

Video Compression 101

By Richard Jackson

Video is Big. Not only as in, “It’s the happening media - gotta have it!”, but also, “I just tried to record last Sunday’s sermon on my computer and now my hard drive is full!” When turned into digital media a standard definition video signal requires a little over 30 MegaBytes (MB) of disk space per second – or a whopping 105 GigaBytes (GB) per hour (see sidebar 1). The sheer size of digital video files may be a disk drive manufacturer’s best friend, but for users who want to record, edit, and distribute digital media it can be an equally large headache.

Fortunately, there are techniques for reducing the size of digital video files called video compression. If you own a digital video camcorder you’ve already used video compression to record that last vacation or your child’s Little League game. In this [series of?] article we’ll look at the technology behind video compression as well as some common compression techniques and their tradeoffs.

What Is Video Compression?

Simply put, video compression is reducing the size (in bytes) of a digital video signal. If we’re talking about a file (i.e. recorded video), video compression will reduce the size of the file on your disk drive. This not only means more files (or minutes of video) will fit on your hard drive, but it also means that your drive does not have to be a high performance model to play back the video file without stuttering or “dropping frames” (if you want to play back a 30 Mbytes/sec video file in real time, then your hard drive will have to be able to read data at least this fast in order to keep up).

If we’re talking about a video “stream” - for example the broadcast channels you’re receiving over the air (or cable, or satellite) - a compressed video signal will take less bandwidth (think “number of channels”) than an uncompressed signal. This means that more video channels can fit into the same broadcast space, which means more room for home shopping channels. (Hey, every technology has its dark side!) In the case of streaming web video, lower bandwidth means viewers with lower-speed Internet connections – and/or computers - will be able to view your video.

Before a compressed video file or stream can be viewed, it has to be decompressed back to its original form. In the case of compressed video files, this is done by a program designed for that purpose (possibly the same program that did the original compression). In the case of broadcast video streams, the decompression is done in your television receiver or set-top box just before it is displayed on the screen. In the case of streaming web video, the decompressor is often built into the “player” software, or may be available as a plug-in for web browsers. A software program or hardware component that does compression and decompression is called a codec. (Note: even though the abbreviation “codec” implies both compression and decompression, most people also use the term to describe software that only does one or the other).

You’re probably already familiar with data file compression. Programs such as WinZip or StuffIt take a computer file and make a compressed file (e.g. .zip or .sit) that contains all of the essential information of the original file but in a smaller size. These programs use some of the same compression techniques as video compression – in fact, you can use them to compress video files and you will get a smaller file. However, because these compressors

are designed to be used with any kind of computer files, they must use lossless compression techniques. In other words, the decompressor program has got to be able to recreate every byte of the original file without any loss of data.

For most video applications we're willing to throw away some picture information in order to get more compression. This is called lossy compression. When a lossy file or stream is decompressed we'll get back something that looks like our original picture, but it may not be an exact replica. As we'll see below, lossy compression schemes will usually give us smaller files (more compression), and if we're careful the resulting picture won't look too bad.

How Does Compression Work?

The trick to video compression is to get rid of stuff no one will notice is missing (or if they do notice – hopefully it's not something terribly important). Like packing for a trip, your goal is to make your suitcase (the compressed file) smaller and lighter but still contain all of the important items you think you'll need. There are a lot of techniques for removing “unnecessary” video information – which is why there are lots of different compression standards. We'll discuss some of the basic tricks that codecs use so you can get an idea of some of the tradeoffs involved.

Department of Redundancy Department

It sounds obvious, but if there is redundant data in a file, it means that the data can be removed without impacting the rest of the file. For example, if we assume that in English the letter ‘Q’ is always followed by the letter ‘U’, then we could compress a text file by looking for all of the ‘Q’ letters and removing the following letter. We could then decompress the file by again looking for the Qs and re-inserting a U after them. This trick can be used by lossless compression techniques, since we can restore the original data when we decompress. Of course this technique is only as good as our assumptions: if there is an exception (“Iraq”) then the algorithm will break, or we'll have to make it more complicated to handle the exceptions.

