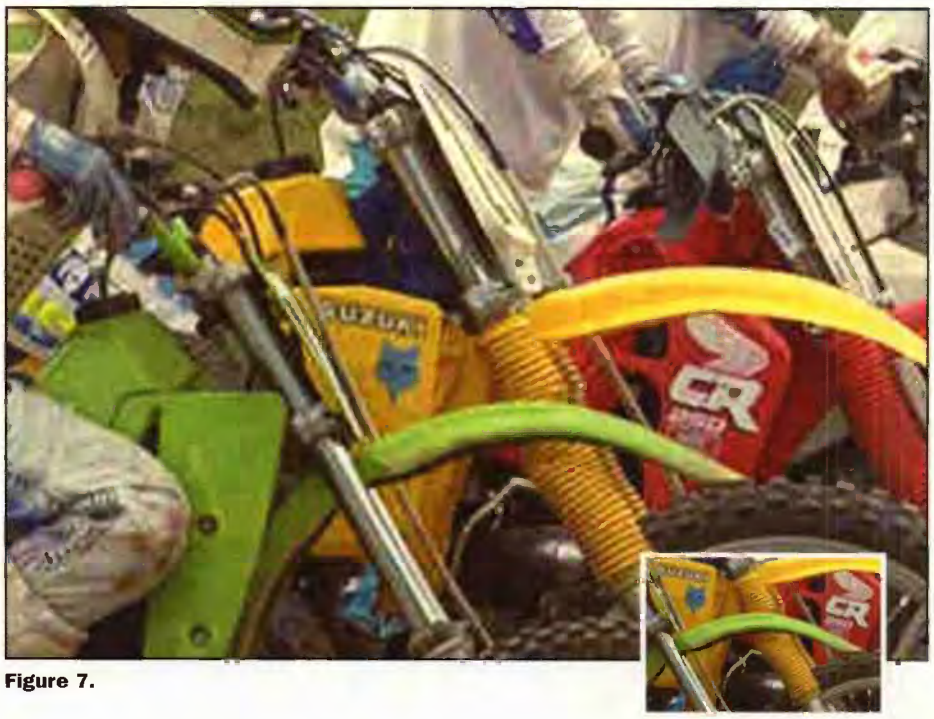


Figure 7.

The next type of frame is the P-frame and this stands for predictive frame – here's how it works. The frame is split up into 16x16 pixel blocks. Now the previous frame is scanned to find each of those blocks or, at least, something that appears to be very similar. Having found a matching block, its position in the frame is compared to its position in the previous frame to give something referred to as a motion vector. Typically the motion vectors will be zero or small but, of course, in the case of a fast moving object they can be significant. But transmitting motion vectors alone isn't adequate. You'll remember that I referred to finding a closest match to each 16x16 pixel block in the previous frame. The fact is that blocks don't just move between one frame and the next, they can also change slightly. To take this into account, each 16x16 pixel block in the current frame is subtracted from its counterpart in the previous frame to produce a block of error information. This – which is often strings of zeros and thereby easily compressed – in addition to the motion vectors adequately describes the differences between one frame and the next and is the data contained in a P-frame.

The third type of frame is the B-frame or bi-directional predictive frame. This works in a similar way to the P-frame but it looks, not only at the closest previous I-frame or P-frame, but also at the closest future I-frame or P-frame. Since most blocks will be found in one or the other of these two reference frames, the amount of data in the error signal will be smaller than in a P-frame and, therefore, a higher degree of compression can be achieved. There is a disadvantage in using B-frames, though, specifically that these frames cannot be interpreted until a subsequent frame is received. This can result in a delay that would not be acceptable in some real-time applications such as video-telephony and video-conferencing. These delays are reduced by placing a limit on the number of B-frames that can be interspersed between I and P frames.

MPEG-2 does not specify the particular sequence of I, P and B frames. This can be chosen by the user to give a level of performance, which is appropriate for the application. Specifically a balance has to be struck between the amount of compression, of random access, and of the encoding delay. Using I-frames only (I I I I I I I I I I I ...), for example, will result in total random access and no delays but the compression won't be very good. If we add P frames only (e.g. I P P P P I P P P P P I ...) the compression ratio is improved, there is still no encoding delay but a lower degree of random access is available. And by adding B-frames (e.g. I B B P B B I B B P B B I ...) the compression ratio improves yet again but an encoding delay is introduced. An interesting feature of MPEG compression, and of many other schemes for that matter, is that the compression ratio depends on the data being compressed. For example, an action movie will compress much less well than a chat show since there will be more motion and hence the motion vectors and error information will be greater. If the data is being transmitted over a fixed bandwidth channel – and this channel has been designed to cope with the average bit rate – this will give problems at times. A means has to be found, therefore, of further reducing the bit rate at times when lots of action would otherwise cause frames to be lost. This is achieved in the intra-frame compression stage that follows the inter-frame compression. The quantisation stage of the DCT compression is controlled so that the output buffer will never overflow. In particularly bad cases, this can result in a very pixelated image for a fraction of a second and this is an artefact that many digital TV viewers will recognise. Figure 9 is an overall block diagram of MPEG compression and Figure 10 sheds a bit more light on the way images are reconstituted from a stream of I-, P- and B-frames.

