Faculdade de Engenharia da Universidade do Porto

Platforms for Handling and Development of Audiovisual Data

José Pedro Sousa Horta

Project/Dissertation concerning
Mestrado Integrado em Engenharia Informática

Advisor: Professor Eurico Carrapatoso

July 2008

Platforms for Handling and Development of Audiovisual Data

José Pedro Sousa Horta

Project/Dissertation concerning
Mestrado Integrado em Engenharia Informática

Approved in public display by the jury:

President: Prof. Doutor António Fernando Coelho

Examiner: Prof. Doutor Nuno Magalhães Ribeiro

Vowel: Prof. Doutor Eurico Manuel Carrapatoso

July 17th 2008

Abstract

Everywhere around us, digital is replacing analogue. Such is especially true in the audiovisual: be it in consumer or professional market, the advantages of computer-based media have quickly persuaded investors. To choose in which technologies, software should be based, proper understanding of the major aspects behind digital media is essential.

An overview of the state of the art regarding the compression of digital video, the existing container formats and the multimedia frameworks (that enable easier playback and editing of audiovisual content) will be given, preceded by an introduction of the main topics. The professional video market is particularly focused on this document, due to the context of its elaboration, predominantly MPEG-2, MXF and DirectShow. In order to obtain more accurate results, fruit of the practical contact with the technology, a hands-on programming project was accomplished using DirectShow to playback both local and remote WMV files.

Compression algorithms are continuously being improved and new ones invented, the designers of container formats are paying increased attention to metadata and some serious alternatives to DirectShow are emerging. In a continuously mutating environment, light is shed at what has, what is and what will be happening in the upcoming years on the digital video scene. Results show that the choice of a particular video coding format or multimedia framework depends heavily on the purpose of its application and that there is no universal solution. Various use scenarios are given an answer as to which is the best course of action concerning which technologies to apply. The software developed was ultimately given a positive feedback by the client.

Keywords: Digital video compression, Digital video containers, Multimedia frameworks; DV, Motion JPEG 2000, MPEG 2 / 4 / 21, ASF, MXF, DirectShow, Helix DNA, QuickTime

Resumo

Um pouco por todo o lado, o formato digital está a substituir o analógico. Este facto torna-se realmente óbvio no mundo do audiovisual: seja no mercado profissional ou de consumo, as vantagens da *media* baseada na informática rapidamente convenceram os investidores. De modo a facilitar a escolha de quais tecnologias digitais que devem ser empregues, um conhecimento suficiente dos principais aspectos por detrás do tratamento de *media* numérica é essencial.

Uma vista geral do estado da arte dos formatos de compressão e transporte de vídeo digital e das plataformas multimédia (que permitem o fácil manuseio e edição de conteúdo audiovisual) será efectuada, precedida por uma introdução aos tópicos principais. O mercado de vídeo profissional é particularmente tido em conta neste documento, devido ao contexto da sua elaboração, predominantemente as tecnologias MPEG-2, MXF e DirectShow. De modo a obter resultados mais pertinentes, foi concluído um projecto prático baseado em DirectShow com o intuito de reproduzir ficheiros WMV, locais e remotos.

Os algoritmos de compressão estão continuamente a ser melhorados e inventados, os formatos de armazenamento têm cada vez recursos dedicados a *metadata* e sérias alternativas ao DirectShow começam a emergir. Num ambiente em contínua mutação, é levantado o véu sobre o que aconteceu, o que está a acontecer e o que irá acontecer nos próximos anos, na área do vídeo digital. Os resultados obtidos indicam que a escolha de um formato de ficheiro de vídeo particular ou de uma *framework* multimédia depende imenso da utilização pretendida, não existindo soluções universais. Vários cenários são considerados e várias soluções recomendadas acerca das melhores opções tecnológicas a adoptar. A aplicação implementada foi avaliada positivamente pelo cliente.

Acknowledgements

I would like to thank all my colleagues at MOG Solutions for an outstanding work environment that considerably eased the flow of work. My special thanks to my MOG Solutions responsible, Vitor Teixeira, for letting me find my own pace and for his valuable knowledge, to Luis Barral for his priceless expertise on DirectShow programming and his will to share it, to Orlando Ribeiro for his constant availability to help and to Pedro Campos for keeping me motivated along these four and a half months.

Big thanks to my faculty advisor Prof. Eurico Carrapatoso for kindly embracing this project.

And a final thank you to my brother Frederico, for providing me with his layman's insight when reviewing this document.

Table of Contents

Figure List

Table List

1. Introduction

This report refers to the project Platforms for Handling and Development of Audiovisual Data issued through the collaboration of the Faculdade de Engenharia da Universidade do Porto and the company MOG Solutions. It stands as the final project for the completion of the Mestrado Integrado em Engenharia Informática e Computação degree, on the mentioned institution.

1.1 Scope

The range of topics covered on this document is vast, but all falls under the common denominator of digital media. It follows a comprehensive approach of introducing the reader to key aspects of each of the main subjects, portrayed in a deep enough style to teach something to the area experts but still maintaining a not too technical speech, suitable for layman readers.

On video compression formats, not only the known formats specifications, such as the lossless Motion JPEG 2000 or the lossy MPEG-4, are focused but also the underlying techniques used on most video encoding formats (those that take advantage of the human senses' limitations) and the more generic ones used on all kinds of digital data. Variations of the main compression techniques are not described, just the generic algorithms, which are sufficient to easily understand their variants.

Under video container formats, a clear definition of this kind of file format introduces the description of the major formats of the professional market. Two lists of analysing software were populated for both video compression and container formats.

For multimedia frameworks, subsequent to explaining what they are, the main software solutions available are illustrated. Afterwards some less important platforms are referred, and finally a wide-ranging comparison is made among the main frameworks.

It is also detailed a programming exercise in C++ on DirectShow, intended to expand a media player with WMV advanced playback.

1.2 Motivation

MOG Solutions develops an effort of continuously updating its knowledge of the state of the art of the extensive existing array of coding and wrapping of audiovisual content technologies, as well as, the use of such tools in the Digital Cinema and Television markets.

It is a part of the daily work of MOG Solutions operatives, the contact with clients from all around the globe and the analysis of the structure of multimedia encoded in a variety of formats (MXF, AVI, MPEG, etc). Such activity demands that the support line operatives to have rapid access to a knowledge base of technical information and diagnostic tools that permit a quick assessment of the received files correctness.

With this internship, it is expected the compilation and development of a base of knowledge and a study of a set of tools necessary to optimize the work of the aforementioned operatives.

One of the most important tools within the company is MOGPlayer that allows the playback of audio and video, for both visualization and editing purposes. This application makes use of the DirectShow framework and the augmentation of its features, by adding support to new compression formats, is intended. To further understand the methodologies of handling compressed video, an exercise on DirectShow made the problems and advantages of using a multimedia framework, much clearer.

Being a movie and series *aficionado*, the processes behind the coding and rendering of audiovisual content have fascinated me for long. Keenly, I embraced this project, aimed at determining which components were involved and how the internal procedures toiled to reproduce multimedia content. I was further stimulated while studying the digital data compression algorithms, finding the applied concepts and ideas very interesting, specially the practical aspects of their implementation.

1.3 Objectives

The initial main objectives for the successful completion of this project were:

- Compilation of a base of knowledge in the areas of coding and wrapping of audiovisual material, and its dissemination through the different areas of the company in an appropriate manner, resorting to distinct vehicles of information such as databases or wikis;
- Practical application of the acquired information in testing of existing or projected tools that allow an analysis of different encoding and wrapping formats of audiovisual content, such tools should confer the optimization of specification, development and testing of the processes developed by the company;
- A study about the current most used multimedia frameworks is proposed and the use of the attained knowledge to develop applications for editing and playback of video.

A more detailed specification of the desired results is:

- Integration of the intern in the enterprise and its workflow;
- Definition of a supporting format for the knowledge base and the process of its propagation throughout the company, heeding the specific needs of each department;
- Study of the main technical aspects of the selection of coding technologies, including lossy formats (e.g. MPEG-2, DV, VC-1 and JPEG 2000) and lossless (e.g. DPX or BWave);
- Study of existing or developing tool for analysis of the previously chosen coding formats;
- Study regarding the chief technical facets of a selection of audiovisual wrapping technologies (such as MXF, ASF, AAF or QuickTime);
- Study of developing or existent tools for analysis of the previously chosen wrapping formats;
- Comparative study of the central aspects of selecting a multimedia framework (example: DirectShow, Microsoft Media Foundation, Java Multimedia Framework or QuickTime);
- Development of extra functionalities of an application for visualization or editing of video and associated medias;
- Assemblage of information, news groups, forums and open sourced developments, regarding the frameworks.

A short while after the arrival at the company, it was assessed that there was no need to create different versions of the same document, due to the similar technical level of all debugging operatives, regarding the knowledge of video coding and containing formats. Therefore, it was acknowledged that the report format would be sufficient, thanks to its structure that allows a rapid search of the needed contents.

Some of the initially considered formats for study were dropped or exchanged by other more meaningful ones.

1.4 Structure

The document is divided in 6 major parts: an introduction to the project and its report, digital media compression formats, digital media container formats, multimedia frameworks, DirectShow implementation and conclusions. Each of the three major chapters contains an introduction and conclusion, besides the actual contents.

Firstly, the reasons behind the elaboration of this project are introduced and the structure of the report examined.

On video compression formats, the major generic concepts and algorithms for digital data compression are exposed, continuing towards more video specific techniques and lastly to completely defined video formats.

Video wrapping formats (used to contain the previously discussed compression formats) are contextualised and depicted, and the major standards available today presented.

Finally, the concept of multimedia frameworks is exposed and the current, most relevant solutions comparatively studied.

Afterwards the practical contact had with the programming of a DirectShow playback application is portrayed.

To terminate, the realized conclusions are presented, elucidating on the original content present and the future work envisioned to further extending this project.

In annex, a list of software solutions for the analysis of multimedia files' structures for both the discussed types of formats is presented, and the code for a simple player is available for each of the studied multimedia frameworks.

A bibliographic review chapter for the implementation made was put aside, in view of the fact that the equivalent content exists in Digital Video Compression and Multimedia Frameworks chapters.

1.5 Digital Video

As computers start to be omnipresent, increasingly powerful and smaller each day, most analogue processes are becoming digital, improving performance, whilst considerably reducing operational costs. A part of those processes involves video (with the associated audio and metadata), be it its capture, playback, editing or storage.

Digital video has definitely taken charge of the consumer and professional markets. Several dedicated, embedded devices exist to handle most tasks related to the editing, broadcast and reproduction of digital media, but it is in the polyvalence and performance of personal computers that the principal solutions rely on.

The challenge nowadays (which has already been present in the professional market for some time) is the processing of High Definition material, since it demands large quantities of computation power, storage space and bandwidth. With the advancement of compression technology, highly compressed video for specific applications (such as videoconferencing), can provide much better image quality under the same bandwidth limitations and higher resolution, near or actually lossless, image quality is practicable.

All of this is possible because hardware constraints have been lifted significantly in the last years, enabling new techniques that were previously not viable. Also new methods to prepare data for compression and the compression methods themselves have been enhanced and tuned to perform as efficiently as possible. With larger storage available, it is possible to save much more information relating to the media contents; this can be seen on the newest digital video wrappers, where the gathering of metadata has taken a major role, much to the advantage of all spectres of video professionals, broadcasting, services or cinema.

To avoid repeating the design and implementation of the same media handling applications over and over again, multimedia frameworks provide for a platform where several essential tasks are already available and the introduction of new interoperable features is streamlined. This makes it possible to focus on the compression and the transforms themselves, rather than the structure used to employ them.

Many new innovative applications are being brought into both video codecs and multimedia frameworks such as the support for 3D animation, and compositing of complex interactions between media elements, objects and layers, of various kinds – video, audio or other.

Some concepts must be delivered before venturing into the thousand different video formats that are accessible in the market. First, there are two kinds of video formats: adapted image formats and native video formats. The first kind is mandatorily intra coded and consists of a series of addendums to an existing image compression format that maps how the different captured frames are stored and played back, and sometimes also how sound and other data can be stored and reproduced together. The second kind, the native video formats, is made expressly for video and is usually more efficient in lossy terms.

Single image support by these last formats may be possible using intra coded frames only, but it is not common to derive image compression from video formats, since temporal compression is much more emphasised than spatial compression. Interlaced video is only truly supported by native video formats; image formats sometimes offer workarounds that enable the coding of interleaved video.

An even less obvious distinction occurs between the compression formats themselves and the file structure they can be stored on: system layers or wrapper layers. Both layers contain information on how to decode and reproduce the video and associated media. The first is usually described as being part of the compression format itself, even though no compression purposes are achieved with it, and it is specific to that format and channels of transmission. Wrapper layers are much more flexible, capable of incorporating several different formats of video, audio and/or other media as well as different communication channels.

In order to better comprehend some of the technicalities associated to the handling of audiovisual material, be it on a computer or a dedicated hardware device, many of its important processing components will be explained in the following pages. This includes digital video compression, digital video storage and transport and software frameworks, which focus on allowing a smoother handling of a large array of multimedia formats.

2. Digital Video Compression

It is so natural today to turn on the television set and play a DVD, or watch our digital cable channels or go to the cinema and watch a (good) movie, that we seldom stop and think – how is it all possible? Well, it involves a lot of different expertise areas and technologies, some of which will not be discussed, but one thing that is common to all the above scenarios is the use of video compression. Without it, the costs of handling uncompressed media, both resource and financial wise, would be excessively big, and therefore digital video would not be available as widely nor be as sophisticated as it is today.