Variable Length Coding

The actual data values that make up each picture element (“pixel”) are called codes. In our original video signal we said that we used 3 bytes (one each for red, blue, and green information) for each pixel – regardless of what color the pixel was. But what if some colors happen more often than others? We could invent a coding system that maybe used only one byte for popular colors, and used more bytes for unlikely colors. If there were more “popular” pixels with one byte codes than “oddball” pixels with four or more byte codes, then the total number of bytes in the file would be less.

For those who remember the telegraph days, this is exactly what Samuel Morse did when he invented the “Morse Code”. He created patterns (codes) of short “dots” and longer “dashes” for each letter of the alphabet. The most commonly used letters got shorter codes (“E” is a single dot), while less-used letters got longer codes (“Q” is 3 dashes and one dot – which takes 13 times longer to transmit than a single “E”). The result is a lossless coding system that requires less time to transmit a telegram – as long as the telegram follows the same standard English usage rules Mr. Morse had in mind. (Obviously a telegram

consisting of all Qs would break the code and probably take longer to transmit than if it had never been coded at all! We'll see that almost every compression technique can be "broken" by giving it data or video that is different than what it was designed for.)

Coding techniques are usually lossless. Popular coding techniques include Huffman Coding and Arithmetic Coding.

Run Length Coding

If you took a picture that was one solid color, it would be a lot more efficient to describe it by saying, "the whole picture is red" than to send an entire picture's worth of the same pixel values. We don't often run into entire video frames that are only one color, but in certain kinds of video (notably graphics and animations) we often do have large areas that are the same. Run Length Coding takes advantage of long "runs" of the same pixel value by sending the pixel value and the length of the run (for example, "the next pixel has a Red/Green/Blue (RGB) value of 75/23/90, and the next 100 pixels are exactly like it."). This compression technique is lossless, but it can be broken by pictures that don't have long runs of constant pixels.

Differential Coding

In a typical picture, pixels don't change value much from one pixel to the next. So if you look at the RGB values of one pixel in the middle of a blue sky, the chances are pretty good that the adjacent pixels will be very close in value. Differential Coding takes advantage of this by sending the full value of the first pixel, but after that only sending the "difference" from one pixel to the next. For example, "the next pixel is just like this one, but with 2 steps more Blue and 3 steps less Red". If our premise is correct ("pixels don't change much"), then most of the time we will be sending very small "difference" values. By combining this with a clever coding system (see above) which uses small codes for small difference numbers and larger codes for large differences (that are less likely), this technique can be used in lossless compression.

Vector Coding

Sometimes certain data patterns will happen more often than others. In English we have a way of "compressing" certain common words by abbreviating them. So if I write a note that says, "I will be going to the store on Fri.", I have saved some ink and paper by not spelling out "Friday". (Of course, the decompressor has to assume we didn't mean to say, "I will be going to the store on Frieda's bicycle").

Vector coding works by looking for common patterns of pixels and substituting a smaller code. To take a silly example, if all cars looked alike (i.e. the pixels describing every car were always identical) we could insert a short message saying, "draw a car here", instead of sending all of the car's pixels over and over.

Quantization

When a video signal is digitized, the Red, Green and Blue signal levels are sampled at regular time intervals and turned into digital numbers. The precision of each pixel is the accuracy of the sample. For example, we could say that the Red value of a pixel is 4, or 5, or 6, ... If we wanted more precision we would keep more accuracy, for example: 4.3, 4.4,

4.5, ... Or even more precision: 4.31, 4.32, 4.33, ... Precision is good but it comes at a price: each sample has to contain more data bits to save the extra precision.

Quantization is the reverse: we reduce the number of bits in each sample by throwing away precision. For example, we've been assuming that our original uncompressed video has three 8-bit bytes per pixel. This allows us to specify the Red, Green and Blue values as numbers that range from 0 to 255 (see sidebar 3). If we quantized these RGB values to only use 5 bits each, we could fit each pixel into 2 bytes (3 samples * 5 bits/sample = 15 bits, with one bit left over). This would reduce the size of the pixel (and the whole file) by 1/3. However, now each RGB value would only have 32 possible levels instead of the original 256 levels. This loss of precision shows up as "stair-stepping" in areas of the picture where there are smooth changes of color. Because there is a loss of picture precision that can't be made up by the decompressor, quantizing is a lossy compression technique.

