13 File Formats

Many VTR formats evolved as the various manufacturers came up with multiple ways to lay data down on tape. At one point, it was hoped that all the players would come together and settle on a common format, as the interchange between users of different formats involved dubbing, or making a copy of the media from one VTR or video server, to another with a different format. Each copy resulted in media quality getting a little bit worse.

The beginning of the use of digital to convey video and audio information caused the explosion of the frequency bandwidth required to convey that information. A way to minimize the frequency spectrum used became paramount.

Enter compression. There are two types: lossless and lossey. Lossless uses techniques to minimize the amount of bits required to convey information without losing any of the content. When the data is uncompressed, it is an exact copy of the original. Lossey means just that, some data is thrown away with the hope that it won't be missed.

Video Compression

Video and audio compression use both lossless and lossey methods, but it is the lossey part of the process that carries by far the most freight. Almost all video compression methods today use the same upfront process for eliminating data that hopefully won't have an adverse effect on the end result. Digital VTRs use the same process, as do over the air digital television, satellite and microwave transmissions, as well as the local cable company and DVD players.

The first step in going digital is to take a continuous video and audio signal and chop it up into samples. We then take each sample and convert it to a number. Now in many instances, we are done. The only problem is that whereas analog video would fit in a bandwidth of just over 6 MHz, the digital video equivalent would be over ½ GHz, about a 100 times increase. Why? It's all the ones and zeros that are needed to describe each sample and all the sharp edges that digital bits need to have so that equipment at the other end of the line can decipher them.

Chroma Sub-Sampling

As we have discussed earlier in the camera and Digital 101 sections the first way to eliminate extra data is something that's been going on since color television first hit the scene. Our eyes do not see color, or chroma, detail as well as they do black and white, so we can eliminate chroma samples. Remember in component video, we usually generate a luminance and two color difference signals. These orthogonal signals represent a vector representation with the phase being tint information and the amplitude being saturation. The 2 chroma values represent a complex number which is a value that not only has a magnitude but a direction also. If we sample the separated chroma values at half the rate of the luminance, we have 4:2:2 sampled video. This is a way of reducing the raw bit rate by 33%. Here the chroma sub-sampling is in the H direction only. Both chroma difference signals are co-sited with a Y sample.

The data samples are organized as follows:

Cr/Y/Cb, Y, Cr/Y/Cb, Y, etc.

We can continue to reduce the bit rate by sub-sampling the color information even more. 4:2:0 is chroma sub sampled in both the H, and the V directions. There are 2 ways to do this. In the first way, even lines are sampled 4:2:2, while odd lines are sampled 4:0:0 (no chroma samples). The second way has the chroma sampled in a quincunx pattern. What this means is that on odd lines you throw the even chroma samples away and on the even lines you throw the odd chroma samples away. This makes an interwoven 4:1:1 sample rate. 4:2:0 chroma sub-sampling can reduce raw data to be processed by the encoder by 25%.

Chroma sub-sampling is a good start, but not nearly enough. So the digital number explosion still has a ways to go to be knocked back down to a reasonable size. MPEG compression works because of limitations of our eyes and brain. Our eyesight allows larger visual errors to occur unnoticed at high luminance levels. When observing an MPEG compressed video stream these errors become apparent if the display's brightness is turned down. Coding errors are also less visible around sharp edges or other high frequency areas. In MPEG streams, errors that occur in areas with fast motion are also less detectable.

Discrete Cosine Transform

So at this point, we need to transform spatial video into a more efficient realm. We can do this as a single horizontal line of digital values at a time, or better yet, in both the horizontal and vertical direction at the same time, as in, 2D instead of 1D. It would be too hard if we tried to do a complete video frame at once, so it was decided to split the complete video frame into small square blocks instead.

The size of those blocks ended up being 8x8 pixels; eight horizontal samples by eight vertical samples. The method decided upon was to wring some average values out of that block and see patterns that represented what was going on horizontally, vertically, and diagonally for the whole block. These values gathered for an 8x8 block, representing 64 discrete samples, would be less than capturing each individual cell in the block. In the case of a wall painted gray over the entire area of the block it would be a single value, not 64 values all equal.

So how do we do this? The most common way is by using the algorithm below. We break that up piece by piece and see that it is actually a fairly straightforward process. But first let's explain what it does and then we'll see how it does it. The whole object is to re-organize all the information in our 8x8 blocks from spatial, into cells that describe a pattern of what is being represented. These patterns are based on sample frequencies that sample patterns across the horizontal, vertical, and diagonal bit patterns of the original spatial block.

The process that does this is called the Discrete Cosine Transform (DCT). Discrete simply means our 8 cell square block. Cosine… (Oh heavens… Trigonometry!) Don't panic, it sounds much more complicated than it is. As to Transform, remember we mentioned we're taking the 8x8 block where each cell represents a place in space, to where each cell represents a pattern. This is a poor man's version of a Fourier Transform, where we take something represented by time, or space, and convert it into the frequencies that comprise the function in time. The DCT does that, but more simply. Let's see exactly how.