Wavelet Compression

The up-and-coming compression technique which everyone is now talking about is wavelet compression, as employed in the soon-to-be-released JPEG 2000 standard for the compression of still images. This is quite a big topic and the subject of wavelets and their application to data compression would deserve an article in itself. Nevertheless, no article on data compression could be complete without

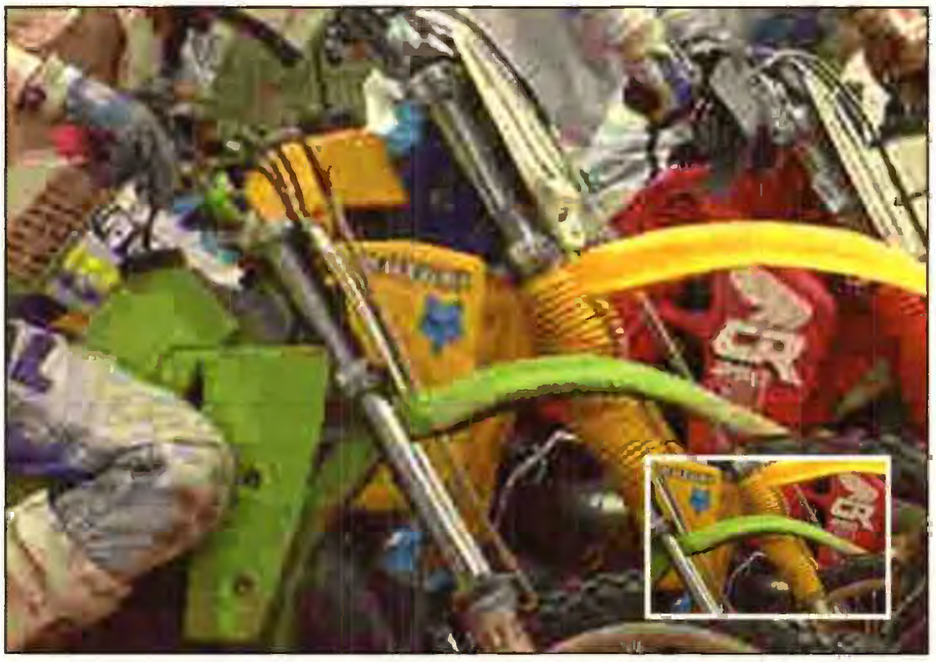


Figure 8.

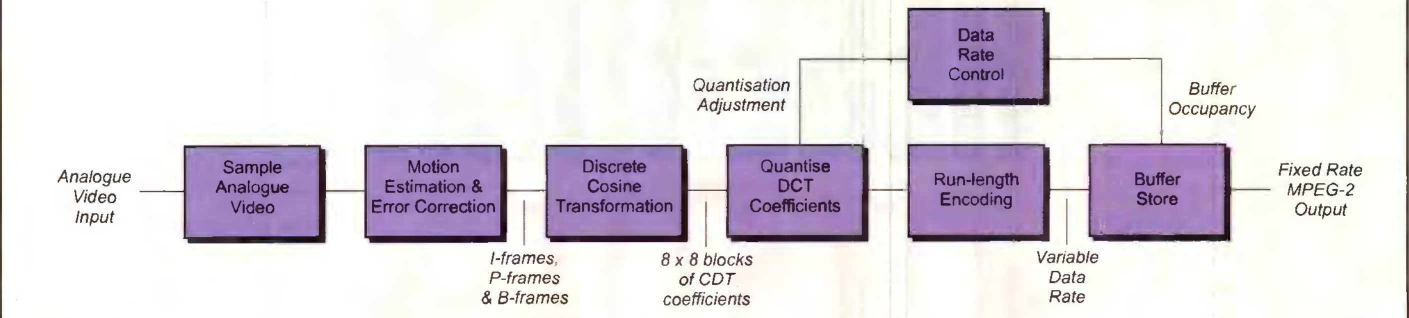


Figure 9.

at least introducing the subject.

