
FLUFFY SPIDER TECHNOLOGIES

# MPEG(2) Encoding & Decoding

## AN ANALYSIS AND FEASIBILITY STUDY

AUTHORS: CARSTEN HAITZLER / NICHOLAS KLOPFER-WEBBER

Fluffy Spider Technologies Pty Ltd.
Suite 87
330 Wattle Street
Ultimo NSW, 2007

phone: +61 2 9281 9055
fax: +61 2 9281 2944
email: info@fluffyspider.com.au
web: www.fluffyspider.com.au
abn: 72 070 239 217

# MPEG(2) encoding & decoding fundamentals

## Prologue

This document intends to give the reader an insight into how feasible it is to write a real-time MPEG2 encoder to run in software on a modern Intel x86 CPU (1Ghz or higher). As a bi-product of this it is necessary to start going into image compression from JPEG on as it was this standard that gave birth to MPEG and then MPEG2. It is the same compression techniques pioneered in JPEG carry onto its more complex cousins MPEG and MPEG2. JPEG, being simpler, is a good place to start in understanding all about video compression, and builds a solid base for later concepts.

## What is MPEG/MPEG2?

MPEG and MPEG2 are almost the same, so this covers both, with MPEG2 being a later version of the specification allowing for higher resolutions, interlaced video and slightly better compression. Otherwise the principles between MPEG and MPEG2 are so close that they can be treated the same. Since MPEG is a little simpler than MPEG2, it is a good place to start. There are many similarities to JPEG in terms of encoding, colour space and other parameters of an MPEG file, so understanding JPEG fully is essential to grasping its bigger and more extensive cousin, MPEG and MPEG2.

## A basic guide - What is JPEG?

JPEG is an encoding standard for still images, mainly photographs, allowing for loss of data to achieve high compression rates. It makes use of the fact the human sensory system is less acutely aware of certain aspects of imagery, and so can afford to remove data with little, or no impact to a person viewing the image. It also combines run-length bit compression and standard Huffman encoding techniques to take the resultant data where information has been removed, and turn it into a small bit-stream of information.

The first big difference between conventional RGB colour space and the colour space used in JPEG is that JPEG uses YUV (also know as YIQ and YcbCr, depending on your RGB to YUV conversion co-efficients). YUV is a different way of representing colour in terms of brightness (luminance) which is the Y element, and then U and V which define the colour and saturation of the pixel (in combination). Since the human eye is most sensitive to brightness, and less sensitive to colour and saturation, this means the colour resolution of U and V and the spatial resolution of U and V can be reduced with little visual impact for the viewer, providing quick compression just by removing a lot of redundant information.

Converting between YUV and RGB is actually a fairly expensive process for a CPU to do, and should be avoided if possible. It required interaction between each channel to provide each input or output channel, requiring either several operations or several lookups per colour channel. The equation to go for RGB to YUV is:

```
y =  (0.257 * r) + (0.504 * g) + (0.098 * b) + 16;
u =  (0.439 * r) - (0.368 * g) - (0.071 * b) + 128;
v = -(0.148 * r) - (0.291 * g) + (0.439 * b) + 128;
```

And to go from YUV to RGB is:

```
r = (1.164 * (y – 16)) + (1.596 * (u – 128));
g = (1.164 * (y – 16)) - (0.813 * (u – 128)) - (0.391 * (v – 128));
b = (1.164 * (y – 16)) + (2.018 * (v – 128));
```

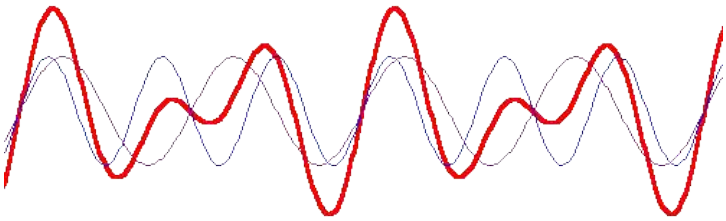
As you can see the operations per pixel just to convert are relatively hefty. This can be optimized by means of lookup tables and fixed point arithmetic, but still does not remove the work. When a video stream for television (720x480 pixels at 30 frames per second) could involve having to convert more than 10 million pixels per second. For decoding this would involve something like 180 million operations per second just to convert colour spaces. Of course this can be optimised, but this is just an indicator of what is needed. It gets worse the

more you learn about JPEG, MPEG etc.