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & for \quad u = 0 \\ \sqrt{\frac{2}{N}} & for \quad u \neq 0. \end{cases}$$

Let's break this up by starting with all the x and ys, and u and vs. The x and ys simply represent where in space, on the 8x8 array a cell is. 8x8 means there are 8 xs in a row and 8 rows (ys) in the array. 64 locations each represented by a combination of x and y. x =1 and y =1 means the top left most cell. X = 8 and y = 8 represents the bottom right most cell.

As to u and v, those represent locations in the new, transformed array, which represent patterns culled from the x, y array.

How are those patterns discerned? That's what the two sets of cosines (cos for short) do. One works on the horizontal axis, x and u, and the second on the vertical axis, y and v. Together both these functions also work in the diagonal direction.

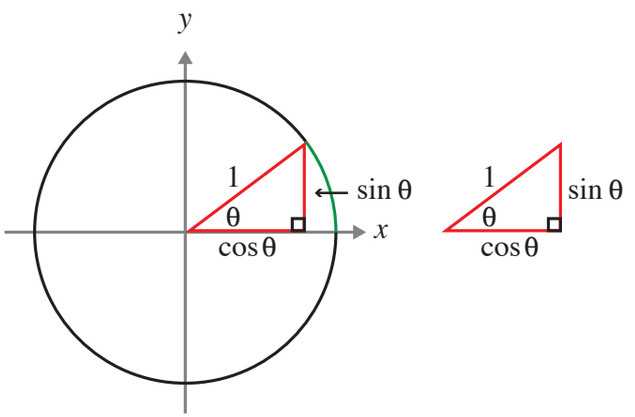
The summation symbols (Σ) mean add all the values collected in each column, starting at

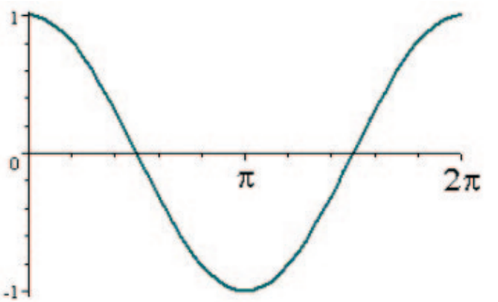
x = 0 and y = 0 (start at 0 and not 1) and ending at x = 0 and y = 7

then the next column x=1 and y = 0 to x = 1 and y = 7

Continue until all 8 columns have been summed.

The values inside the brackets [ ] after the cos represent the frequency of the cos function.

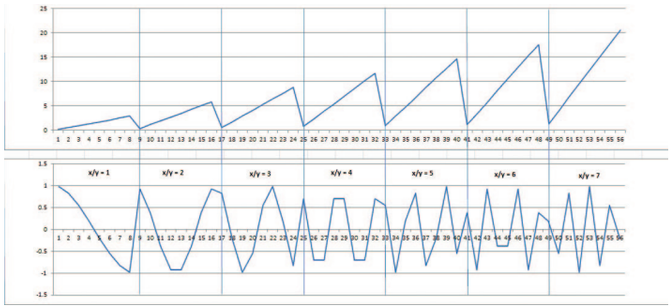




The cos function is used to represent the ratio of the triangle's adjacent side and the triangle's hypotenuse, or the long side emanating from the center of the circle. As the value of sin increases, the value of cos decreases. When sin = 0 cos = 1 (positive x axis); sin = 1, cos = 0 (positive y axis). As the two values sweep around the circle, the remaining values where one or the other is zero (when the hypotenuse crosses the x or y axis) are sin = 0, cos = -1 (negative x axis); and sin = -1, cos 0 (negative y axis). Each varies between 1 and -1, with cos 90 degrees ahead of sin as you sweep around the circle. If you plot what the cosine value looks like as it goes around the circle, you get the graph on the right.

The period, that is the length of time between the peaks of the cosine function, determine the frequency. As the period drops, it equates to the value between the brackets [ ] increasing the effective frequency.

So looking at the values between the brackets, you can see that the value will increase as x or y, and u and v increase. Thus 64 cycles are run through the 8x8 spatial square, and the summations from the various samples will be put into a single (u, v) cell each pass.



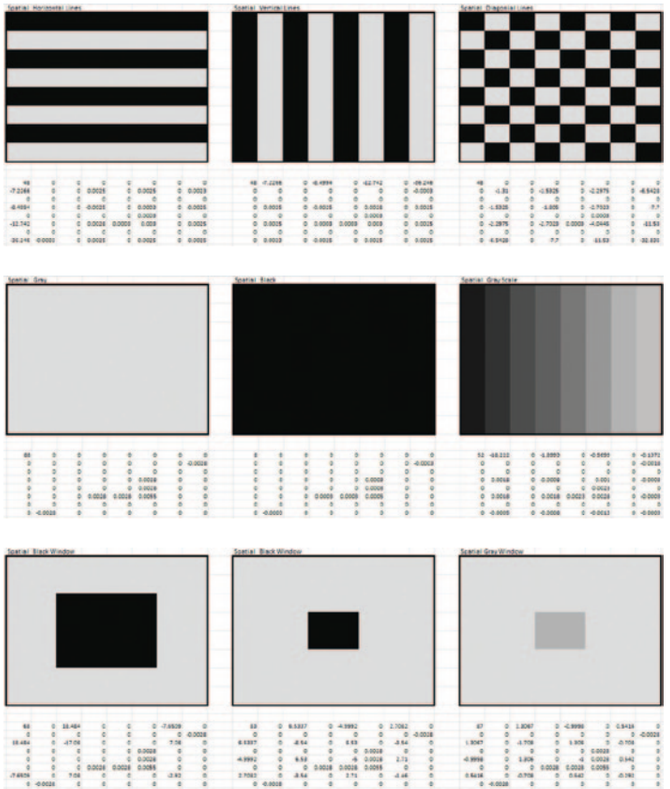
On the graphs shown above on the top one you can see that every step in x value left to right through the array, and subsequent y increase as a new row is reached increases the frequency value of the cosine function. The same happens with both u and v increases, first as you go from left to right (x or u) and then from top to bottom (y or v) in an ever increasing "sawtooth" pattern. This increasing frequency value results in a cosine like pattern with ever more sampling points as you move from left cells to right side cells as the bottom half of the graph depicts.