We've already looked at the Fourier Transformation and the closely related Discrete Cosine Transformation which is key to many of today's image compression standards. In both cases, a signal is converted from the spatial domain (or in the case of an audio signal the time domain) into the frequency domain. In other words, the input signal ends up being represented as a series of sine waves of different frequencies. Wavelet analysis is like Fourier or DCT analysis in that the end result is the sum of multiple waveforms. However, instead of an infinite pure sine or cosine waves, the waveform which is summed is irregular in shape and is localised. In fact a number of so-called wavelets have been used but one of the most successful ones, at least in the area of data compression, is called the Daubechies wavelet and is shown in Figure 11. You'll notice that it contains a mixture of frequency components and is localised. As such, it is better suited than sine waves for representing many real world waveforms.

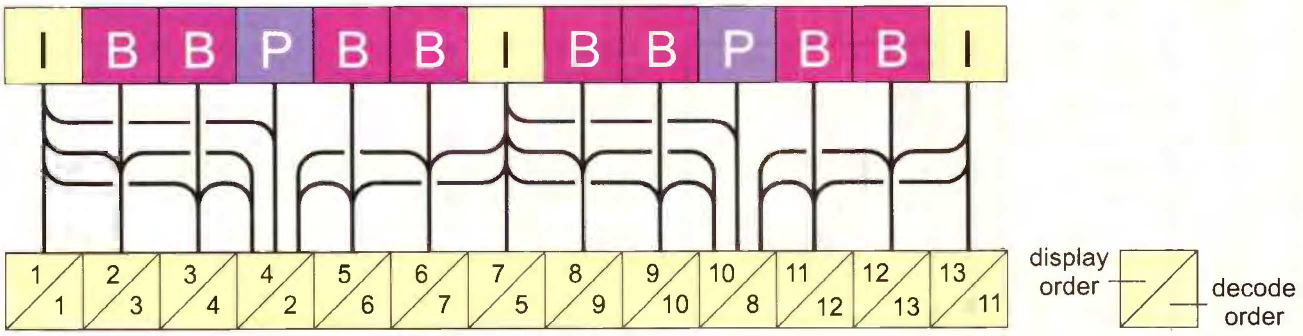


Figure 10.

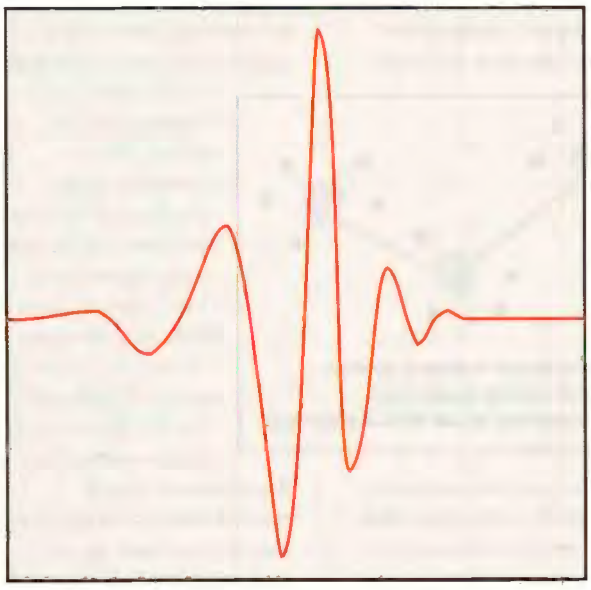


Figure 11.

To cut a long story short, an image expressed as a series of wavelets of differing amplitudes, frequencies and translations can be compressed in much the same way that an image consisting in a series of sine waves can be compressed. Furthermore, it's generally true that a higher degree of compression can be achieved at the expense of less visual degradation. To wind up this brief look at wavelet compression I'll provide one or two facts about JPEG 2000. This will illustrate the advantages of wavelet compression compared to the DCT compression employed by the initial JPEG standard. Its designers claim that JPEG 2000 will "avoid some of the more unpleasant JPEG DCT artefacts – the ringing near sharp edges, the clear tile boundaries, and the harsh colour quantisation" which tend to occur with DCT compression at high compression ratios. Furthermore it is their aim and expectation that the new standard will offer "fair" quality image reproduction at rates down to 0.1 bits per pixel and below. It has also been suggested that a JPEG 2000 file compressed at 200:1 will offer superior quality than a standard JPEG image compressed at 5:1.

The Squeeze is On

But this talk of the advantages of Wavelet compression compared to DCT isn't just of pure academic interest. With BT dragging its feet over the roll-out of ADSL, and with indications that prices will be significantly higher than most people will be prepared to pay, many Web users may be stuck with 56k modems over POTS (the Plain Old Telephone Service) for some time yet. But with new compression standards promising more for less, the lack of wide-band Internet access means that we might not have to continue to endure the World Wide Wait for much longer. And remember, this is just one application of data compression. Digital TV, digital radio, DVD, mobile phones all rely on data compression for their very existence. Like it or not, the squeeze is on and life in the 21st century just won't be the same without it