Blocking and Reordering

A video frame is normally scanned – and transmitted – from left to right and top to bottom, i.e. from the upper-left corner of the picture to the lower right. We've talked about some compression "tricks" that take advantage of neighboring pixel values (see Run Length Encoding and Differential Coding above). Sometimes we'll get better results if we change the scanning order of the pixels to put similar pixels adjacent to each other.

Some compression algorithms break pictures into small "blocks" of pixels (typically 8 pixels wide and 8 pixels high) and analyze the Red, Green and Blue pixel values in this small region. Since close neighboring pixels are more likely to be similar in value, this technique helps with other algorithms, such as Run Length Encoding or Differential Coding.

Scaling (Size Change)

This is the "microfiche" technique: shrink the picture by making it smaller in size (i.e. fewer pixels), which will also be smaller in bytes. At the decompressor end, enlarge the picture to its original size (or, if the ultimate destination is a small web browser window, don't bother to re-enlarge it!). Just like when you lose quality when you shrink and re-enlarge a picture in a photocopier, you will also lose quality when you do this as a compression technique.

Transforms

Many compression techniques rely on transforming a picture's pixels into a new form that is easier to compress. One example is a color-space transform that converts a set of Red, Blue and Green pixel values into luminance and color difference signals (often abbreviated Y, U, and V). Because human eyes are more sensitive to luminance (or black and white) detail than to color detail, we can throw away more of the detail data in the color difference channels without being too noticeable.

A more complex form of transform is the Discrete Cosine Transform, or DCT. This is a mathematical process that changes a picture from a bunch of Red, Green, and Blue pixel levels to a matching set of Red, Green, and Blue frequencies (see sidebar 4). By applying

some of the previous compression techniques to pixel frequency values instead of level values, we can get better compression results.

Transforms are seldom used by themselves. However, applying some of the other compression techniques to transformed data will often give better compression results than when the same techniques are applied to the original picture data. To decompress the signal, it needs to be “reverse transformed” by the codec to recover the original pixel values.

Temporal Compression

Video signals consist of a series of still pictures (“frames”) in rapid succession. In the case of NTSC video there are 30 frames per second – a rate high enough to fool a viewer’s eye into thinking there is smooth motion. Some of the techniques outlined above can be used to not only compare neighboring pixels to the left, right, top, and bottom, but also neighboring temporal pixels (i.e. the same pixel position from one frame to the next or previous frames). This is a particularly fruitful area for video compression, since most video does not have a lot of change from one frame to the next.

Imagine a video where a car is driving across the scene in front of a stationary background. In the areas of the picture not covered by the car, the background is the same from frame-to-frame and we can use a differential coding technique to describe one frame in terms of its difference from the previous frame (“the next frame is just like this one”). The only “new” pixels we need to describe for the next frame are where the car has moved.

Motion Compensation

This is another temporal compression technique, which takes advantage of the way objects typically move from one frame to the next. In the Temporal Compression technique, our example had a moving car in front of a stationary background. We had to send the pixels for the car each frame, since it was always moving relative to the previous frame. In Motion Compensation, we can take into account objects that have already been described in a previous frame (e.g. the car), but have moved in the new frame. By saying “this group of pixels is just like the previous frame – but shifted over 100 pixels” then we can save the bytes required to describe the car’s pixels for each frame.

Constant / Variable Bit Rate

The bit rate is a measure of compression per unit time. As you might guess from the descriptions of the above compression techniques, some work really well (i.e. compress a lot) for certain kinds of video, but work less well for other kinds of video. This means that the amount of compressed data can change over time, as the video changes from easy to compress segments to harder ones. If we do nothing, the bit rate will vary from shot to shot, which is called variable bit rate. This is fine if you are saving the compressed video in a file for playback on your own computer. But if you are planning to make this file available for video streaming, it can be a problem for the receiver to download a transmission over the network that seems to “slow down and speed up” throughout the broadcast. For these applications it is better to have a constant bit rate, where there is a relatively constant amount of data regardless of what the original video looked like.