As can be seen in the bottom part of the graph the cosine period increases by ½ a cosine cycle as each new row is encountered. The second pass down the first column you can see a full cosine cycle for the length of the whole column. Each successive pass results in the frequency of the samples being ½ cycle greater than the previous column sample.

In this way, a single value is inserted into the top cell for the ½ cycle summed sampled values. For the next v cell down the summed value for the entire column taken at 1 complete cosine cycle is collected. Next v cell down and a 1 ½ cosine cycle pattern is taken and stored.

The first vertical column is done, then the next one to the right and so forth until the whole x, y spatial array has been sampled, summed and mapped into the u, v array.

Below are some common patterns that might be seen in these 8x8 spatial arrays and the number values that would end up in the u, v arrays below. Notice that the number in the upper left corner of the u, v array represents the average value of the whole 8x8 array. In fact numbers along the left column and the top row generally represent DC values. The numbers outside those row and columns, especially the ones toward the bottom right corner represent higher frequencies.



As can be readily seen, in all cases there are more zeros than cells with values. But believe it or not, with all the precision numbers generated, the actual compression at this point is not that great. The next step is usually one of the secret sauce ingredients in many compression vendors' approaches to compression.

A coefficient value is added to these samples, to add a scalar to all the values in the u, v array. This scalar value will be used to divide the computed values. That will lower some numbers below a threshold, again selected by the vendor, where they will be thrown out. The threshold and scalar coefficients are based on the target final bitrate that is desired. How well one vendor balances this compared to another usually results in the quality one compressor or encoder as they are often called, offers compared to another.

Now that we have a set of blocks that are no longer in the time domain, we can actually start to compress the data. The DCT by itself is a loss-less transformation. In fact, in many cases the transformation expands the data. Eight-bit input pixel data might come out with 11 bits of frequency coefficient data. What happens now is that this frequency coefficient data undergoes quantization. Each location in the frequency coefficient matrix is given a denominator (divisor) value based on its position. This can be done simply by giving the upper left corner a low number (such as 3), and increasing the number by some fixed amount (say 2) as you go diagonally through the matrix down towards the lower right corner.

If you did that you would see that the lower right corner would have a denominator of 31. Early on, the JPEG committee actually came up with unique denominator values for each coefficient location in the matrix by trial and error. Each coefficient value is divided by its quantization denominator. This process makes many of the coefficients not located in the upper left corner of the frequency matrix become zero. Changing the value of these quantization values is one way to control the amount of compression that takes place. A starting quantization number over 20 in the upper left corner is about the limit for any kind of recognizable picture out of the process.

Next the 2 dimensional matrix will be converted to a serial string of the coefficients (or simply a string of numbers). But it will be put into a string in such a way that most of the zeros that have been created in the matrix will come out the string together. This is done by reading the matrix not by row, or column, but in a zigzag manner. Coefficient (1,0) is read then (1,1), (1,0), (2,0), (1,1), (1,2) and so forth up to (7,7). In this way the zeros that have been generated in the lower right corner of the matrix follow one another out as a group of zeros.

Loss-Less Compression

Here's our first chance at loss-less compression. Now a process called Run Length Encoding is done. The data stream coming from the zigzag process feeding out of the frequency coefficient matrix has a Run Length value tacked on it that indicates how many preceding zeros were in front of it. Only values that are not zero themselves are kept. The values that are zero are thrown away.

Next, the value is combined with a set of bits indicating how many bits the DCT coefficient has. The number of bits varies, since we will not store the absolute value of the coefficient, but rather the change in its amplitude from the previous value. The relative value between successive coefficients is usually small. Hence a type of Hoffman coding is used. Changes most likely to happen are given short numbers of bits, less likely values are given longer numbers of bits. This is similar to Morse code where E is a dit, and T is a dah, both short sequences because they are sent so often. Therefore, a bit count of 1 represents a change either of +/- 1. Bit counts of 2 represent -2 to -3, or 2 to 3. Bit counts of 3 represent changes of either -7 to -4, or 4 to 7, and so forth up to 10 bits representing the upper half of possible change -1023 to -512, or 512 to 1023. These large changes will seldom happen. Thus most words will have short numbers of bits.