To delve into this topic, first, there will be an introduction of some basic ideas and concepts related to compression of digital data in general – specifically algorithms that are able to reduce the amount of any kind of data maintaining the same amount of information, lossless techniques.

Secondly, digital image compression will be focused on presenting a more specific set of processes aimed at separating image structure from plain image information, based on the handicaps of human perception, thus effectively compacting picture data. This separation consists in distinguishing between the data that is repeated from picture to picture (structure) and the data really signifies the picture (information); like separating a canvas from the ink itself that makes up a painting. Although lossless compression is also widely used, there is a preponderance of lossy formats due to bandwidth, storage and processing limitations.

Lastly, digital video compression will take the lessons learned on the first two sections and add the particular methods inherent to each of the studied video formats. Again, both lossless and lossy formats will be discussed with some relevance given to the latter. In the end, some tools will be shown that perform analysis on video files to determine if they are compliant with the standards or not.

In annex, section A.1. Video Compression Formats, there is a compilation of some of the available software tools that perform analysis on video files to determine if they are compliant with the standards or not and to track other errors.

2.1 Digital Data Compression

In order for transport and storage of information to be temporally and economically viable, it is necessary to reduce the size of the data involved. Compression is the act of removing redundancy in order to compact data, while trying to maintain the most information possible. There are two kinds of compression methods: lossless and lossy. Both rely on the fact that most information is not random: it follows some sort of pattern or order. By making use of the recognised patterns, that type of data can be efficiently compacted.

By using a lossless compression algorithm, it is assured that the information before compression will match exactly the decompressed result, as opposed to lossy compression where there is always some degree of degradation associated. Needless to say that the first type of compression yields smaller compression ratios (ratio of uncompressed by compressed volume) since it has to output, bit by bit, all the information provided.

Compression essentially takes advantage of the different types of redundancy in data:

- Temporal – insignificant changes from one time instant to the next;
- Spatial - correlation between neighbouring pixels or data items;
- Spectral - correlation between colour or luminescence components. This uses the frequency domain to exploit relationships between the frequency of change in data;
- Psycho-visual - exploit perceptual properties of the human sensorial system.

Some technical concepts will now be introduced that detail the more complex processes behind most coding algorithms used on video data. This information is not required to employ correctly the existing codecs but forms a more complete background to understand them.

2.1.1 Lossless Compression

This kind of compression works by removing any redundant information. Since it is limited by the need to output exactly the same data it received, it has the disadvantage, when compared with lossy compression, of achieving smaller compression ratios but its efficiency is less dependent on the input data.

2.1.1.1 Run-Length Encoding

RLE is a technique used to reduce the size of repeating strings of characters. A repeating string is called a run; typically, RLE encodes a run of symbols into as *counter / symbol*. It can compress any type of data regardless of its information content, but the data to be compressed affects the compression ratio. RLE cannot generally achieve high compression ratios compared to other compression methods, but it is easy to implement and quick to execute.

To better comprehend this method, let us examine the following exercise:

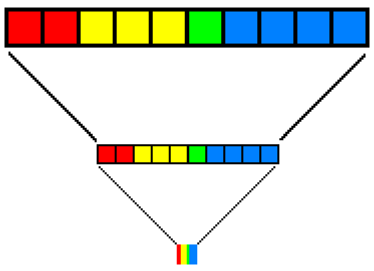


Figure 2.1.1 - 10x10 pixel example image, zoomed with added black lines for better legibility.

Take this sequence of ASCII characters (8 bits each) as a possible representation of the first 10-pixel line of the previous image as a source:

RRYYYGBBBB, `01010010 01010010 01011001 01011001 01011001 01000111 01000010 01000010 01000010 01000010`

It amounts to 10 bytes, 80 bits. If we were to code it using RLE, it would compress in the form of counter/symbol:

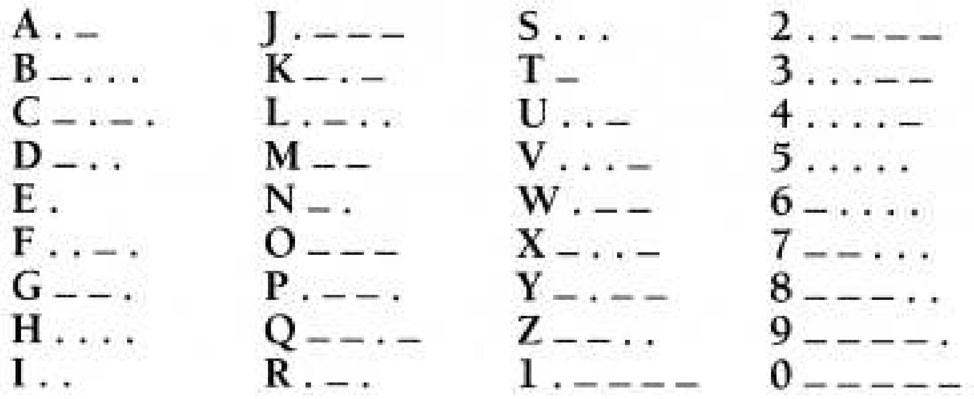
2R3Y1G4B, `00110010 01010010 00110011 01011001 00110001 01000111 00110100 01000010`

The result amounts to 8 bytes, thus a compression ratio of 1,25:1 was achieved. This simple example shows that this kind of algorithm is only efficient if the source data contains long runs of the same symbol since when there are many single symbols within the source, this method might even had output near double the original size.

2.1.1.2 Entropy Encoding

This kind of compression is usually the last step of compression since it is much more effective after all the lossy techniques are used, furthering the compression ratio. It compresses the data by substituting the most frequent symbols by shorter ones and least frequent symbols by slightly longer ones; it is therefore a statistical method. This is only possible if, when sending a coded symbol we have some way of knowing where it ends, enabling symbols with different lengths.

A well-known example of entropy encoding is the Morse code. The dots and dashes that compose a symbol (like a letter for instance) were chosen based on the frequency with which it appears in the English language. Within the code table of Morse code an "E", the most used letter in the English alphabet, is encoded as a single "dot", whereas a "Q" is coded with the sequence "dash dash dot dash". In order to choose which symbols to use, it is necessary for both sender and receiver, to know beforehand the code table. This table can only be produced effectively from a significant sample of the data to be transmitted. If we were to send a Portuguese text through Morse code the compression ratio would be much less than that of an English text. One alternative to redoing the table for each different message is to use fixed tables known by both parties beforehand and send its reference in the beginning of the transmission.



A .–	J .–––	S ...	2 ..–––
B –...	K –.–	T –	3 ...––
C –.–.	L .–..	U ..–	4 ....–
D –..	M ––	V ...–	5 .....
E .	N –.	W .––	6 –....
F ..–.	O –––	X –..–	7 ––...
G ––.	P .––.	Y –.––	8 –––..
H	Q ––.–	Z ––..	9 ––––.
I ..	R .–.	1 .––––	0 –––––

Figure 2.1.2 - Morse code symbol / codeword table

The term entropy is the measure of uncertainty associated with a random variable. It quantifies the information contained in a message, usually in bits or bits per symbol. It is the minimum average symbol length necessary to communicate information.

As a curiosity, the previously used example (Figure 2.1.1 - 10x10 pixel example image, zoomed with added black lines for better legibility.) source would encode in Morse code as ".- . .-. -.-- -.-- -.-- --. -… -… -… -…" that uses 46 bits or roughly 6 bytes, resulting in a compression ratio of 1,74:1. All that was carried out was to encode the letters into Morse code and then substitute the utilised symbols, from dots and dashes to binary (take into consideration that spaces between codes and the end of file (EOF) marker were not accounted for). [Sym04]

Huffman Coding

This entropy compression method is based on symbol probability to construct a binary tree to code the symbols, and it is the most used compression technique today. It involves two phases: source reduction followed by codeword construction.

First, the two least probable symbols are combined, shortening the alphabet. The combined symbol has the summed probability of both its components. This process is repeated until only two symbols are present. The next phase, codeword construction, then begins by attributing either one of the binary symbols (0 by default) to the most probable symbol and the complementary to the least probable one, for each level, by separating the symbols generated in the previous stage. In case of equal probability, either way is allowed, thus introducing different possible trees, depending on the implementation. In the end, there must be a unique code to each symbol.

Decoding is done by parsing the given tree until it hits a leaf, which means it has found a value, and repeating the process until finding an EOF symbol, or reaching a given length.

Using the same source and assuming the same distribution from the previous example, a straightforward Huffman encoding process will be demonstrated.

	Step 0				**Step 1**			**Step 2**	
Symbol	B	Y	R	G	B	Y	RG	YRG	B
Probability	0,4	0,3	0,2	0,1	0,4	0,3	0,3	0,6	0,4

Table 2.1.1 - Huffman's encoding source reduction phase.

As explained before, each step adds the two least probable symbols and constructs a new one.

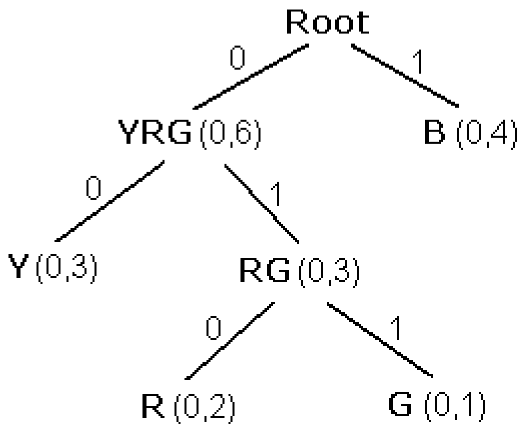


Figure 2.1.3 - Huffman binary codeword tree.

Codeword construction, the reverse process of source reduction, starts attributing the codewords as it goes, starting at the highest probability symbols.

If we search all the symbols generated by the tree we get:

Symbol	**B**	**Y**	**R**	**G**
Codeword	1	00	010	011

Table 2.1.2 - Huffman generated codewords.

Which applied to our source RRYYYGBBBB originates the binary:

`010 010 00 00 00 011 1 1 1 1` (3 bytes)

Coding has been done with a compression ratio of 3,33:1, a clear improvement over RLE. This level of compression was possible because the assumed probabilities were perfect since they exactly match our source; on real life, practically only approximations are available, hence these ratios are simply demonstrative. When using an approximation to the symbol probabilities, results tend to decay. This is true to all compression methods based on symbol probability.

It can be shown mathematically that it is not possible to generate a code that is uniquely decidable (each code corresponds exactly to one symbol) and more efficient than Huffman [Abr63].

This and all following compression examples are simplified due to the reduced dimension of the data, by the lack of headers, which carry information needed for the decoding to be executed, and, in some cases, the lack of an EOF symbol, or a length value, that allows the decoder to determine when it should stop parsing incoming data. On the other hand if the source message was larger, which it usually is, the compression achieved would be the same or even superior in view of the fact that the header and footer transmission size will become irrelevant, relatively to the size of the data itself.

Arithmetic Coding

Contrary to Huffman encoding, this method codes the source into a single real number, a fraction between 0 and 1, instead of several codes. This fraction is coded in binary form or on a base equal to the number of symbols, if they are few.

Encoders start by determining a model for the source, which is a prediction of the symbols' probabilities. Having set on a model, a range from between [0 - 1)[1] is attributed to each symbol according to its probability. Then the coded number is recurrently refined by narrowing the range it is on according to the range of the symbol being coded. Decoding is done by the reverse process.

This methodology is not easily understood without an example, which follows once more with the same source from the previous ones. Setting a model:

Symbol	**B**	**Y**	**R**	**G**
Range	[0 – 0,4)	[0,4 – 0,7)	[0,7 – 0,9)	[0,9 – 1]

1 The square brackets "[ ]" denote that the number is included, opposed to parentheses "( )" that indicate the number is excluded of the range.

Table 2.1.3 - Source symbols' ranges.

Symbol	Ranges				
R	0	0.4	**0.7**	**0.9**	1
R	0.7	0.78	**0.84**	**0.88**	0.9
Y	0.84	**0.856**	**0.868**	0.876	0.88
Y	0.856	**0.8608**	**0.8644**	0.8668	0.868
Y	0.8608	**0.86224**	**0.86332**	0.86404	0.8644
G	0.86224	0.862672	0.862996	**0.863212**	**0.86332**
B	**0.863212**	**0.8632552**	0.8632876	0.8633092	0.86332
B	**0.863212**	**0.86322928**	0.86324224	0.86325088	0.8632552
B	**0.863212**	**0.863218912**	0.863224096	0.863227552	0.86322928
B	**0.863212**	**0.8632147648**	0.8632168384	0.8632182208	0.863218912

Table 2.1.4 – Calculation of the fraction codeword.

On each line, the next range is defined by the symbol being coded, as it is made evident by the bold highlighting used.

Any value between the last two bold values [0.863212 – 0.8632147648) will decode into the same result, which means that we still have some redundancy to eliminate. In this case, it is because decimal fractions were used for enhanced clarity of the example. If in turn, base four was used (one for each symbol), the result would have spawned much less redundancy.