Since video compression quality is usually related to the bit rate, it turns out that a variable bit rate is usually a constant quality level, and a constant bit rate requires a variable quality level. This is because the only way to modulate the bit rate (i.e. hold it constant with changing video) is to increase and decrease the amount of compression – which affects the quality.

Video Compression Standards

While each of the above techniques can be used by themselves to compress a video file or stream, typically they are used together to achieve even more compression. Here are some popular compression standards and the combination of techniques they use.

JPEG

The JPEG (Joint Photographic Experts Group) compression standard was designed for still pictures and is a combination of Transform (DCT), Quantization, and Coding techniques. While there is a provision in the standard for lossless JPEG compression, most people use it in one of the lossy modes. To use JPEG with video, simply take each individual video frame and treat it as if it was a single still picture. When JPEG is used this way to compress a video stream it is usually called “Motion JPEG” or “M-JPEG” (note: this is NOT the same as MPEG – see below). JPEG is lossy and variable bit rate (although the amount of compression can be varied to keep the bit rate nearly constant).

MPEG

The MPEG (Motion Picture Experts Group) set of standards was designed for video and adds temporal compression and motion compensation to the techniques used in JPEG. There have been three major “releases” from the MPEG standards body, in chronological order: MPEG-1, MPEG-2, and MPEG-4. As you might expect, time has given us more technology and tricks to play with, so MPEG-4 is much more “rich” and complex than MPEG-1. The official MPEG-1 was limited to a maximum frame size of 320x240 pixels, and has been mainly used for CD-ROM and Web multimedia. MPEG-2 was designed for broadcast video, and is the basis of most broadcast, cable, and satellite transmission. It is also the standard video compression for mastering DVDs. MPEG-4 has capitalized on the experience gained from its older siblings, and offers improved compression at the cost of much more complex algorithms. MPEG is lossy and variable bit rate (although the amount of compression can be varied to keep the bit rate nearly constant).

DV (DV25, DV50)

The Digital Video (DV) compression standard was invented by Sony and Panasonic (check this...!) for use in consumer camcorders. It has become very popular because the camcorders also include computer ports (iLink, or “FireWire”) that allow users to directly copy the compressed files from the camcorders into their computers. DV is similar to M-JPEG in design. DV is lossy, but it has been made constant bit rate to meet the needs of recording to tape. It is limited to standard definition (NTSC or PAL) video.

HDV

HDV is a new high definition camcorder video compression, which is trying to duplicate the success of DV. However, despite the similarities in name, HDV actually uses MPEG compression at a constant bit rate.

Choosing a Video Compression

There is a bewildering array of video compression choices – and it seems like new codecs are being introduced every day. Why doesn't everyone use the same type of compression, and how should you choose which one to use?

One reason there are so many choices is that different codecs are designed for different purposes, so the “best” choice depends on why you're using video compression. Another reason is that technology continues to march forward: “old” codecs (in this business, “old” can mean 6 months) continue to live on because they're needed for legacy media (as long as multimedia CD-ROMs live on, we'll still need MPEG-1 codecs to decompress them!).

But faster computers and new algorithms mean that we continue to get more sophisticated codecs capable of squeezing just a little more data (and/or keeping better quality) than their predecessors. A sophisticated standard like MPEG-2 was unthinkable in the days of MHz-speed i386 PCs, but today's GHz Pentiums can chug through the computations without batting an “i”.

Here are some questions to ask when considering different compression standards:

1) What is the application? Some codecs have been specifically written for video streaming: they are very computer-intensive on compression (which only has to be done once), but are relatively easy to decompress (which is done by lots of viewers). Other codecs are designed for video editing applications: these will emphasize quality over compression, and work well in real-time. In some cases you'll find the decision has been made for you: distribution by DVD requires MPEG-2 compression; a web site might have already standardized on a single format like Real; or a video editing application might only support a few codecs. In other cases you may be starting off with already compressed video, for example from a DV camcorder. While the footage could be transcoded to a different compression, this will take time and could reduce the video quality.