The above process allows 2:1 compression with essentially no loss of information, essentially loss-less compression. 8:1 is generally regarded as high quality. At the other extreme, 22 to 1 compression yields what used to be referred to as VHS type quality. Some systems use adaptive compression. This means that higher compression rates are used on fields with low information content (such as scenes with large redundant backgrounds).

Next, these DCT blocks are organized as slices. A slice is all the blocks that comprise the width of the raster. For 720p a slice would be (1280/8) 160 blocks long. 1080 it would be (1920/8) 240 long. So 90 slices make a 720 frame and 135 a 1080 frame.

Temporal Compression

Now we are done doing compression in the spatial realm. We move into the temporal realm of compression. With MPEG type compression schemes, we make three types of frames. The process just described would generate what is called an I frame. Some formats only generate I frames. An I frame is the same as a JPEG frame. The I stands for Intraframe encoding. An I frame only contains compressed spatial information from itself. There is no temporal information from other frames. This frame stands on its own. It is often called an anchor frame.

There are 2 other types of MPEG frames that don't stand on their own; they rely on information from other frames. One of these is P (Predictive) Frames; they use Interframe encoding. Motion compensation vectors are generated from the previous I or P frames. What this means is that if blocks from previous frames can be found to be exactly the same, but that they have moved in their entirety to a new location, those blocks don't have to be resent, as a motion vector points to their new location. These vectors are encoded as part of the P frame data. Also, any errors left over after the motion vector's predictions undergo a DCT compression themselves. This means the error difference between the two frames is compressed. This data is also sent as part of the P frame data. This results in less code than an I frame.

The 3rd type of MPEG frame is the B (Bi-directional) frame, which also relies on Interframe coding. Here interpolation is used to estimate DCT coefficients from past and future I and P frames. There are generally more B frames than P frames, and few I frames. B frames are generated from the nearest preceding and succeeding I or P frames, while P frames are generated from only the nearest preceding I or P frame. I frames are the starting point from which P and B frames are estimated.

A measure of the accuracy of an MPEG sequence is therefore the ratio of total frames to I frames. Each slice header also indicates what type of frame (DCT type) is being sent. At the decoder or receiving end, B frames result in delay since they cannot be interpreted until a future I or P frame is received. This implies a requirement for buffer capacity at the source and destination. We will see shortly that this is solved at the encoder.

Every I frame begins a new group of pictures (GOP). GOPs can be of various lengths; fifteen is common. But GOPs as short as 2 are used; this would be only an I and a B frame, followed by another I frame.

An example of a GOP of 10 would be: I B B P B B P B B P

The B frames rely on the I and P frames on either side of them, while the P frames rely only on the previous I or P frames.

Now in the example above, the decoder will need the first P frame before it can decode either of the first 2 B frames. So the GOP shown above will not be sent in the order shown above.

The transmit order would actually be: I P B B P B B P B B

In this way, it is the encoder that needs lots of memory and processing power and not every decoder. This is much more efficient, as decoders vastly outnumber encoders. The decoder knows in what order to display the out of order frames because each comes with a time stamp embedded in the stream.

Audio Compression

Let's pause here for a moment and look at how audio is compressed, as it is radically different from video. Audio compression, as used in Dolby E or AC-3, receives its coding efficiency by taking advantage of limitations in the human audio perception capability which is known as a Psycho Acoustic compression technique. Humans hear in the range of 20 Hz–20 KHz. Signals are inaudible if their sound pressure level is below the threshold of quiet, which varies by 60 dB over this frequency range. At 2 Hz the threshold is at a sound pressure level of 60 dB. At 60 Hz it is near 35 dB. It is lowest at approximately 1.5 KHz (near zero). At 5 KHz it is about 10 dB, at 10 KHz it is 20 dB, and at 20 KHz it is at 40 KHz. Spectral masking causes louder tones to mask the presence of nearby, softer tones. Temporal masking causes louder sounds to mask the presence of softer

sounds immediately before, or after the occurrence of the louder sound.

AC-3 breaks the audio frequency spectrum into small bands. It then counts on "masking" to reduce the audio data rate. Low level sound frequencies near frequencies with high level sounds are eliminated. This greatly lowers the bit rate. Spectral masking causes louder tones to mask the presence of nearby, softer tones. Temporal masking causes louder sounds to mask the presence of softer sounds immediately before, or after the occurrence of the louder sound. This can lead to interesting effects if the bit rate is set too low. Often, the first to go is background noise. A sporting event with a sellout crowd might sound like a sparsely attended event. A five piece band might sound like it is a 3 piece band. A singer might no longer have any instrumental accompaniment.

Compression takes time to accomplish, even for audio. Dolby type encoders produce large audio delays of 70 to 160 ms. Typically this delay is referred to as latency, which appears as lip sync errors, or lip flap. A typical viewer can not see lip sync errors that are less that two frames, but once you spot it, it's difficult to ignore. Truck engineers have to be very conscious in their signal routing to make sure equivalent delay is occurring in both the audio and video processing paths to insure proper audio & video alignment is maintained.

Codecs

While there are many media file formats in use today, like the VTR formats that came before they are all geared towards a specific solution to serving a particular need. Almost all codecs are based on DCT to turn baseband video into compressed data. The files produced today most likely end up as a file stored on a hard drive. Once created, they are either streamed, moved as media that could be played on a device if desired, or simply copied as any computer data file from one storage device to another, such as one replay device to another.