863212 (6 bytes), all data is sent as ASCII and discarding the assumed "0,"

```
00111000 00110110 00110011 00110010 00110001 00110010
```

86 32 12 (3 bytes), if each number is coded as a Binary Coded Decimal in 4 bits;

```
10000110 00110010 00010010
```

Decoding is done in a very undemanding way: given the intervals, the decoder simply verifies in which range the fraction fits. It should be noted that we do not require one codeword for each symbols because the same number can "fit" in several consecutive intervals.

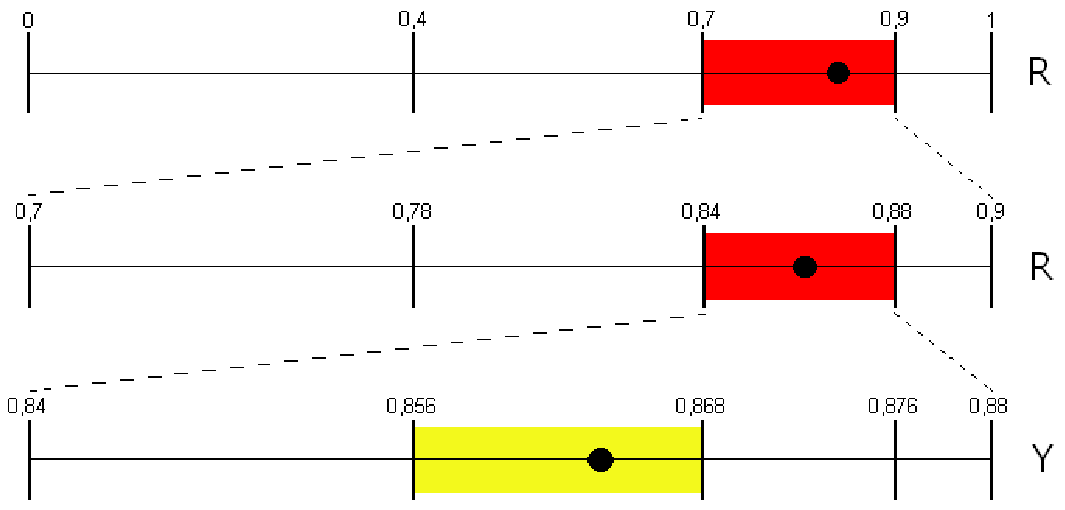


Figure 2.1.4 - The first three steps of decoding the value 0,863212.

Again the closer the probabilities are to the source, the more effective the compression will be. We have reached the same compression ratio as with Huffman code (3,33:1), even though we used a decimal base, instead of binary or base 4 for the fraction. Typically, arithmetic encoding performs better than the previously studied methods, since it does not require one codeword for each symbol.

2.1.2 Lossy Compression

This kind of algorithms can only be used when the lost information is not absolutely needed, often in situations relying on exploring the limitations inherent to the human senses. Thus, it is used on image, video and audio compression, since the human eyes and ears do not need all the information available on a given channel to provide the viewer or listener with a coherent and clear image, video or sound. For example the human hear can perceive frequencies ranging from 20 Hz - 20 kHz and can sense sounds from 0 to 130 dB; all of which and more has been studied and compiled into psychoacoustic models, studies of the psychological correlation of subjective human perception of sounds to the physical parameters of acoustics. Also the human eyes are not very sensitive to changes in colour, whereas changes in brightness are easily spotted, making it possible to code colour with less information, but still maintaining good viewing results.

The problem associated with lossy compression is the introduction of compression artefacts, either when the available data rate is too low, discarding important information, or when the algorithm assumes some data is not relevant, whilst it is in fact, objectionable to the user. These artefacts are compression format specific, although some are common to similar compression methods. It should be noted that most lossy codecs for multimedia usually have a lossless encoder portion to encode the integer values of the quantized prediction or transform operators.

Each lossy compression technique is adapted to the specific kind of data being compressed, since the errors introduced must not be too noticeable on the compressed material. For this reason, lossy compression will be further studied with image and video compression on chapters 2 and 3 correspondently.

2.1.2.1 Markov Sources

A Markov source is a statistical model for predicting the occurrence frequencies of letter or word pairs and triplets, meaning that each symbol is dependant from one (first level Markov source) or *n* (n^{th} level Markov source) previous symbols. One simple example of a Markov source is any language, like English for instance. In this language after the letter “q” there is a very high probability of the letter “u” to follow, and much less probably the letter “y”. Considering “qu”, it is almost impossible not to be followed by a vowel.

Other sources with memory, where a value depends on the previous one(s), include images, video and audio; such characteristic is one of the cornerstones of all lossy compression algorithms. By considering a Markov source, information can be separated from its structure (de-correlated), by taking advantage of the different kinds of redundancy. This way an efficient algorithm can be devised to compress a particular type of source data accurately, by only storing or transmitting (nearly) plain information.

2.1.3 Transform Coding

A transform is a mathematical rule that consists in converting one set of values into a different one. This principle is used before compression in order to enhance it, since it arranges information in a particular redundant fashion. It is used on both image and video compression, and on some other fields besides compression, such as in signal analysis.

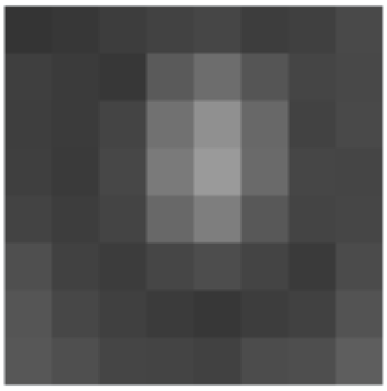
First, a transform is applied to the source data. This produces as many coefficients as there are elements. Then, these coefficients can be more easily compressed because the information is statistically concentrated in just a few coefficients. This, of course, is very data specific; whereas a certain transform might perform well on image data, that same transform can even worsen the compression of sound data, for instance.

Although associated with lossy compression methods, transforms are indeed lossless processes, provided that there is enough resolution when calculating coefficients and when decoding them back to the original format.

A thorough mathematical analysis is beyond the scope of this document but a few key points should be demonstrated in order to better understand why this technique is one of the most used in compression algorithms.

2.1.3.1 Discrete Cosine Transform

DCT transform is associated with lossy compression, although the process is inherently lossless – for the recovery of the exact same data, only enough arithmetic precision is required. DCT is a modified version of the Discrete Fourier Transform that uses real instead of complex numbers and handles only cosines and not sines. DCTs express a signal in terms of a sum of sinusoids with different frequencies and amplitudes, on a finite number of discrete data points.
Let us apply the transform to a simple example of a greyscale sub-image block. DCT can be employed on any rectangular array of pixels; in image compression a block of 8x8 was the most commonly used, since it provided a good relation between coding complexity and compression scope. Today 16x16 is increasingly used, possible by the advances in hardware and software performance.



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}$$

Figure 2.1.5 – Square image block and correspondent 8-bit greyscale values matrix.

The first step involves centring all the values on zero, which is done by subtracting half the number of possible values. In this case, considering an 8-bit greyscale, there are $2^8 = 256$ possible values; hence, we subtract 256 / 2 = 128.

$$
\begin{array}{c}
\xrightarrow{x} \\
\begin{bmatrix}
-76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\
-65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\
-66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\
-65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\
-61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\
-49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\
-43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\
-41 & -49 & -59 & -60 & -63 & -52 & -50 & -34
\end{bmatrix}
\end{array}
\Bigg\downarrow y
$$

Figure 2.1.6 – Matrix of the greyscale values minus 128.

The matrix can now be submitted to the transform and then rounded to the nearest integer. Several different DCTs exist, being the most frequently used the type II and for reverting, its inverse the type III. DCT-II's mathematical equation is provided here solely as a reference.

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

Equation 2.1.1 – Mathematical expression of DCT-II.

$$
\begin{array}{c}
\xrightarrow{u} \\
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2
\end{bmatrix}
\end{array}
\Bigg\downarrow v
$$

Figure 2.1.7 – Final matrix.

AC coefficient is the nomenclature given to all values except the one on the top left corner, which is called DC coefficient and is always the greatest one (AC and DC names come from AC/DC currents). The advantage of the DCT is its property of aggregating most of the signal in one corner of the resulting matrix, as may be seen in Figure 1.7. The quantization step that follows on compression formats, such as JPEG and others, accentuates this effect while simultaneously reducing the size of the DCT coefficients, resulting in a signal with a large trailing region containing zeros that the entropy compression stage can simply throw away (not the entropy compression itself but an associated process). Note that the formerly used 8-bit resolution gave place to 9-bit to accommodate the DC coefficient. Even though this is a step back in compression terms, the next compression phase will return the matrix to the initial 8-bit form.

It is computationally more efficient and easier to implement, to regard the DCT as a set of basis functions, which given a known input array size can be pre-computed and stored. It only requires computing values with an easily applied mathematical operator - a convolution mask.

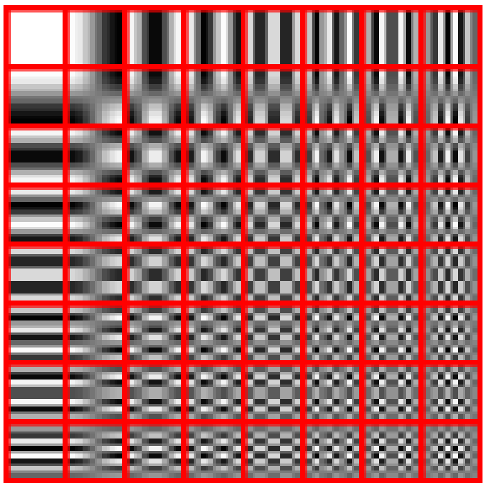


Figure 2.1.8 – Two dimensional DCT basis functions.

2.1.3.2 Wavelet Transform

A wavelet is a kind of mathematical operation, used to divide a given function or continuous-time signal into different frequency components and study each component with a resolution that matches its scale. A wavelet transform is the representation of a function by wavelets. The wavelets consist of scaled and translated copies (known as “daughter wavelets”) of a finite-length or fast-decaying oscillating waveform (known as the “mother wavelet”).

Wavelet transforms can be classified into discrete (DWTs) and continuous (CWTs) wavelet transforms. Note that both are continuous-time transforms. They can be used to represent continuous-time (analogue) signals. CWTs operate over every possible scale and translation, whereas DWTs use a specific subset of scale and translation values or a representation grid.

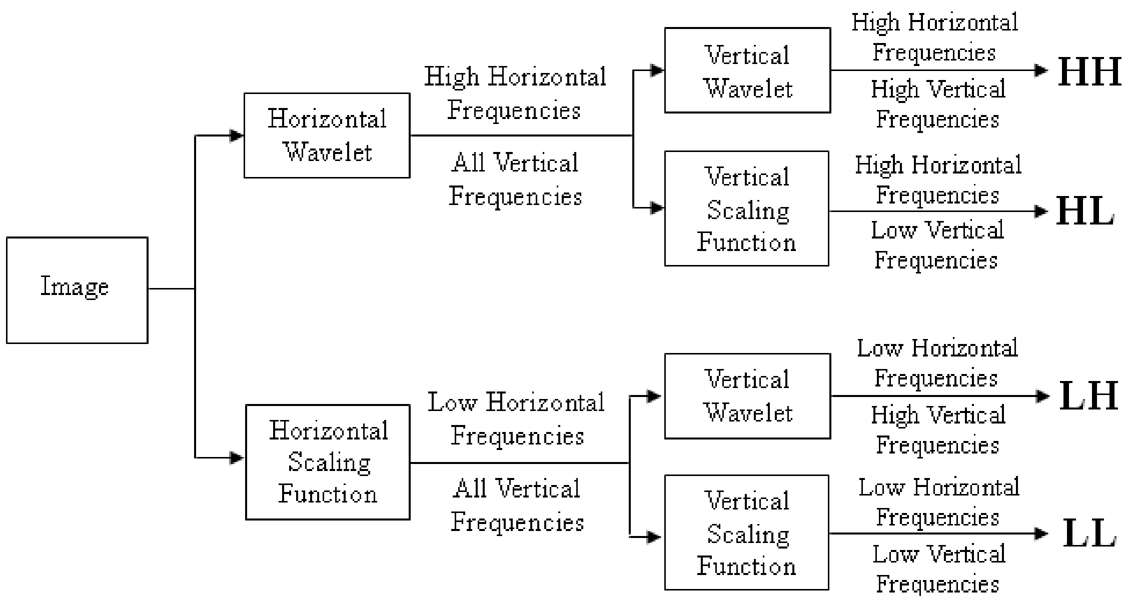


Image
Horizontal Wavelet
High Horizontal Frequencies
All Vertical Frequencies
Vertical Wavelet
High Horizontal Frequencies
High Vertical Frequencies
HH
Vertical Scaling Function
High Horizontal Frequencies
Low Vertical Frequencies
HL
Horizontal Scaling Function
Low Horizontal Frequencies
All Vertical Frequencies
Vertical Wavelet
Low Horizontal Frequencies
High Vertical Frequencies
LH
Vertical Scaling Function
Low Horizontal Frequencies
Low Vertical Frequencies
LL

Figure 2.1.9 – Division of an image by wavelet convolution. [Sym04]

Wavelets act as filters, and may be used multiple times in a row. besides the wavelet itself there is its associated scaling function. They are both employed in the vertical and horizontal directions, so each pass separates the image into four parts according to their frequencies.

The advantage of using wavelet-based coding in image compression is that it provides significant improvements in picture quality at higher compression ratios, over previous techniques. Since wavelet transforms have the ability to decompose complex information and patterns into elementary forms, it is also used in acoustics processing and pattern recognition.