As an aside RGB colour space is what most output devices accept natively, most graphics cards, input devices etc. It is one of the simplest colour spaces to work in mathematically, and is easy for people to understand. Thus it almost always appears when doing graphics. YUV is optimal for human perception but is harder to work with algorithmically. It leads to better compression capabilities, but means more work going to and from RGB. Today's machines provide video capture hardware that can provide YUV data natively and graphics chip sets that can accept YUV data to scale and display saving the CPU from doing an expensive colour conversion process. If at all possible this hardware should always be used if provided, and needless RGB to YUV and back conversion should be avoided like the plague.

Another large part of the encoding and decoding process is known as the DCT (Discrete Cosine Transform). It is this stage that turns a 2 dimensional spatial representation of image pixels into a linearised frequency domain representation. In simple terms it describes how a set of pixels can be described by a set of horizontal and vertical Cosine based waveforms superimposed and each waveform multiplied by a given value, indicating pixel values for a set of pixels within the image.

This is based very much on the same theory that has been well known for audio signals. Any given linear amplitude waveform graph can be re-built given a sufficient number of constituent Sine (or Cosine) waves each at a different frequency and amplitude (see Drawing 1). Given enough of these any waveform can be re-built and thus could be encoded as a set of frequencies and multipliers.



*Drawing 1Simple sine waves making more complex waveform*

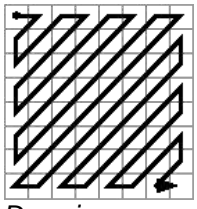
Because the human ear is less likely to hear very high frequencies or very low ones, frequencies close together, high frequencies immediately after a high volume sound (a drum beat, crash etc.), a psycho-acoustic model can be built that reduces the number of frequencies encoded, so the “softer” sounds do not get encoded because the ear will not hear them, and other techniques.

Based on similar principles, JPEG (and MPEG and MPEG2) create a linear “waveform” of pixel values taken from an 8x8 selection of pixels,  This is then put through an DCT algorithm to create a list of 64 co-efficients for the block that can be used in re-building the waveform from a series of Cosine waveforms multiplied by these co-efficients. The reason DCT is used instead of FFT (Fast Fourier Transform) is that a DCT can approximate a straight line much more readily than FFTs can with fewer co-efficients.

| | ***Pixels*** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Original | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 |
| DCT | 100 | -52 | 0 | -5 | 0 | -2 | 0 | 0.4 |
| Truncate | 100 | -52 | 0 | -5 | 0 | -2 | 0 | 0.4 |
| IDCT | 8 | 15 | 24 | 32 | 40 | 48 | 57 | 63 |

*Table 1Example 8 element DCT and IDCT*

All the co-efficients are divided by a particular value taken from a Quantization table to increase the resolution of the most important co-efficients (those at the top-left) and decrease the accuracy of the less important ones. This makes it easier to throw out co-efficients that will not be significant in the image reproduction. This table is hand crafted to make the best of quality and compression based on “real world” input images.



*Drawing 2How DCT values are scanned in a block*

The co-efficients are then scanned diagonally (see Drawing 2). The advantage of doing a diagonal scan is that the largest (and most visually significant) values in the DCT end up being the first values in the list. Quick compression can be achieved simply by throwing out the smaller elements (see Table 1). When rebuilding the input signal using just half the co-efficients (the IDCT step) the result is remarkably close to the original, with only half the data.
Once the DCT co-efficients have been linearised and reduced to only the most important values at the start, this set of numbers are encoded as a relative stream of values (since each value is normally close to the previous one). The excess “0” values that will be found in this encoding are encoded as skip-blocks, and then the final set of numbers are fed to a Huffman encoder for compression.

Each block of pixels has its Y, U and V planes encoded this way (generally with the U and V planes encoded at half the horizontal and vertical resolution compared to Y because the eye is less sensitive to colour than it is to luminance). The encoder will encode all the pixels this way producing a stream of encoded blocks.