Codecs fall into three distinct areas; acquisition, immediate, and presentation. Often, because of the lower quality accepted for new gathering, acquisition and presentation use the same codecs. High quality content will generally be encoded by an intermediate codec.

A common early codec was DV. It was originally intended only for interlaced video. It also uses 4:1:1 sub-sampling, which made it a poor choice if any chroma-keying was involved. Unlike slices used in MPEG2, DV codecs produced 80-byte Digital Interface Format Blocks (DIF) which were multiplexed into a 150-block sequence which became the DV video stream. Ten of these blocks made up a frame. When this was intended for tape, these DIF blocks were laid down on tape. Now when intended to end up on a hard drive these blocks are wrapped in what is called a container, additional bits intended to organize the file for playout. We will look at containers shortly. DV codecs don't compress the audio and leave it as PCM.

The DVCPro file format closely mirrors the DVCPro tape format we described in the Replay section. The major difference is that it is going to computer type storage and not onto tape, additional data that will wrap the media. The same is true with XDCAM files.

Intermediate codecs are intended for post-production editing rather than by the end-user. These codecs tend to work at higher bit rates to preserve quality. One reason for these higher bit rates is that many intermediate codecs only create I frames. Remember that these are complete compressed frames in themselves and do not rely on any other frames to describe them. This allows edit cut points to be on any frame, and not only on the border of GOPs. ProRes and Apple derived format is an intermediate codec, and has bit rates varying from 100 to 220 Mbps. Going along with its high quality is that it uses 4:2:2 chroma sub-sampling, has 10-bit sample depth, and handles SD, HD, 2K and 4K video.

Another intermediate codec is Avid's Digital Nonlinear Extensible High Definition (DNxHD) format which is an implementation of SMPTE's VC-3 standard. With the wide range of bit rates it can produce, it straddles both intermediate and presentation codec realms. It will produce bit depths of 8 or 10 bits and has bit rates ranging from 36 to 220 mbps using I frames only.

MPEG4 takes MPEG2 encoding to a new level. H.264 and MPEG4 are the same technically; they were initiated with separate standards and now have essentially merged into a single standard. MPEG 4 is used for Blu-Ray discs and by YouTube. MPEG4 compression much more intelligently breaks apart a picture based on objects it finds in that picture. It disassembles a complete 2-D picture into individual 2-D or 3-D objects and sprites. Sprites could be the floor and background, such as a wall, in the picture. Objects could be pieces of furniture or a chart that is in the scene. The "talent" in the picture would also be an object. MPEG4 also offers conventional audio/video streaming. It shares the same timing and clocks as MPEG2.

MPEG4 also generates two types of I frames; normal ones, and ones that are labeled as IDR (Instantaneous Decode Refresh) frames. Not all I frames are IDR frames. An MPEG4 file must have an IDR at the beginning of a file. Some files have none except the first IDR frame.

Unlike MPEG2, where P and B frames reference to the last I frame and any information past the last I frame is lost, MPEG4, for the sake of efficiency, can reference all the way back to the last IDR. Every IDR is an I frame, but is also deemed a reference frame, where all other I frames in the stream are not.

MP4 is a container format much like AVI and it can be used to "house" many different types of compression codecs, not just H.264. It is true though, that MP4 is a very popular choice for the H.264 format.

File Containers and Wrappers

At this point, it is time to assemble the video and the audio compressed media and to put it in a container or wrapper. What are those, you ask? We need to organize the media, audio and video essence as it is often called, so that upon playback decode, the player knows how to extract the media.

There are a couple of basic ways to do this. The first is to treat the entire program as a file and copy and transport it using standard computer networking methods. The second is to stream it. Streaming implies that the receiver of these streams can start decoding and playing back of the file before the end of the file is received. When captured replay footage is moved from one server to another, it very well might be treated as a file copy, but when it is played back to air, depending on the server, it might start out as a stream, and then be converted back to baseband audio and video as it leaves the server. But invariably, when video is transported back to the NOC, from the NOC to the local affiliate, from the local affiliate's studio to its transmitter, and then broadcast over the air to viewers, that playback video finds itself in a stream more than once.

Some servers store all the media in a single self contained file and stream out media in a single stream, while others keep all the audio and video files separate internally. The servers that keep media files separate internally and externally will produce a wrapper for those media files, but the wrapper won't actually contain any media, just 'pointers' pointing to the various files holding the video and audio file. These types of wrappers are called "reference" files, as all they contain is references to outside files. These files can be copied or moved between servers but not played out to end users. The other type is called "self-contained", as the wrappers actually have the media essence contained within.

So to be able to stream, either live or out of a server, you obviously need a self-contained wrapper. The wrapper file must have one more important attribute; it must have the audio and video interleaved. That means you continuously send enough audio chunks, to keep up with the video delivered. These chunks of media will arrive and then be buffered out in time with each other.

There are also two types of streams; Program Streams and Transport Streams. Program Streams are designed for reasonably reliable data paths, such as are found in a DVD. A program stream also only carries a single program. That single program can contain a video and multiple audio tracks. A transport stream is intended for non-reliable paths, such as over-the-air ATSC broadcasts or microwave and satellite transmission. A transport stream is also able to carry multiple programs at once. Let's start with MPEG's transport stream.