Wavelet transforms have advantages over traditional Fourier transforms (like DCT), by being able to represent functions that have discontinuities and sharp peaks, and for accurately deconstructing and reconstructing finite, non-periodic and/or non-stationary signals. As the previously studied transforms, it changes the data distribution into a more suitable form for enhanced compression, in this case making it more redundant.

A feature that is enabled by using this kind of transform is the possibility of encoding data on several layers with increasing detail so that a gradual transmission can be done: first, a low quality image is sent, following the rest of the picture in quality increments over time.

Some wavelet implementations can be made very efficient, processing wise, by using Fast Fourier Transform (FFT) a known and proven algorithm used for digital signal processing, solving partial differential equations and for quick multiplication of large integers.

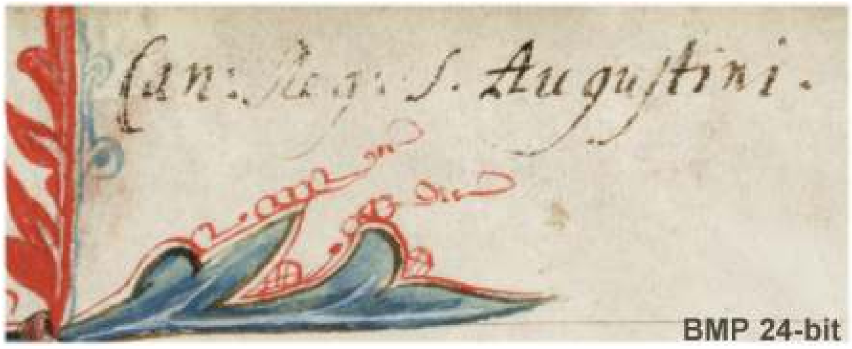


Figure 2.1.10 – Original image in a lossless format. [Kno00]

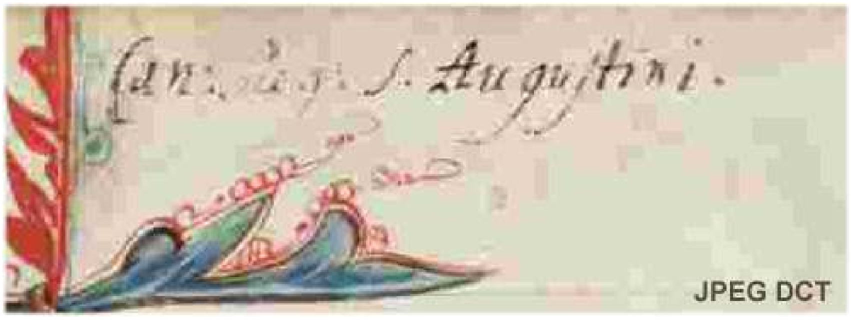


Figure 2.1.11 – Heavily compressed image (60:1) with a DCT based compression format. Objectionable blocking artefacts are clearly visible. [Kno00]

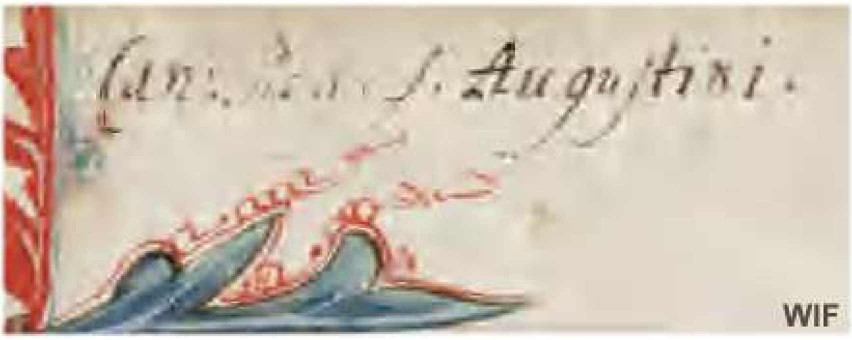


Figure 2.1.12 – Image compressed to the same ratio, encoded with a wavelet based format. Less disturbing artefacts of smearing of detail are noticeable. [Kno00]

2.2 Digital Image Compression

There are two types of images, each with its own radically different compression methods: natural and computer generated (CG). The first kind, focused on this document, that includes digital photographs for instance, exists in continuous tone and will be the target of this study. Opposed to it, CG imagery is usually represented in halftone, to simulate continuous tone.

2.2.1 Sampling

This process converts a continuous signal into a discrete signal, for example an analogue signal to digital form. It does so by taking measurements of the amplitude of a varying waveform at regular intervals. The variations might be temporal, spatial or on any other dimension. Reconstruction of the signal is possible to the degree of precision given by the sampling frequency (the number of samples obtained during a second).

If the sampling is done on already digital media, it is called resampling. Resampling is the digital process of changing the sample rate or dimensions of digital imagery or audio by temporally or spatially analysing and sampling the original data.

If samples are taken with very long intervals between them, aliasing might be introduced. This happens since it is not known what values exist between long measurements, leading to an erroneous reconstruction of the original signal. Therefore, to guarantee that a faithful reconstruction to the selected level of detail is possible, the sampling frequency must be more than twice its maximum frequency, according to the Nyquist–Shannon sampling theorem.

An example of temporal aliasing is a fast spinning wheel of a moving car in a movie: since the frequency of the rotation can easily surpass the sampling of the video camera (even at 60 frames per second) the illusion that the wheel is moving slowly or even backwards is created. This means that not only high frequency data is being skipped but also erroneous information is being introduced.

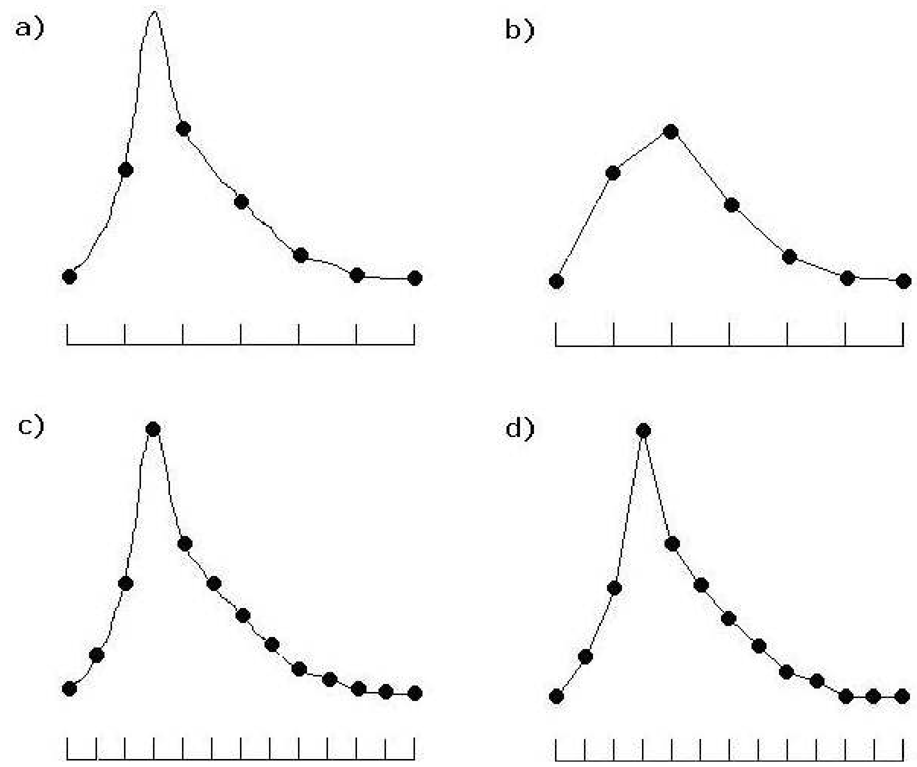


Figure 2.2.13 – Sampling examples. Original signal on the left; reconstructed signal on the right at low (top) and high sampling rate (bottom). Only d) faithfully rebuilds the source.

2.2.2 Quantization

This process limits the value of a function at any sample, to one of a predetermined number of permissible values. It can be performed before or after sampling, but it is usually done afterwards. Quantization is a form of lossy compression, and as such, information will be lost: the quantification process is not reversible. As before, care must be taken when selecting the quantization ranges because too much detail will be lost if the intervals are too large and if the intervals are too short excessive amounts of data might be generated for the same amount of information.

This step is where a big part of the compression occurs, paired with chroma subsampling and to a minor extent sampling and image prediction, which is not always used.

Colour quantization reduces the number of colours used in an image by attributing each pixel colour value to a smaller set of colours with less bits; this is important for displaying images on devices that support a limited number of colours and for efficiently compressing certain kinds of images. It is common to combine colour quantization with dithering to create an impression of a larger number of colours and eliminate banding artefacts. Dithering makes up for a poorer colour palette by combining pixel alternately between two or more colours, which are interpreted by our eyes as nearly the original colour.

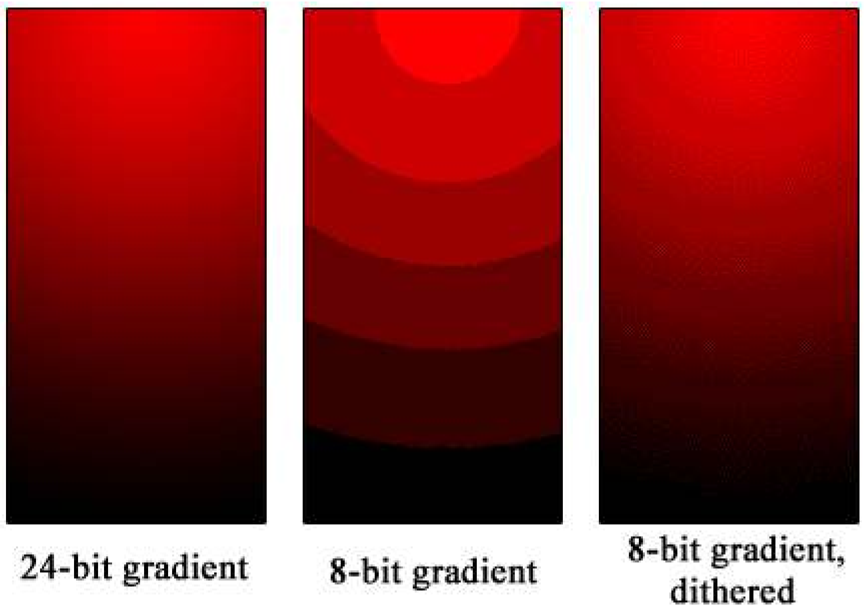


Figure 2.2.14 – Original 24-bit colour gradient, quantized to 8 bit with and without dithering.

There are two types of quantization: scalar, used when one-dimensional variable needs to be quantized and vector based, used to quantize several or multi-dimensional variables. Quantizers may also be either uniform or not, depending on whether the intervals are all of the same size or not.

The human eye is good at distinguishing small differences in brightness over a relatively large area, but not so good at determining the exact intensity of high frequency brightness variations. This fact allows one to get away with a greatly reduced amount of information in the high frequency components. This is done by simply dividing each component in the frequency domain by a constant, and then rounding to the nearest integer. This is the main lossy operation in the whole compression process. As a result, it is typically the case that many of the higher frequency components are rounded to zero and many of the other components become small absolute values.

Picking up on the DCT's example image matrix after applying the type II matrix,

$$\begin{array}{cc} \overset{u}{\longrightarrow} & \\ \begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} & \Big\downarrow v \end{array}$$

Figure 2.2.15 – Post DCT matrix.

And using a common image quantization matrix,

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 2.2.16 – General quantization matrix.

The resulting matrix, of dividing each element on the post DCT matrix (Figure 2.2.15 – Post DCT matrix.) by the correspondent ones on the general quantization matrix (Figure 2.2.16 – General quantization matrix.is achieved:

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -3 & 4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2.2.17 – Final matrix.

The resulting values are concentrated in the upper left corner and are all very close to zero making it suitable for being efficiently condensed by any appropriate compression algorithm, such as run-length or entropy encoding.

2.2.3 Predicting Image Values

As an alternative to coding the intensity of a value, we can predict it by knowing the values of the intensity of the nearby pixels. This way all we need to transmit or store is the difference between the value and our prediction, taking advantage of the spatial redundancies inherent to image data and the temporal redundancies of video data. Any number of neighbouring pixels may be used but the more are employed, the better the results, but also the more processing resources and complexity is needed, which can be wasteful. Tests have been made by Habibi in 1971 [Hab71], which demonstrate that beyond three prediction pixels, the increase on the number of elements evaluated for the prediction, provides insignificant improvements in compression and image quality.

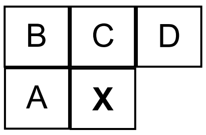


Figure 2.2.18 – Prediction pixels positions.

The most used predictors are, in increasing compression performance:

$$\mathbf{X} = A$$
$$\mathbf{X} = 0.5\ A + 0.5\ C$$
$$\mathbf{X} = A - B + C$$
$$\mathbf{X} = 0.75\ A - 0.5\ B + 0.75\ C$$

$$\mathbf{X} = \begin{cases} \min(A,C) & \text{if } B >= \max(A,C) \\ \max(A,C) & \text{if } B <= \min(A,C) \\ A - B + C & \text{otherwise} \end{cases}$$

Some newer codecs extend this technique with temporal redundancy, adding a third dimension into the equations.