**JPEG to MPEG**

A need for encoding motion arose and many of the same experts involved in JPEG became involved in defining the MPEG standard. A lot of the MPEG standard is very much like JPEG due to the experience of the groups members in defining JPEG as well as its success and ability to compress images very well. Some mistakes where made when defining the JPEG standard that could have allowed for better compression and some of those lessons were adopted as part of MPEG and later MPEG2. These changes though are minor and the basic principles still hold.

This is the core of encoding a still image as a JPEG, and is also the exact same principles that MPEG and MPEG2 build on for all their video encoding. MPEG adds advanced features such as temporal redundancy reduction, more spatial redundancy reduction and greater use of temporal information to produce more spatial data.

**What exactly does MPEG add?**

MPEG provides almost what JPEG does when we talk about MPEG's I-frames. MPEG delivers a file format that encoder multiple frames (images) intended to be sequenced one after the other to generate animation. MPEG also allows for audio, but that is out of the scope of this document. MPEG has 3 kinds of frames. I, B and P frames, as they are known.

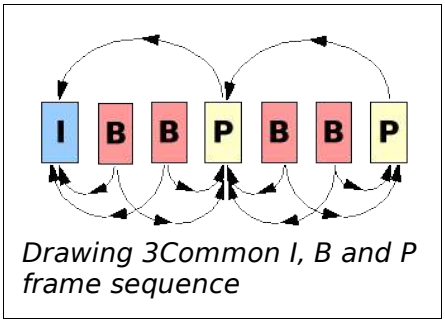
The I frames are fully encoded images. They do not rely on data from any other frames or images anywhere else in the stream and thus are often a common point of reference or starting point for a sequence following that image. These frames (for all intents and purposes) are basically JPEG images, with all pixels encoded in DCT blocks.

P frames are “Predicted” frames based on information from a previous frame. You must have decoded the previous frame in order to make any use of a P frame.

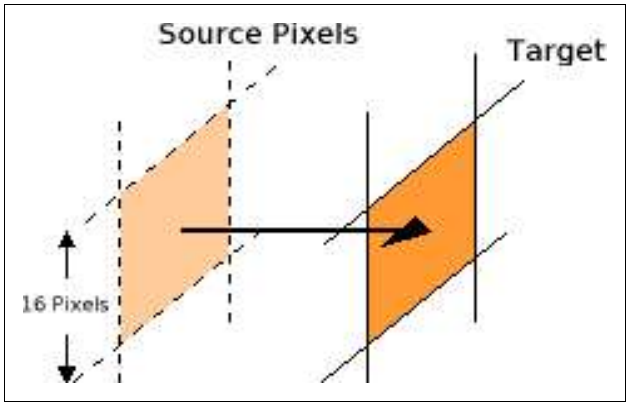
B frames are “Bi-directionally predicted” frames. They rely on information from BOTH a previous AND a following frame and thus require having decoded both frames they rely on to display a B frame.

In MPEG1 you will often find the sequence of frames to be IBBPBBP. This sequence repeats and is known as a GOP (Group Of Pictures) and could be seen as the “fundamental” unit of video in MPEG. The sequence starts with a known defined “state” initialized with the I frame. It contains 2 P frames, the first being based off the I frame and the second being based off the first P frame. The B frames sandwiched in between use the previous I (or P) and following P frame as references. To decode any particular frame in a GOP it is necessary to decode all frames it depends on, and so on down the dependency tree until you decode only an I frame.

From an encoder perspective, it is possibly necessary to buffer several frames of input before it can produce an MPEG output sequence. This depends on the choice of I, B and P frame sequence, But adds to the complexity of the encoder, its memory requirements and bandwidth, and the cache affinity of data.

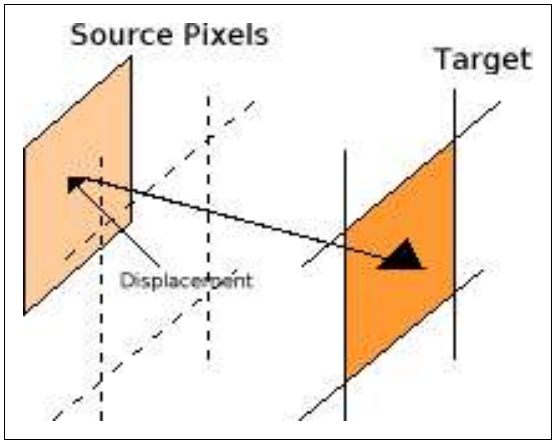


*Drawing 3Common I, B and P frame sequence*




In addition to having different frame types, each “block” of pixels in each frame can be of the type I, P or B. I frames contain only I blocks. P frames contain I or P block types and B frames contain either I, P or B type blocks. Each block (macro block of 16x16 pixels) can also introduce a motion vector. Since a lot of film involves the camera panning, or items moving along the film, motion vectors are designed to detect motion within a set of frames and reduce stream data by using data from another location in a previous (or future) frame as reference data.