MPEG Transport Streams

Transport Packets are the key to flexibility and extensibility. They are fixed length packets that are relatively short, and are amenable to error correction, and fast switching. This is best for error prone media such as terrestrial broadcast.

Transport Stream Attributes:

- Each packet has a header or label that identifies the contents (PID).
- Each packet contains only one kind of data (audio, video, etc.)
- Packetized Elementary Streams (PES) is what comes out of the video/audio coder, or what incoming data becomes. These PES streams are the same program streams we just mentioned above.
- A PES has a header and a video, audio, or data payload
- A transport stream consists of fixed length transport packets, which are re-packaged PES packets
- A PES packet header is always preceded by a transport header
- A Transport packet is 188 bytes long.
- It consists of a: 4 byte Transport Header which includes

    8 bit sync word
    3 consecutive 1's
    13 bit PID
    2 bit scrambling flags
    an adaptation field (program clock reference (PCR), etc.)
    a video, audio, or data payload up to 184 bytes (if no adaptation field)

- There is a packet scheduler, and mux that permits packets into the bit stream according to need and priority. This allows dynamic allocation of the channel

A point of common confusion here: Transport packets are 188 bytes long, but elementary streams (PES) coming from an MPEG encoder to a multiplexer which will build the final transport stream can be up to 65,536 bytes long. The multiplexer that puts multiple PES into a single stream will break the various long PES into transport sized packets.

In the transport stream packet header is an indication as to whether the start of a PES packet occurs in that transport stream packet and whether the payload is scrambled. The first byte of each PES packet header is located at the first available payload location of a transport stream packet. The way individual PES can be identified in the MPEG transport stream is with a Packet ID (PID).

To organize the transport streams, a hierarchy of tables is sent often, usually at least a few times a second.

The root table is the Program Specific Information (PSI) table. PSI data can be up to 1024 bytes long. The beginning of this information is indicated by a pointer in the transport stream packet payload. It contains the Program Allocation Table (PAT), the Program Map Table (PMT) and the Network Information Table (NAT)

- The PAT serves as the starting point for locating the various PMTs
- The PMTs of the various programs are listed in the Program Allocation Table (PAT)
- The PMT is the map for all PIDs of a given program

Example PMT map for a program in a transport stream	
PID #	Stream Type
32	PMT
33	Video & PCR
36	Audio
42	Data

- Besides PMTs in the PAT there can be a PID that identifies the Network Information Table (NIT). Therefore the PAT has the NIT and various PMTs.
- PAT and PMT are required in every MPEG-2 transport stream. They form a "mini program guide".
- Because it is possible for a particular PES to be used in multiple programs, some PIDs could be in multiple PMTs.
- The PID is a 13 bit field in a MPEG transport packet
- PID bits 0-3 define whether the PID is a video, audio, or data PID, or the PID for the Program Map Table (PMT)
- PID bits 4-11 of the PID is the program number
- This allows for 256 separate programs
- This specifies which program the PID is applicable to
- PID bit 12 is 0.

Reserved PID values:

- 00 = Program Allocation Table
- 01 = Conditional Access Table
- 1FFF = Null packets

If all 13 bits in the PID field are high, it signifies that this is a null packet. These are used to stuff the data stream when there is no other data to send.

Sample top table hierarchy

PSI Info		
Table Name	PID #	Description
Program Association Table(PAT)	0	Associates program # with PMT
Program Map Table (PMT)	Assigned	Associates PID with program(s)
Network Information Table(NIT)	Assigned	Contains physical network

Example PAT for 3 program multiplex		
Program #	PMT PID #	Meaning
2	32	PID 32 contains map for program 2
3	48	PID 48 contains map for program 3
4	64	PID 64 contains map for program 4

PAT provides correspondence between a program number and the PMT PID that carries program definition. The program # is similar to Channel # in broadcast TV. The PAT is always assigned to PID 0. ATSC programs in range 2-255.

Each PID # above would represent a PES stream. Together they represent one program stream. Each program has a PMT. The PID for every program's PMT is in the PAT. PMT provides correspondence between a program number and the elementary streams that comprise it.

Packet de-multiplexing is done by reading the PIDs, and based on info from the PAT, and PMT tables, sorting the packets.

The confusing thing about PIDs is that they can be found at 3 levels of the table structure, the PAT, PMT, and for individual PES. To identify where a particular PID belongs in the hierarchy of the MPEG stream (with the exception of PIDs 01 and 02) is to work down from the PAT.

The overall system multiplexing approach can be thought of as a combination of multiplexing at 2 different layers.

In the first layer, single program transport bit streams are formed by multiplexing transport packets from one or more Packetized Elementary Streams (PES) sources. In the second layer, many single program transport streams are combined to form a system of programs.

The Program Specific (PSI) table contains information relating to the identification, and the components of each program.

MPEG Transport packets are 188 bytes long. This is so they fit the payload capacity of an ATM packet which is 188 bytes. The first 4 bytes are the MPEG header with a sync byte with a value of 71h (0100 0111b). The next 2 bytes have the first 3 bits used as flags and last 13 bits for the PID

After the header, 184 bytes of payload can be sent. This would usually be compressed video or audio data PES packets. But the payload could be data from one of the PSI tables. Any of the 184 bytes not used would be filled with null bytes (FFh).