Note that the last two predictors perform with very similar results, although the last one, called adaptive, only needs to process integer values and simple operations, offering a faster processing, on both encoding and decoding.

2.2.4 Chroma Subsampling

Compression wise, it is possible to take advantage of the human eye's inability to detect small differences between colours, by focusing on the amount of pixel information, not on colour, but on brightness, property to which the eye is more attuned. With this process, part of the colour data is simply discarded.

No existing colour space perfectly matches the range of colour information a human is capable of distinguishing. Some contain most of the spectre, others even enclose areas of light we cannot see; still RGB is considered lossless although it does not fully incorporates our visible colour scope. A common way of displaying our range of visible colours is the horseshoe diagram.

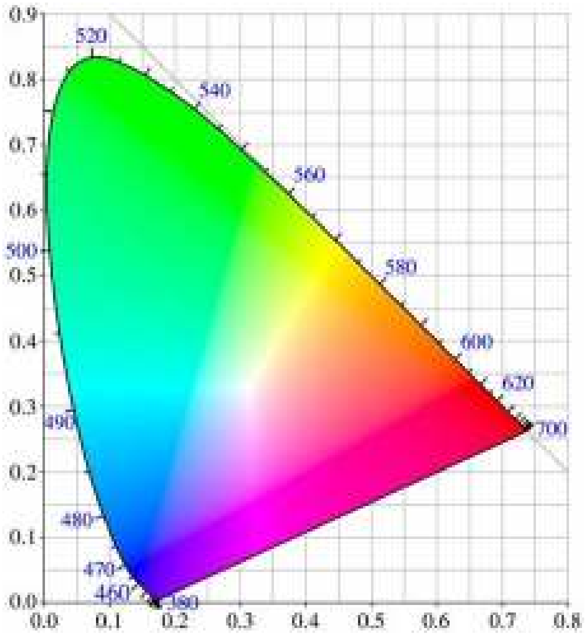


Figure 2.2.19 – Horseshoe diagram of the gamut of humanly visible light.

There are different types of colour spaces, being a full colour system known as 4:4:4, such as the RGB and a compressed one the 4:2:2 YCbCr for instance, which is a digital version of the analogue YUV. The three components of YCbCr stand for luma (brightness), blue chrominance (or more precisely the colour deviation from gray on a blue-yellow axis) and red chrominance (or more precisely the colour deviation from gray on a red-cyan axis). One of most commonly used high quality subsampling is 4:2:2, which effectively halves both chroma components horizontally. This specific subsampling reduces the bandwidth of a video signal by one-third with little to no perceivable visual difference.

The numbers that denominate a colour system refer to the relationship between the frequencies at which each component is sampled. 4:4:4, for instance, has all components sampled at the same frequency, whereas 4:2:2 both chroma values are sampled at half the horizontal resolution of the luma.

This measure of colour faithfulness is one of the differentiation factors between consumer and professional market: the movie industry uses colour samplings as close as possible to 4:4:4 whereas television utilises lesser colour resolutions and consumers even less. With the rising availability and lowering prices of memory supports, increasingly higher samplings are being used with each new technological iteration.

The following images display several diverse macroblock arrangements corresponding to different colour spaces.

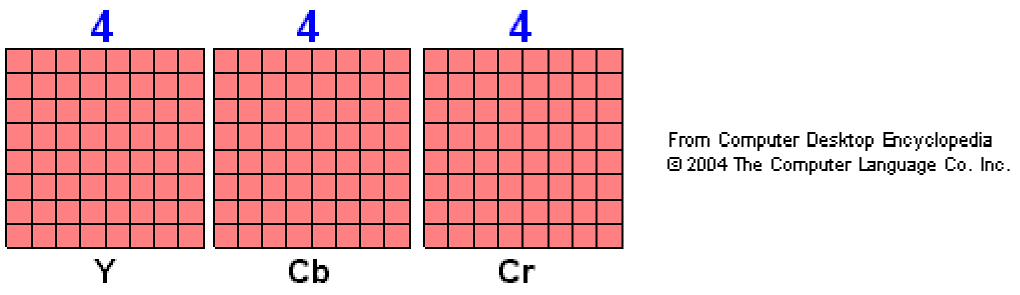

4
4
4
Y
Cb
Cr
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Figure 2.2.20 – 4:4:4 Chroma subsampling; Cb and Cr are sampled at the same resolution as luma. It is rarely used, except for very demanding high image quality requirements.

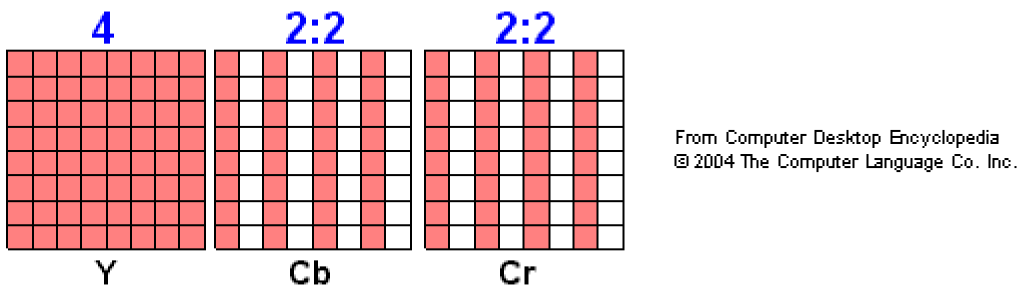

4
2:2
2:2
Y
Cb
Cr
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Figure 2.2.21 – 4:2:2 co-sited, meaning all components sampled at the same instant; Cb and Cr are sampled at half the horizontal resolution of luma.

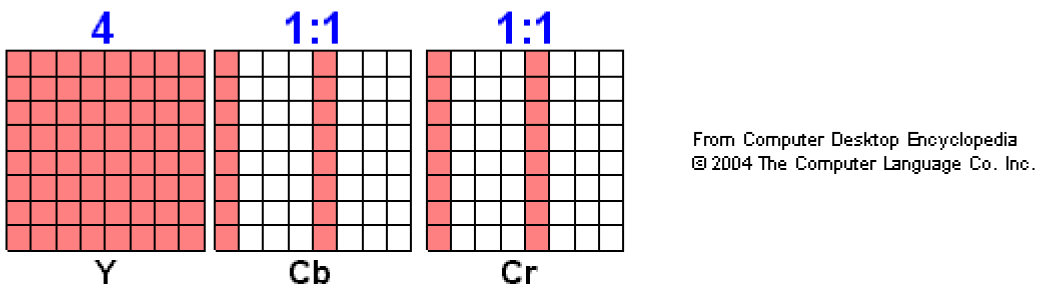

4
1:1
1:1
Y
Cb
Cr
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Figure 2.2.22 – 4:1:1 co-sited; Cb and Cr are sampled at quarter the horizontal resolution of luma.

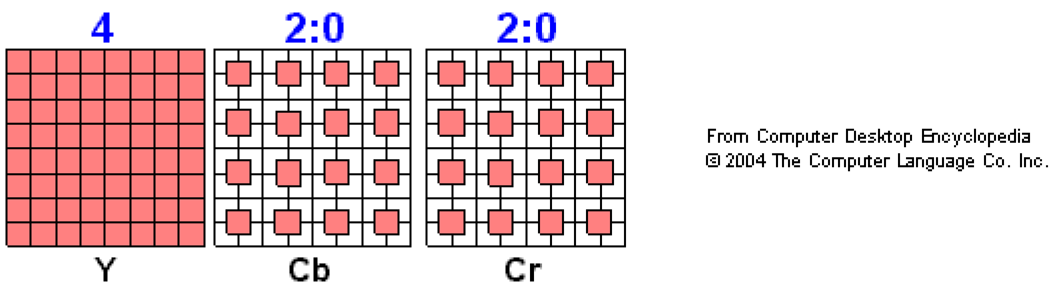

4
2:0
2:0
Y
Cb
Cr
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Figure 2.2.23 – MPEG-1 4:2:0 Chroma subsampling; Cb and Cr are sampled at half the resolution of luma both horizontally and vertically.

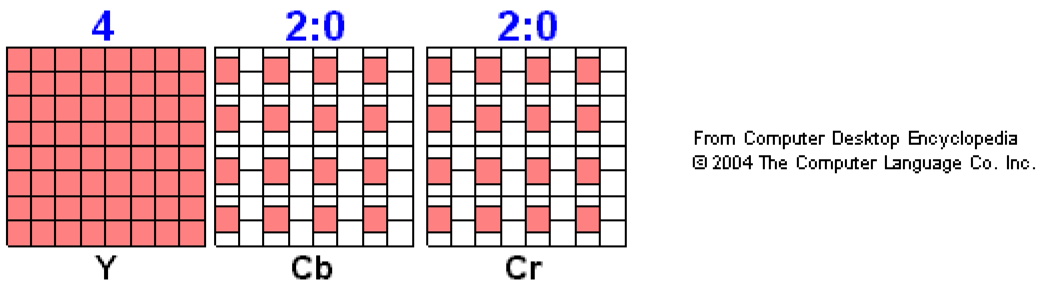

4
2:0
2:0
Y
Cb
Cr
From Computer Desktop Encyclopedia
© 2004 The Computer Language Co. Inc.

Figure 2.2.24 – MPEG-2 4:2:0 Chroma subsampling, same properties as MPEG-1's 4:2:0, except that samples are taken at different intervals, making it visually similar to 4:2:2.

2.3 Digital Video Compression

As in other kinds of compression, video compression aims to reduce the amount of data necessary to store and transmit video media. It is a combination of colour subsampling (on lossy codecs), image compression and motion compensation (which introduces a third dimension - time). Digital video is divided in discrete moments in time, corresponding to still images, called frames. If the source is interlaced however, each frame corresponds to two fields each relating to the even and odd lines of the picture. As digital storage capabilities increase, more lossless formats are emerging, mainly for professional applications.

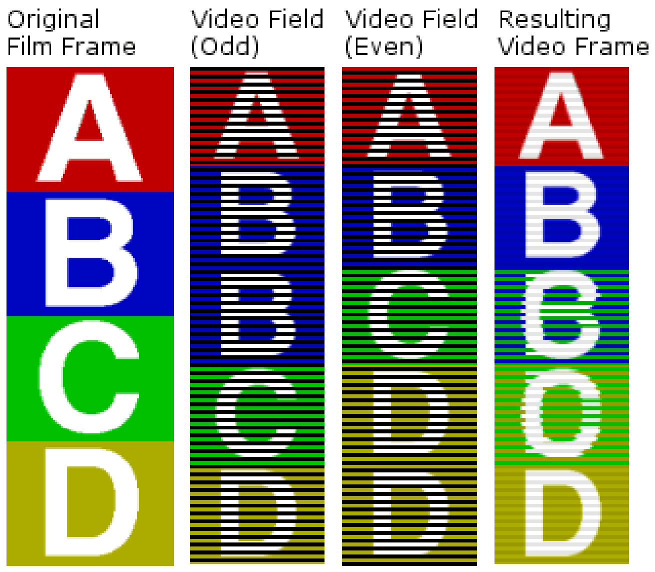


Figure 2.3.25 – Difference between progressive (first column) and interlaced frames (fields). The overlapping effect on the resulting frames can be minimised with deinterlacing methods.

2.3.1 Motion Estimation and Compensation

The human eye / brain ensemble can sample the world at a rate higher than 200 times per second (!) and is also remarkable at compensating the lack of input in between two or more samples. This flexibility makes motion estimation possible by representing some frames with less information than others, permitting only to encode the differences between a certain number of frames. In addition, videos typically have the same objects across several frames, either moving or stationary, so motion estimation can be performed, resulting in not coding what objects or parts of each frame have, but the way they relate from one to the other. This is possible because of the wide existence of temporal redundancy in video media, due to the high correlation between consecutive frames, especially on slow-paced scenes.

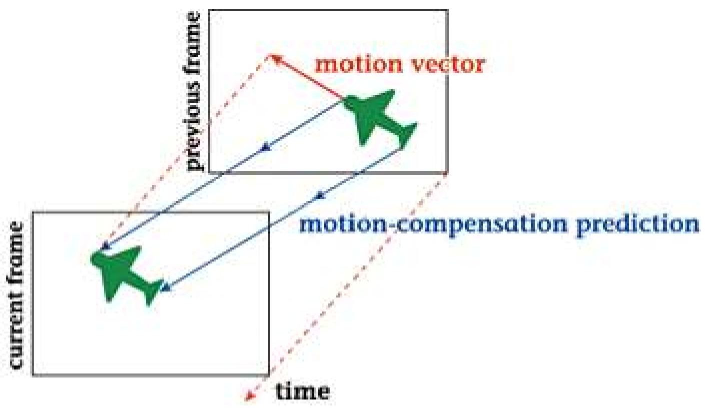


Figure 2.3.26 – Simple example of motion estimation and compensation process.

Motion estimation consists of, based on a reference image, instead of coding a second picture, to encode the differences between the two by generating motion vectors for image blocks that represent the displacement of a sub-image from a frame to another. For example, in the frames of a camera panning through a still background, this back scene maintains parts of the initial frame on the following ones, just on different positions. This can be efficiently encoded by using motion vectors. These are later reconverted into images by the decoder in a process called motion compensation, using the same reference(s) frame(s).