This allows the encoder to indicate motion of a block of pixels as well as a correction based on anything else that might change. MPEG even allows to specify a motion vector in non integer pixel unites (up to 0.5 pixel resolution) allowing very slow panning motion to be encoded more efficiently.




The advantage of such complex encoding methods, such as motion vectors, allows for much better quality video for the same compression, since it makes use of common every-day properties of video. The problem comes with the encoder where it now has a much more complex task in trying to efficiently produce an MPEG stream using such features, as finding an appropriate motion vector which would result in a minimal delta block is a slow process because the image pixels need to be scanned multiple times.

## Existing solutions

There are very few existing MPEG2 encoding solutions that are open source and publicly available. Two available are:

1. The primary one is the reference implementation, and it is only capable of encoding at 1/7$^{th}$ the speed needed for real time encoding (platform: 1.6Ghz Athlon desktop).

2. **FFmpeg** (http://www.ffmpeg.org) very recently attained MPEG2 encoding (as of version 0.4.8, the latest version as of this document), and is capable of encoding a video quality (720x480@29.97fps) stream on a 1.6Ghz Athlon with about 50-60% CPU overhead. It doesn't seem to encode perfectly smoothly, stuttering with some frames, but that is likely an issue with the encoder itself, as there is CPU to spare.

   Tests on a slower platform (PIII 600MHz Intel desktop) at ¼ resolution only used 30-40% Cup, however exhibited the same stuttering as with full resolution on a faster platform. It is possible that since the Cup usage is so low the problem lies with memory bandwidth and buffering of the captured frames not in the encoding itself.

   Further testing without a capture card, from raw YUV to an MPEG stream were not helpful as FFmpeg uses different scheduling policies for real time encoding. All the tests took longer to encode than the original video takes to play. These tests were performed on a 1.2GHz Athlon desktop.

## FFMPEG Licensing

From the FFmpeg Web Page:

> *FFmpeg is released under the GNU Lesser General Public License. That means that if you make a modification to this program/library and distribute the resulting program, you must republish your modifications under the GNU LGPL License.*
>
> *As an exception, the libraries liba52 and libpostproc are released under the GNU GPL License. If you use them in FFmpeg then the whole FFmpeg project becomes automatically released under the GNU GPL License.*
>
> *Note for the use of FFmpeg in commercial products: alternate licenses are not available for FFmpeg. So you must either use FFmpeg under the LGPL license or not use FFmpeg at all in your product.*

## Implementation Considerations

## Disk Performance

Although disk IO should not be a problem, the strategies employed by the Operating System (OS) for buffering data before it is written back to disk will likely need to be looked at.

Modern disks can easily keep up with writing encoded data with time to spare. It only becomes a problem when the OS delays writing back data such in the case of a Journaling File System (JFS), where writes may be delayed and combined to be written back on one go. When a large write occurs it can delay other processes causing them to miss time critical operations such as reading from the video device. In this instance it may be necessary to fine tune how often these writes occur to minimise or completely remove any delays made to other processes.

## Subtitles

When capturing video there are parts of the image that are not viable but contain extra information. These are contained in the banded regions around the image. Close captions are encoded in these banded regions and allow Televisions capable of decoding the

information to display subtitles over the image.  MPEG2 allows for subtitles to be stored along side the audio and video.

The encoded MPEG stream doesn't contain enough information to properly decode these banded regions on playback.  For this reason and to get better compression the banded regions are usually left out of the encoded data.  However it is possible to decode the subtitles during the encoding process and save them along with the audio and video as a subtitle stream.  DVDs often contain these subtitle for different languages.

Depending on the target device it may not be possible to decode the subtitles without affecting the performance of the video encoding.  This would require testing and possible optimising.