MXF Container

A very common container format used for high end television applications is called the Material eXchange Format (MXF). It is used to carry most of the common video codecs' essence; DNxHD, DV, DVCPro, and XDCAM among others. Besides carrying the media essence, it also carries extensive metadata. This metadata can be timecode and Advanced Authoring Format (AAF) data. This means that besides normal media it can carry still images, graphics, text such as production notes, or Edit Decision Lists (EDL), essentially a complete project package in a single container.

The MXF wrapper concept works on KLV (Key Length Value). The Key is usually 16 bytes reserved for use as globally registered unique identifiers that detail what the payload or value will be; various video and audio codecs, or the type of metadata. The length indicates how long the packet will be, the value is the actual payload. A decoder that doesn't understand a key value will skip it and look at the length value and skip to the start of the next packet. This is referred to as dark data. Since MXF supports only a subsection of AAF, this is often AAF metadata. MXF streams are broken up into Partitions; these are the smallest elements that have enough audio and video elements to provide synchronized playout.

AVI Container

Audio Video Interleave (AVI) is an early container format introduced by Microsoft in 1992. AVI files allow synchronous audio/video playback. Like the DVD video format, AVI files support multiple streaming audio and video, although these features are seldom used. AVI also divides the files' data into chunks. Each of these chunks is given a tag. There is a chunk at the beginning of the file, the file header that contains metadata about the video, such as its width, height and frame rate.

AVI does not provide a standardized way to encode aspect ratio information, and time code can not be embedded in AVI files. AVI was never intended to contain variable frame rate material, nor contain MPEG type temporal (multi-frame) compression, although methods now exist to do this.

AVI was not intended to contain video using any compression technique which requires access to future video frame data beyond the current frame. Approaches exist to support modern video compression techniques, (such as MPEG-4) which rely on this function, although this is beyond the intent of the original specification and may cause problems with playback software which does not anticipate this use.

AVI cannot contain some specific types of variable bitrate (VBR) data reliably (such as MP3 audio at sample rates below 32 kHz).

Overhead for AVI files at the resolutions and frame rates normally used to encode standard definition feature films is about 5 MB per hour of video, the significance of which varies with the application.

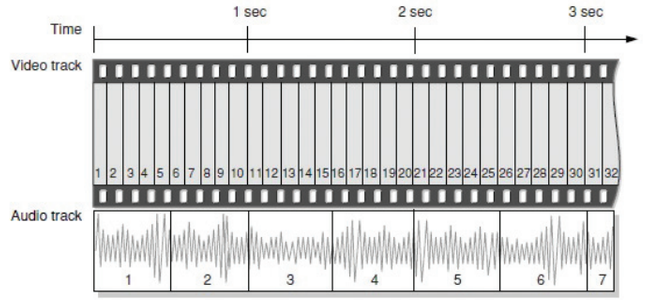
Quicktime Containers

Quicktime (QT) files are comprised of atoms. Whereas in an MXF file, the Key indicated the start of a collection of data, so do atoms. Atoms are hierarchical in nature, as one atom can contain one or more other atoms of varying types. These child atoms can also contain atoms and so forth. Atoms that don't contain any atoms themselves are called Leaf atoms. With the exception of handler description atoms, as we will see, atoms do not have to be in any order. Handler description atoms have to come before their data. These headers have fields that indicate the atom's length and its type. All types are four letter (32 bit unsigned integer) descriptions.

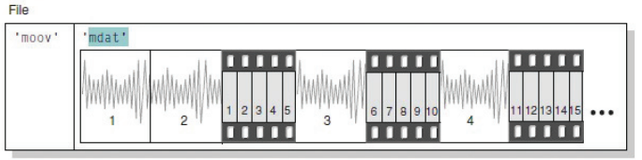
With atoms inside atoms, QT is structured to allow quick parsing between parents, children, and grandchildren atoms. Included in the headers are offsets to child atoms and unique index numbers so that if there are multiple child atoms of the same type the right atom is addressed.

Below is the atom hierarchy of a short 10 second QT file. Top level atoms are 'ftyp', 'mdat' and 'moov' atoms. The first one indicates the file type, the second is the actual media payload, and the final is information about the attributes and components that make up the file.

Each file chunk is a child atom of mdat; 'vide' indicates video data, 'soun' indicates audio, and 'tmcd' indicates the time code track. With the video tracks, each of the samples represents a video frame. An important attribute in the file above is that you will notice that video and audio chunks are interspersed with one another, or 'interleaved'.