2) Who will be watching your compressed video? If you're only compressing video for yourself (i.e. to use in a video editing session, or to free up some hard disk space), then only you need a copy of the decompressor. But if you're going to distribute the compressed video to others, make sure that the decompressor (software or hardware) is available to them as well. Some decompression software costs money, or may have only been written for certain kinds or speeds of computers. You may be able to run the latest compression algorithm on your faster-than-light Unobtainium 2000, but if you're planning on sending copies to Grandma – who is still running the 8 yr-old PC that someone gave her to read e-mail – you might want to use a codec that is less demanding.

3) Which is more important: compressed size or video quality? Most video compression algorithms are lossy, and the more you compress the video the more the quality will suffer. If you are compressing for archival purposes you may wish to err on the side of quality: you'll be unhappy if you come back to get your video later and find that it isn't good enough to use. On the other hand, if you're compressing a video file for a web streaming application then you'll want to make it as small as possible so that it will be available to the widest possible viewing audience (some multimedia players allow for multiple files of differing compression sizes and qualities: the player then chooses the best file to match the viewer's connection speed).

Conclusion

Because there are so many applications for video compression, there is no one codec that can be said to be “best” for all. Each compression system involves tradeoffs between the amount of compression, the speed or processing power it takes to do the compression and decompression, and the resulting video quality. And new codecs are becoming available as technology advances and speeds up. I hope this article has given you some information on how compression works, which will give you the tools to evaluate the best codec(s) for your application(s)!

Sidebar 1:

Where did those numbers come from? A standard video “frame” (a single still picture) is 720 pixels wide and 486 pixels high. Each pixel consists of 8 bits (1 byte) each of Red, Green, and Blue (RGB) information.

$3 \text{ bytes/pixel} * 720 \text{ pixels/line} * 486 \text{ lines/frame} * 30 \text{ frames/sec} * 3600 \text{ sec/hr} = 113,374,080,000 \text{ bytes/sec.}$

In school you learned that metric prefixes (Kilo, Mega, Giga, etc.) are built on powers of 10: a “KiloSomething” is one thousand Somethings, a “MegaSomething” is one million Somethings, and a “GigaSomething” (besides being the next 7-Eleven Slurpee® size) is one billion Somethings.

However, computer engineers come from a different planet where items are measured in powers of 2: so a “KiloByte” is 210, or 1024 Bytes. A “MegaByte” is 220, or 1024 * 1024, or 1,048,576 Bytes. A “GigaByte” is 230, or 1024 * 1024 * 1024, or 1,073,741,824 Bytes.

Sidebar 2:

Astute video savants will recognize that these numbers are for “NTSC” video, used primarily in North America and Japan. Most of the rest of the world uses a television standard called “PAL”, which is 720 pixels wide, 576 lines high, and 25 frames per second. Compared to NTSC, each frame is “taller” – but there are fewer frames per second. so the total size is very similar.

Sidebar 3:

Computer engineers specify information in terms of “bits”. A single bit is like a light switch: it can be in one of two states, “Off” and “On”. If we use two bits (two light switches) together, we can express four states: both switches Off, the first switch On and the second switch Off, the first switch Off and the second switch On, and both switches On. Three bits can express eight states, four bits can express 16 states, etc.

[This is why Paul Revere needed two lamps (“one if by land, two if by sea...”). He was anticipating three possible conditions: no attack (no lamps lit), attack by land (one lamp lit), and attack by sea (2 lamps lit). One lamp by itself could only cover two of the states.]

Sidebar 4:

We’re used to thinking about audio signals having frequencies (“pitch”), but “picture frequencies” are strange to most people. Think of a video frequency as a measure of how fast pixel values are changing from pixel-to-pixel: if the pixel values are changing slowly in an area of the picture, that portion of the picture mostly has “low frequencies”. If there is a lot of picture detail in an area so that the pixel values are changing quickly, then you can say that it has more “high frequencies”.