2.3.1.1 Frame Types

Three kinds of frames exist according to how they are decoded, more precisely according to from which frames they are derived. The use of different kinds of frames in lossy compression allows saving space by representing some frames with less information and detail, but still maintaining a certain level of quality. In lossless compression, only one type of frame is used. In some video formats (like MPEG-2), a repeating series of different frames is called group of pictures (GOP). Some details of the frame's specification diverge on different compression formats.

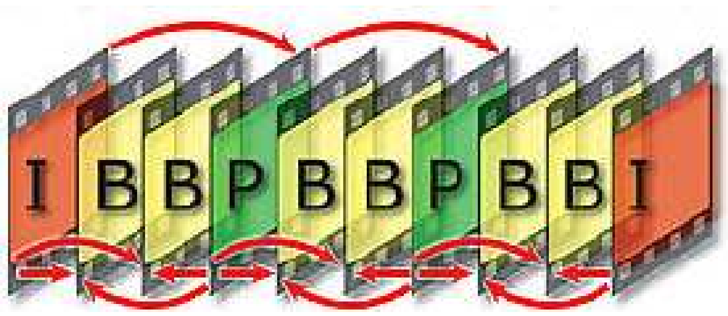


Figure 2.3.27 – Simple GOP; red arrows indicate the reference frame(s) used for decoding.

Intra Frames

Also called key or I frames, they are not decoded in reference to other frames, except themselves.

They are often used as random access points every preset number of frames and exceptionally when the complexity of a scene requires more information. Its placement interval is often fixed but it can be increased or decreased in order to respect bandwidth limitations or quality requirements.

I-frames usually require the most bits to encode and are the only type of frames used in lossless compression.

Predicted Frames

P-frames or slices are decoded in reference to previously decoded frames and may contain image data, motion vector displacements or combinations of the two.

In older standard designs (such as MPEG-2), only one previously decoded I-frame is used as a reference during decoding, and requires that picture to also precede the P picture in display order. In H.264, multiple previously decoded pictures can be used as references during decoding, and they can have any arbitrary display order relationship relative to the pictures used for its prediction.

Typically require fewer bits for encoding than I pictures do.

Bi-directional Frames

B-frames, similarly to P-frames, may contain image data, motion vector displacements or a combination of the two.

To enable B-frames it is necessary to change the order at which frames are sent or stored, differently from the order of display, because it may be decoded using frames both before and ahead of itself, of all I, P and B frame types.

In older standard designs (such as MPEG-2), B pictures are never used as references for the prediction of other pictures. Besides, it uses exactly two previously decoded pictures as references during decoding, and requires one of those pictures to precede the B picture in display order and the other one to follow it.

In newer codecs, they may or may not be used as references for the decoding of other pictures (at the discretion of the encoder). In H.264, these frames can use one, two, or more than two previously decoded pictures as references during decoding, and can have any arbitrary display-order relationship relative to the pictures used for its prediction.

Typically require fewer bits for encoding than either I- or P-pictures do, since its references can come from ahead or before it that makes it use more already existing sections, but requires much more complexity.

2.3.2 Video Compression Formats

As it was seen in digital data compression, video compression formats can be either lossless or lossy; the first kind reproduces each pixel exactly like the source, thus being targeted at higher quality requirements markets, like cinema and television; while the second introduces loss of quality and is used on all video markets.

There are plenty of video formats, each of which has different implementations, called codecs that in turn have different versions. Only the most relevant variants used today, in the professional market, will be described. Some formats support either lossless or lossy compression, by altering a number of their settings.

2.3.2.1 Lossless Formats

Usually achieve smaller compression ratios than lossy formats, as a result of having to reproduce the source bit by bit, these compression formats typically do not make use of predicted or bi-directional frames using intra frames exclusively.

The size of an average length film coded in a lossless format can easily reach several tens of Gigabytes while a lossy version of it with good visual quality could take as little as a couple of Gigabytes, thus making it much more interesting for medium to low end users. Compression ratios generally achieve about 20:1 lossy and approximately 3:1 lossless.

Digital moving-Picture eXchange

This lossless video file format, represented by the extension .dpx, is used for the exchange of resolution-independent, pixel-based (bitmapped) images, and intended for very high quality moving image content for theatrical distribution. DPX is, in fact, the Kodak Cineon raster file format with just a few slight modifications to the file's header. As the reader might have deducted, this is an image format extended to support video. It is an open standard developed by the Society of Motion Picture and Television Engineers (SMPTE).

It is typically used as a middle-state format for material exchange, post-production in movie industry parlance, or archiving; in some circumstances it may be used as an initial state format, i.e., production, when digital cameras are employed.

DPX masters provide the input for film recording (digital images back to film for projection) or D-Cinema digital projection systems. Each DPX file represents a single image with a single component, e.g., luma, or multiple components, e.g., red, green, blue - YCrCb (chrominance-luminance data). Many variations in multiple component data are supported. DPX images may be produced by scanning film or by using a camera that produces a DPX output. Sound information is not incorporated in DPX and producers must manage soundtracks separately.

This flexible image format has a 2k bytes header that specifies characteristics of the image format, film or television related information and user-defined data. The format accommodates the various colour systems including RGB and YUV (or YPrPb) in 1-, 8-, 10-, 12- or 16-bit colour sample sizes.

The pixel data represents "printing density", the density that is seen by print film. Since a system gamma of 1.0 is preserved, any negative can be reproduced on the recorder retaining the original negative's gamma. The data is stored in log format, directly corresponding to density of the original material. Since the scanned material is likely a negative, the data can be said to be "log with gamma". The film can be disassembled by using the unique per layer contrasts of the colour negative.

The format has a notion of the "black point" and "white point" used for conversion to more limited range video signals. Conventionally, these points are 95 and 685 on the 0-1023 scale (but should be adjusted based upon actual negative content). Pixel values above 685 are "brighter than white", such as the sun, chrome highlights, etc. Pixel values below 95 represent black values exposed on the negative (the clear base of the film). These values can descend in practice as low as pixel values 20 or 30.

DPX files contain three sections in addition to picture data: generic file information including data format, motion picture and television industry specific information such as film’s perforation edge number or video timecode information, and user-defined information that may include ASCII data. Since each DPX frame is "one of tens of thousands" in a motion picture, users will track intellectual/bibliographic metadata separately from the set of DPX files.

Some of the drawbacks of the format are the absence of a standardized approach for managing accompanying sound, and limits on bit depth. Moreover, certain aspects of the standard can be quite confusing. As a result, some 10-bit DPX files have Red and Blue components interchanged, or Cb and Cr interchanged due to a different interpretation of the standard, or by getting wires crossed.

[Ima08] and [Lib08]

Motion JPEG 2000

This format is the part 3 of the JPEG 2000 (J2K) image-coding standard and is based on the core coding technology of the baseline JPEG 2000 Part 1. Motion JPEG 2000 (MJP2) is in essence a system layer that utilizes the J2K image encoding and adds support for video, audio and some metadata. It can handle both lossy and lossless compression, but it is in lossless that its use brings the most advantages, since this format in lossy mode cannot compete with other formats, in terms of compression ratio. This is explained by the fact that other lossy video compression formats take advantage of temporal redundancy, unlike MJP2 that encodes using solely intra frames.

This format addresses the market's interoperability problems, lack of standardisation, content protection issues, large storage requirements and the necessity for high bandwidth networks. By using a discrete wavelet transform, blocking artefacts associated to DCTs are avoided, but at high compression levels wavelet artefacts are introduced that take the form of blurring high contrast lines, making the image look softer to the viewer, has it was shown in chapter 1.3.2.

An interesting feature of this format is the multi-resolution compression that enables the division of an image by level of detail layers. By storing the most important layers up front, one can access the media quickly in progressively increasing quality. Furthermore, MJP2 can code Regions of Interest with higher quality, meaning that certain areas of the image can be presented in greater detail. The cost of all its advantages is expressed in a higher coding and decoding complexity and needed resources.

The coding algorithm of JPEG 2000 is a three-step transform coder consisting of a DWT, a uniform quantization with a central dead-zone, and an embedded bit-plane coder for the quantized transform coefficients. The latter operates independently on rectangular blocks of quantized transform coefficients, so-called codeblocks, by performing a context-adaptive binary arithmetic coding in three fractional bit-plane passes. The generation of individual embedded bitstreams for each codeblock is commonly referred to as tier 1 coding.

The organization of JPEG 2000 code-streams, also called the tier 2 coding process, is such that the bitstreams associated to a number of sub bit-planes of individual codeblocks can be aggregated into packets and layers depending on the desired support of different possible modes of progression. The ordering of sub bit-plane bitstreams of a collection of codeblocks can be performed in a Lagrangian rate-distortion optimization process subject to a given constraint on the desired bitrate.

In a kind of pre-processing step, JPEG 2000 supports the partitioning of the input image into rectangular, non-overlapping spatial regions on a regular grid, so-called tiles. Since each image can consist of multiple (spectral) components, such as e.g. RGB, tiling is applied to each component separately but in a spatially consistent way.

The resulting tile components are treated independently in the subsequent coding process, although a joint rate-distortion optimization across different components and/or tiles may be performed in the final tier 2 coding process. MJP2 can achieve 40:1 lossy compression rates, with very good quality.

Motion JPEG 2000 is capable of processing individual fields of a video sequence consisting of pictures with two interleaved fields, which were captured at different time instants. However, the underlying JPEG 2000 coding tools are effectively agnostic with respect to the specific nature of the processed picture, i.e., coding of a single field utilizes the same coding primitives as coding of a whole frame. This contrasts with the H.264/AVC video coding standard, which offers a more fine-tuned set of coding tools for processing interlaced video in the restricted case of pure intra coding.

[Hun03] and [MV03]

2.3.2.2 Lossy Formats

As seen before, by making use of a lossy compression format, some degree of quality loss is to be expected. Some codecs claim to be "almost" lossless, meaning they are slightly lossy, but make up for it with different processes and filters aimed at augmenting the visual fidelity, taking advantage of the human eyesight limitations. These methods like dithering, unblocking or denoising are generally employed by all lossy algorithms, and are most efficient (and needed) at high compression rates.

Digital Video (DV)

For the purposes of this document, this family of related formats is described together, emphasizing the underlying encoding, as it may be stored in media independent files. The DV video format, however, was developed in the early 1990s as a media dependent videocassette format. DV video signals are compressed using discrete cosine transforms within each frame, like the approach used with JPEG DCT. DV, contrary to most lossy formats, does not use frame-to-frame temporal encoding as employed by MPEG-2 and other formats.

The format is represented by open standards. Initial specification was developed by a consortium of ten companies and standardized by the International Electrotechnical Commission (IEC); now the number on involved companies has reached sixty. Further elaborations developed by Panasonic (DVCPRO), mostly for professional uses, were standardized by the SMPTE in three different bitrates: 25, 50 (SMPTE 314M) and 100 Mbits/s (SMPTE 370M). Sony's version, DVCAM is also a very popular professional format while their Digital8 version was produced mostly for the consumer market. [Wil05]

In 1992, AVC-Matsushita (now Panasonic) disseminated a *Blue Book* pertaining to the format. In 1998, based on that document, the original "DV" specification, oriented toward consumer use digital VCRs, was published as IEC 61834; part 2 specifies the video coding as well as many other elements. The specifications for the original professional DV were published as SMPTE 306M (1998) and 314M (1999); these two encoding formats are based on consumer DV and used in SMPTE D-7 and D-9 video systems. Largely, the preceding developments were framed in terms of videocassette media. Increasing interest in media-less handing of the bitstream has led to developments like Panasonic's *P2* solid state memory format cards, featured at the National Association of Broadcasters (NAB) trade show in 2005. Throughout its initial lifecycle, DV has been extremely successful on both consumer and professional markets. [Lib08]

The main goal of DV was to provide a robust solution based on already existing inexpensive technology; since motion estimation was a very resource consuming technique, it was put aside. By eliminating inter-frame coding, the substantial advantage of simple editing becomes possible. In addition, it was made so that the same single inexpensive chip could unfold as both encoder and decoder, meaning a symmetric system, where both opposite processes are of identical complexity. This plays into the designers' intentions of providing a cost effective technology primarily aimed at the consumer market.

The basic type of DV has a bit rate of 25 Mbits/s, being the 50 Mbits/s a buffed up version that uses two standard 25 Mbits/s chips instead of one. The two generated 25 Mbits/s streams are multiplexed into a 50 Mbits/s. Contrary to MPEG it was designed to generate a constant number of bits for each frame. To achieve this, very sophisticated algorithms ensure optimal bit allocation for every frame.

The first step in DV compression is, if necessary, to reduce the number of colour samples. Depending on the video frame rate, different colour spaces are used on the consumer format – DVCAM uses 4:1:1 for NSTC and 4:2:0 for PAL. DVCPRO 25 professional uses 4:1:1 on 25 Mbits/s and 4:2:2 on both 50 and 100 Mbits/s versions. For some video sources, some prefiltering might be required to avoid introduction of quantization artefacts.

Next the image is broke into 8x8 pixel macroblocks. Five macroblocks constitute a segment and each one of these must be compressed to the same number of bytes. The macroblocks from a segment are not sequential but chosen pseudo randomly from a frame to try to spread the complexity evenly among different segments (true randomness is not enforced because the reverse sequence needs to be known by the decoder). Besides that, because inverse shuffling is performed after decoding, it serves to de-correlate any DCT errors resulting from quantization, helping to prevent block artefacts.