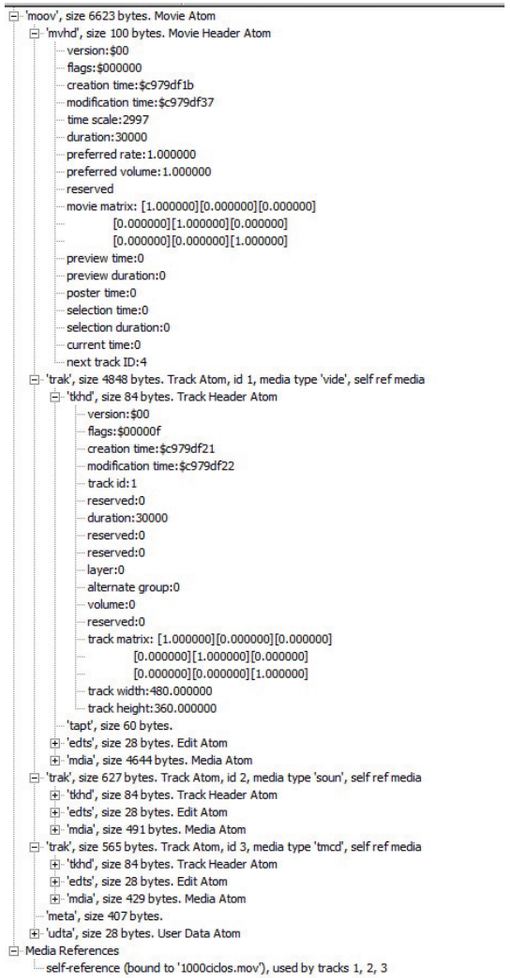


Non-interleaved file, separate side-by-side video and audio tracks.



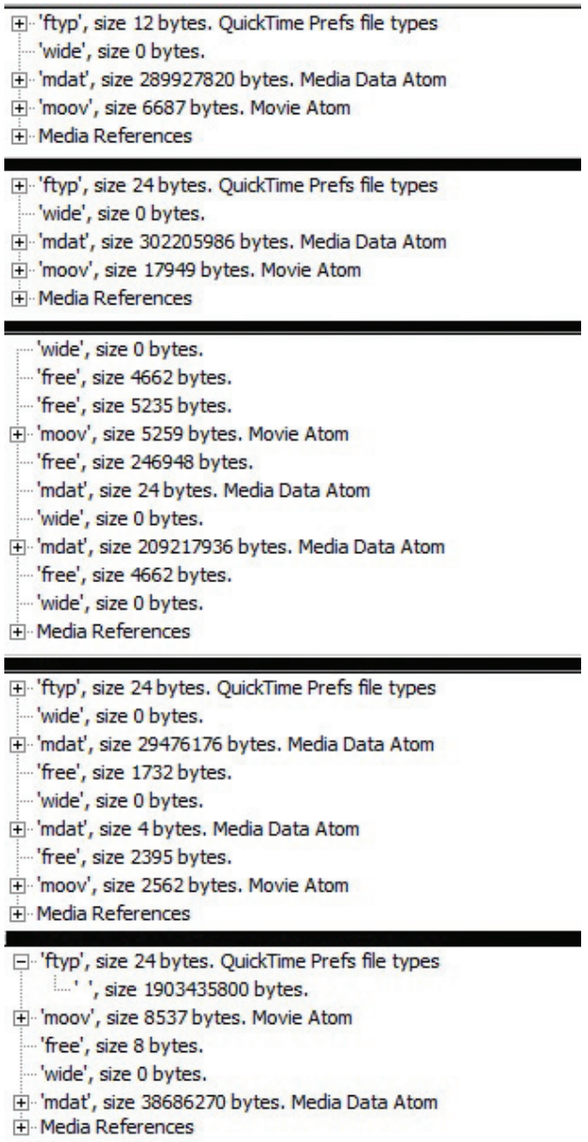
Interleaved file, corresponding audio data is kept in close proximity with the video.

Shown above is an example of the video samples being broken out. The sample labeled "Sync Sample" is used in conjunction with a sync sample table indicating where these samples are located. These samples are used to rapidly find a random location in the file. This is similar to the process of hinting that some formats use. Hinting is the process of putting additional headers in the file which end up as an additional track. What these headers do is tell a decoder/receiver where to find items in the stream. Often I frames are all marked in this hint track, segment start/stops, CC packets, etc. Basically it allows a decoder to know where various things are in a file without having to take the time to decode packets until it finds what it's looking for.



The 'moov' atom contains many child atoms that hold metadata about the movie, such as the number and type of tracks, location of sample data and other attributes. The 'mvhd' atom indicates that this is a self-contained file, versus an 'rmra' atom, which represents a reference file.

Finally, as was mentioned earlier, the order of most atoms can vary. A QT decoder needs to obtain the data in the 'moov' atom before it can start to decode the media data in the 'mdat' atom. Below are five separate QT files; as you can see, they can be organized a number of different ways—like MXF files, QT decoders are designed to ignore atoms they don't understand and to quickly parse to where they need to be first, second, etc.



Conclusion

The media industry will continue to slowly move from live-real-time baseband audio and video to streaming and non-real-time media transport. Even in the live realm of remote television production most production before the event will by moved and acted upon as file based entities. It is only when the event begins that most media associated with the production will be played out in real time. But often the real time output of media will still be in formats, such as a transport stream, that more closely relate to the IT world than to traditional television. While transport streams are the primary way media is transported from the truck venue back to the broadcaster or network NOC today, more and more contribution feeds to the truck from upstream providers will arrive as transport streams also. With ever increasing strides in hardware and software processing power, file based media will only continue to increase its share of the workload in all areas of television production.