Each block is then DCT transformed and analysed to determine its complexity; based on that analysis it is allocated to one of four banks of 16 quantization tables suitable for its energy range. Each coefficient within a macroblock is then quantized according to the previously attributed bank and similarly to MPEG a 4-bit end-of-block code is inserted when no more non-zero coefficients exist.

By using different banks for each macroblock of a segment and optimally searching for a combination of the right tables, all segments are guaranteed to have about the same number of bytes. To actually guarantee that all segments are coded with the same amount of bytes, after the aforementioned processes take place, a three pass algorithm redistributes the surplus data first within blocks, second within macroblock and finally within segments.

DV signals are formatted for transfer between devices in 80 byte blocks referred to as DV-DIF data (DV Digital Interface Format), and these DV-DIF blocks can be stored as files in raw form (.dv or .dif extension) or wrapped in such file formats as AVI, QuickTime, or MXF.

[Sym04]

MPEG-2

Encoding for compressed video and audio data, multiplexed with signalling information in a serial bitstream, was developed to fill the blanks left open by MPEG-1. The format was initially developed to serve the transmission of compressed television programs via broadcast, cablecast, and satellite, and was subsequently adopted for DVD production and for some online delivery systems.

MPEG-2 is defined only for the decoding process - all encoding processes are left to the companies to define, allowing space for improvement and innovation to easily better the compression effectiveness. This makes it extremely flexible, which is noticeable in the fact that MPEG-3 specification for high definition (HD) content was abandoned since it was possible to fulfil the same objective with some fine tuning to the existing MPEG-2 format and still comply to the original standards.

It was widely adopted for filmmaking, DVD disks, and other applications. Most significant, was the format's required use in digital terrestrial broadcasting to homes in the United States and several other nations, as governed by the ATSC (Advanced Television Systems Committee) specifications.

The format supports 4:4:4, 4:2:2 and 4:2:0 colour spaces (as seen in chapter 2.2.4 Chroma Subsampling) and allows 8-, 9- or 10-bit precision for DC coefficients of DCT blocks.

Opposed to MPEG-1, MPEG-2 supports interlaced video, although it still requires each frame to include all scan lines. Both coding and motion prediction are defined differently for frames and fields. Frame DCT and frame prediction are similar to MPEG-1, whilst field DCT and field prediction carry a few changes: splitting the luminance horizontally and vertically for the transform, using an alternate scan pattern and motion estimation uses 16x8 arrays of pixels.

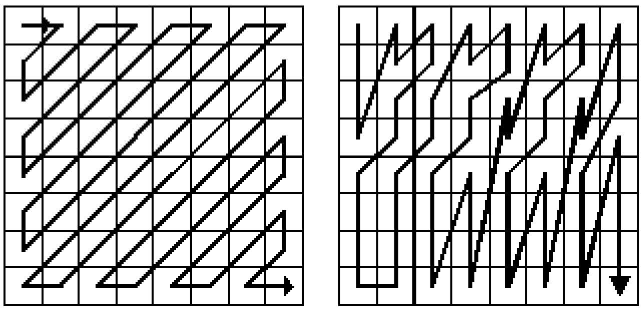


Figure 2.3.28 – Frame and field scan patterns.

Many software tools exist for encoding and decoding this format. Its wide adoption does not extend to all of its profiles, levels, or parts of the standard. A profile is a "defined subset of the syntax of the specification", in other words a profile imposes some bounds to the full syntax and it defines which tools or functionalities may be used to produce a bitstream and how. A level is a "defined set of constraints on the values which may be taken by the parameters of the specification within a particular profile". For instance, a given profile may use scalability or not, while the level gives the bounds to the image size.

Levels	**Profiles**					
	Simple	**Main**	**SNR**	**Spatial**	**High**	**4:2:2**
High		1920 by 1152 @ 60 Hz			1920 by 1152 @ 60 Hz *	1920 by 1152 @ 60 Hz
High 1440		1440 by 1152 @ 60 Hz		1440 by 1152 @ 60 Hz *	1440 by 1152 @ 60 Hz *	
Main	720 by 576 @ 30 Hz	720 by 576 @ 30 Hz	720 by 576 @ 30 Hz *		720 by 576 @ 30 Hz *	720 by 512 @ 30 Hz
Low		352 by 288 @ 30 Hz	352 by 288 @ 30 Hz *			

Table 2.3.5 – Partial specification of MPEG-2 levels and profiles. [Ber06] (* denotes scalability through the use of an enhancement layer)

Scalable coding profiles make it possible to transmit a second stream of lower resolution that a capable player will be able to decode, enhancing the main stream in signal-to-noise ratio (SNR profile), to a higher resolution (spatial profile) and a combination of these and some other improvements (high profile). Simple decoders will still be able to decode the main stream normally and simply ignore the extra one.

Technical (coding) information is contained in the MPEG-2 bitstream in macroblock headers, slice headers, picture headers, GOP headers, and sequence headers. The sequence header provides data needed before decoding can begin, such as the size of the picture and the frame rate. Because it is so important, the sequence header is usually repeated as often as twice a second. Slices are contained in a single row of macroblocks, ranging from just one macroblock to the whole line.

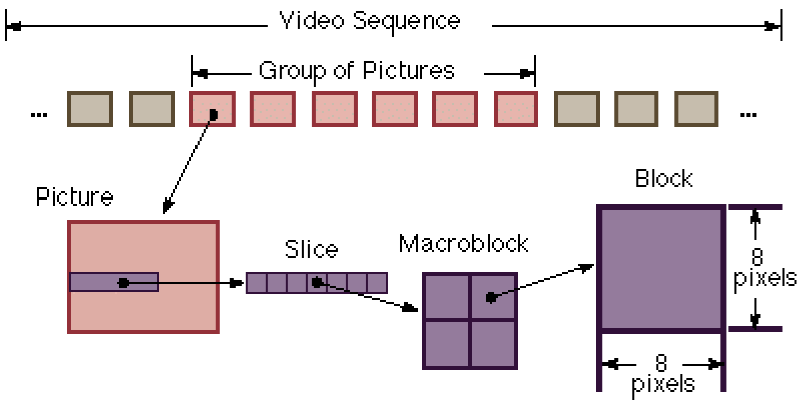


Figure 2.3.29 – MPEG-2 video stream subdivisions.

In an MPEG-2 system, the DCT and motion-compensated inter-frame prediction are combined, as shown in Figure 3.30. The coder subtracts the motion-compensated prediction from the source picture to form a 'prediction error' picture. The prediction error is transformed with the DCT, the coefficients are quantised and these quantised values coded using a variable length coder. The coded luminance and chrominance prediction error is combined with ‘side information’ required by the decoder, such as motion vectors and synchronising information, and formed into a bitstream for transmission in a correct decoding order.

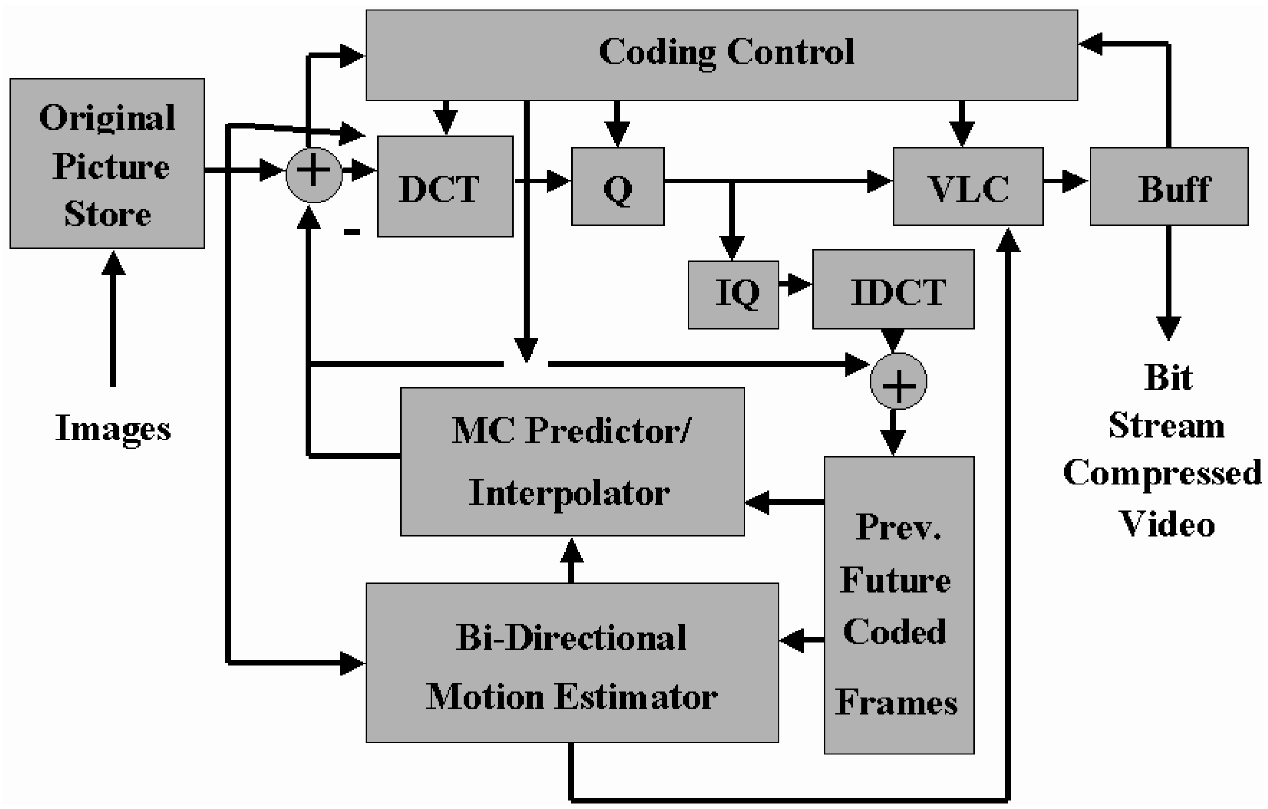

Coding Control
Original Picture Store
Images
+
-
DCT
Q
VLC
Buff
IQ
IDCT
+
Bit Stream Compressed Video
MC Predictor/ Interpolator
Prev. Future Coded Frames
Bi-Directional Motion Estimator

Figure 2.3.30 – MPEG-2 encoder functional diagram.

Apart from the video compression format itself, the MPEG-2 norm also unambiguously specifies its system layer wrappers – one for reliable sources, such as storage or DVDs, and another for error prone channels such as streaming environments. The term *stream* is used to name the format's sequence of encoded bytes. Elementary streams contain video, audio, or other data; a video elementary stream contains compressed video frames, plus sequence headers, group of picture headers, and other data needed to decode the stream. An elementary stream is broken up into packets of variable length, forming a *packetized elementary stream* (PES). Each PES packet includes a header. In many applications, the audio and video are multiplexed, thus combining the two elements. Packetized and multiplexed elementary streams may take the form of single-program *program streams* or be incorporated with other programs in a multi-program *transport stream*.

VOB (DVD *V*ideo *OB*ject) files are closely related to MPEG-2 files. VOB files are assembled by DVD producers, and contain the actual video, audio, subtitle, and menu contents in stream form. VOBs are encoded very much like standard MPEG-2 files. When the extension is renamed from .vob to .mpg or .mpeg the file will still be readable and will continue to hold all information, although most players supporting MPEG-2 do not support subtitle tracks. In order to burn the VOB files to a DVD±R disc, other standard DVD-Video files are needed as well, including IFO and BUP files.

Other large markets that make use of MPEG-2 are the European Digital Video Broadcasting – Terrestrial (DVB-T), Digital Video Broadcasting – Cable (DVB-C) and Digital Video Broadcasting – Satellite (DVB-S); these are used to digitally broadcast television compressed streams in different channels. Lately MPEG-4 AVC is being adopted into DVB-T and DVB-C to allow several enhancements such as HD content.

The lack of metadata, of the type called bibliographic by librarians, motivated the MPEG group to develop MPEG-7, a separately standardized structure for metadata to support all kind of descriptive media information for several purposes. Another drawback of this format are the associated artefacts, most commonly blocking artefacts as seen previously; there is also *mosquito noise* on sharp edges, *dirty window* in the form of non-moving residuals and *wavy noise* resulting from coarse quantization of high frequency coefficients.

[Fog96], [Lo] and [Sym04]

MPEG-4 Part 10 – AVC / H.264

MPEG-4 is defined in several parts pertaining to different aspects of this feature packed format. Part 10, called Advanced Video Coding (AVC), specifies a distinct video coding from the MPEG-4 part 2 (Visual Coding), and focuses on providing good image quality at very low rates, while still being flexible enough to handle HD content. The standard is explicitly optimized for 3-bit rates: below 64 Kbits/s, between 64 and 384 Kbits/s, and between 384 Kbits/s and 40 Mbits/s.

The most significant departure from conventional video formats is the concept of objects. Different parts of the final scene can be coded separately as video, audio and other objects to be later composited back by the decoder. Different objects may be generated independently or by separating the foreground objects from the background ones. Another example is the possibility of using a simple text file that contains subtitles and how to display them, avoiding the introduction of the typical mosquito noise artefacts associated with text's high frequency coefficients and bettering the compression efficiency, at the expense of a higher decoder complexity. All objects can be coordinated to play in various ways allowing different presentations of the same streams and objects.

In order to use these objects, they must first be sampled (usually at frame rate) and each sampling is called video object plane (VOP). A group of VOPs is called group of video object planes (GOV) and they are similar to GOPs. Scalability of VOPs or GOVs takes the form of video object layers (VOL). All layers associated with an object are included in video objects (VO). At last, a video session (VS) stands at the top level bearing all VOs. In fact, this object-

-oriented approach is so powerful and flexible that even three-dimensional video games can be coded as a series of video objects.

When super imposing two VOs their shared area at a given frame must be made transparent for the VO most behind. This is achieved by separating the image itself (texture) from its figure (shape). Binary shapes are used (known as binary alpha blocks, BABs) if the object is only either transparent or opaque, greyscale shapes are used instead if a smoother variation on transparency is required. Both shapes may be coded using a context-based arithmetic encoding (CAE), with a continually updated probability estimate, extended to include motion estimation. Intra CAE uses 10 pixels located to the left and above the one being coded for estimation while inter CAE uses some pixels of the same frame and others from a reference frame.

Texture coding resembles, in the most part, MPEG-2 intra- and inter-frame coding, with several improvements applied. A rectangular VOP is the closest equivalent to a video frame; since VOs can have different sizes they are coded in multiples of 16 pixels in each direction (a macroblock) and only those contained within or partially within the object's boundaries. One of the main differences separating MPEG-4 from MPEG-2's coding algorithm is the possibility of using multidimensional adaptive predictive coding, as described on chapter 2.2.3 Predicting Image Values. The coefficient readout alternates between vertical, horizontal, or zigzag scan if there is no DC prediction. Also new VLC tables are used that guarantee much better error resilience.

A distinct video object is the sprite, a usually considerably larger object used for images that are persistent throughout a scene. These are very common on video games, especially for coding backgrounds on which the action moves about, by panning the field of view all over the complete background. To avoid initial long loading times the sprite can be transmitted by sections as needed, if this is not enough, a different resolution version may also be progressively transmitted.

One of the most impressive features, that opens many interesting possibilities, is the ability in high-level decoders, to animate three-dimensional objects and to map textures onto them that stretch and expand freely. To attain this, wavelet based compression tools are used, since they provide better quality on scaling operations. The 3-D models are specified by a number of nodes, and profiles exist for human body and face templates for general animations.

To improve the coding efficiency of VOs several tools exist: global motion estimation permits an overall notion of an object's movement, quarter pixel motion estimation (quarter pel) allows much more accurate motion vectors and shape adaptive DCT improves the effectiveness of boundary blocks coding, when not all pixels belong to the object in question. Yet another feature is the possibility of interacting with the user, through all sorts of peripherals, making way to new innovative applications. With the use of additional enhancement layers, temporal or spatial scalability can be accomplished.

Typically, intra coding is based on a transform coder that is enhanced by applying some kind of inter-block or inter-sample prediction within one picture or slice. In contrast to previous video coding standards such as H.263 or MPEG-4 Part 2, where the prediction for intra pictures is conducted in the transform domain, prediction in H.264 uses spatially neighbouring samples of previously coded blocks.

There are two types of prediction modes for the luminance samples: the so-called Intra 4x4 mode based on predicting each block separately, and the Intra 16x16 mode for predicting a whole macroblock. In Intra 4x4 mode, the encoding process can choose between nine prediction modes, one of which represents a plane DC prediction, and the remaining eight modes operate as spatially directional predictors corresponding to eight different angles.

Prediction according to Intra 16x16 mode, which is well suited for smooth image areas, utilizes four prediction modes, as well as the separate intra prediction mode for the chrominance samples of a macroblock. Note that intra prediction across slice boundaries is not allowed in order to keep all slices independent of each other.

Profiles	**Levels**					
	Level 0	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Simple	1 VO, 64kb/s	4 VOs, 64kb/s	4 VOs, 128kb/s	4 VOs, 384kb/s		
Advanced Real Time Simple		4 VOs, 64kb/s	4 VOs, 128kb/s	4 VOs, 384kb/s	16 VOs, 2000kb/s	
Simple Scalable *		4 VOs, 128kb/s	4 VOs, 256kb/s			
Core		4 VOs, 384kb/s	16 VOs, 2000kb/s			
Advanced Core		4 VOs, 384kb/s	16 VOs, 2000kb/s			
Core Scalable *		4 VOs, 768kb/s	8 VOs, 15000kb/s	16 VOs, 4000kb/s		
Main			16 VOs, 2000kb/s	32 VOs, 15000kb/s	32 VOs, 38400kb/s	
Advanced Coding Efficiency		4 VOs, 384kb/s	16 VO, 2000kb/s	32 VOs, 15000kb/s		
N-bit			16 VO, 2000kb/s			
Simple Studio *		10 VOs, 180 kb/s	10 VOs, 600 kb/s	12 VOs, 900kb/s	12 VOs, 1800kb/s	
Core Studio *		10 VOs, 90 kb/s	10 VOs, 300 kb/s	10 VOs, 450kb/s	10 VOs, 900kb/s	
Advanced Simple	1 VO, 128kb/s	4 VOs, 128kb/s	4 VOs, 384kb/s	4 VOs, 768kb/s	4 VOs, 3000kb/s	4 VOs, 8000kb/s
Fine Granularity Scalable	1 VO, 128kb/s	4 VOs, 128kb/s	4 VOs, 384kb/s	4 VOs, 768kb/s	4 VOs, 3000kb/s	4 VOs, 8000kb/s

Table 2.3.6 – Maximum values for various MPEG-4 levels and profiles. (* denotes profiles that may use scalability by adding an enhancement layer) [Pen07]

As in MPEG-2, there are several profiles and levels specifications nameable in the form profile@level. In the above table only the most relevant profiles are present since there are even more profiles defined, notably for synthetic and graphic visual contents, some of which were later added in following versions of the specification.

The transform part of H.264 employs – similar to previous video coding standards – a block transform of the prediction residual. However, instead of using the popular 8×8 DCT, H.264 employs a multiplication-free, separable integer transform based on a 4×4 transform block size, which is suitable for implementation in 16-bit arithmetic. This low-complexity design leads to a subjectively pleasing reduction in ringing artefacts, and, at the same time, inverse transform mismatch problems are avoided.

For the four DC coefficients of each chrominance component, an additional 2×2 transform is applied. If a macroblock is coded in Intra 16 x 16 mode, a transform similar to 4×4 is performed for the 4×4 DC coefficients of the sixteen 4×4 transform blocks of the luminance signal, which correspond to a whole macroblock. This cascading of block transforms is equivalent to an extension of the length of the transform basis functions, thus leading to a better decorrelation of the signal in smooth image areas. For the quantization of transform coefficients, H.264 uses scalar quantization, which can be controlled by selecting for each macroblock the quantization parameter (QP) out of 52 values. The QP values are arranged such that there is an increase of the quantization step size by approximately 12% when incrementing the QP value by one. Prior to entropy coding, the quantized transform coefficients of a block are generally scanned in a zigzag fashion.

Entropy coding in H.264/AVC relies on two alternative methods. The basic entropy coding mode employs a zero-order Exp-Golomb code, which in the case of coding quantized transform coefficients is extended by the so-called Context-Adaptive Variable-Length Coding (CAVLC). The CAVLC method switches between various VLC tables depending on already coded syntax elements, and therefore achieves a better match of the VLC and the actually given conditional probabilities. For achieving a significantly improved coding efficiency than that provided by CAVLC, possibly at the expense of higher complexity, Context-based Adaptive Binary Arithmetic Coding (CABAC) as the second entropy coding mode of H.264.

For interlaced material, the H.264 design permits the choice of coding each picture by (1) combining the two fields together (frame coding mode), (2) coding the two fields separately (field coding mode), or (3) combining the two fields to one single frame and adaptively choosing for each pair of vertically adjacent macroblocks the frame or field coding mode. The latter option is referred to as macroblock-adaptive frame/field (MBAFF) coding, whereas the choice between options (1) and (2) is called the picture-adaptive frame/ field (PAFF) coding. Note that the underlying coding processes of H.264 in frame and field mode are specified in a similar way but with some important individual properties reflecting the anticipated specific geometric and statistical nature of frames and fields, respectively.

[Sym04], [MV03] and [Koe02]

VC-1

VC-1 is a video compression format created by Microsoft and based on WMV9, that after a rough initial standardisation process by the SMPTE, has become the current mandatory video compression format for mainstream consumer market high definition media HD-DVD and the “format war winner” Blu-ray. It features native support for interleaved video, making it attractive both for the broadcasting and video industry professionals. It was implemented by Microsoft in three codecs WMV3, WMVA and WVC1. The first two were deprecated because WVC1 implements the most features while being fully VC-1 compliant. MPEG-4 AVC/H.264 has been taking the lead in several market adoptions over this format, even though both formats have similar capabilities and performances, and contrary to initial beliefs, MPEG-4 part 10 can be licensed with lower costs.

This DCT based format can achieve 50% less bandwidth than MPEG-2 for the same quality, and stands toe to toe with MPEG-4 AVC, being less complex to encode and decode. For the final compression process non-computationally intensive high order entropy coding is used.

VC-1 introduces a large number of new approaches to known problems that focus on a good quality/efficiency relation:

- Adaptive Block Size Transform; traditionally, 8×8 transforms have been used for image and video coding. However, there is evidence to suggest that 4×4 transforms can reduce ringing artefacts at edges and discontinuities. VC-1 is capable of coding an 8×8 block using either an 8×8 transform, two 8×4 transforms, two 4×8 transforms, or four 4×4 transforms. This feature enables coding that takes advantage of the different transform sizes as needed for optimal image quality.
- 16-Bit Transforms; in order to minimize the computational complexity of the decoder. This also has the advantage of easy implementation on the large amount of digital signal processing (DSP) hardware built with 16-bit processors. Among the constraints put on VC-1 transforms, is the requirement that the 16-bit values used, produce results that can fit in 16 bits. The constraints on transforms ensure that decoding is as efficient as possible on a wide range of devices.
- Motion Compensation; is the process of generating a prediction of a video frame by displacing the reference frame. Typically, the prediction is formed for a block (an 8×8 pixel tile) or a macroblock (a 16×16 pixel tile) of data. The displacement of data due to motion is defined by a motion vector, which captures the shift along both the x- and y-axes. The efficiency of the codec is affected by the size of the predicted block, the granularity of sub-pixel data that can be captured, and the type of filter used for generating sub-pixel predictors. VC-1 uses 16×16 blocks for prediction, with the ability to generate mixed frames of 16×16 and 8×8 blocks. The finest granularity of sub-pixel information supported by VC-1 is 1/4 pixel. Two sets of filters are used by VC-1 for motion compensation: the first is an approximate bicubic filter with four taps; the second is a bilinear filter with two taps.
- Loop Filtering; VC-1 uses an in-loop deblocking filter that attempts to remove block-boundary discontinuities introduced by quantization errors in interpolated frames. These discontinuities can cause visible artefacts in the decompressed video frames and can influence the quality of the frame as a predictor for future interpolated frames. The loop filter takes into account the adaptive block size transforms. The filter is also optimized to reduce the number of operations required.
- Interlace Coding; interlaced video content is widely used in television broadcasting. When encoding interlaced content, the VC-1 codec can take advantage of the characteristics of interlaced frames to improve compression. This is achieved by using data from both fields to predict motion compensation in interpolated frames.
- Fading Compensation; due to the nature of compression that uses motion compensation, encoding of video frames that contain fades to or from black is very inefficient. With a uniform fade, every macroblock needs adjustments to luminance. VC-1 includes fading compensation, which detects fades and uses alternate methods to adjust luminance. This feature improves compression efficiency for sequences with fading and other global illumination changes.

- Differential Quantization; or dquant, is an encoding method in which multiple quantization steps are used within a single frame. Rather than quantize the entire frame with a single quantization level, macroblocks are identified within the frame that might benefit from lower quantization levels and greater number of preserved AC coefficients. Such macroblocks are then encoded at lower quantization levels than the one used for the remaining macroblocks in the frame. The simplest and typically most efficient form of differential quantization involves only two quantizer levels (bi-level dquant), but VC-1 supports multiple levels, too.

Levels	**Profiles**		
	Simple	**Main**	**Advanced**
Level 0 (Low)	96 Kbps (QCIF)	2 Mbps (QVGA)	2 Mbps (CIF)
Level 1 (Medium)	384 Kbps (CIF)	10 Mbps (576p)	10 Mbps (PAL-SD)
Level 2 (High)		20 Mbps (1080p)	20 Mbps (720p)
Level 3			45 Mbps (1080i)
Level 4			135 Mbps (1536p)

Table 2.3.7 – Maximum bitrates and common aspect ratios for VC-1 levels and profiles.

[LW07]

2.4 Concluding Thoughts on Video Compression

The reader should now have a good grasp of the main concepts that make up the basis of all compression algorithms up-to-date and knowledge of the entrails of the most commonly used video formats on the professional market.

All compression techniques exist towards a single objective – to reduce the amount of superfluous data. The definition of what is unnecessary or not, separates lossless from lossy algorithms. Lossless compression is mainly independent from the type of data and tends to depend heavily on correct prediction of symbol probability. Lossy compression effectiveness is deeply linked to the kind of data and takes advantage of the Markov sources properties inherent to audio, image and video data: they separate, with more or less success, the information from the structure that holds it. To differentiate these two aspects transforms are frequently used; not only for that but also to boost the ensuing compression techniques by giving the data a certain distribution that increases their compression ratios.

By narrowing our scope to the compression of image data, we take into consideration the process of capturing a picture: the rate at which we collect information (sampling) and the levels into which we store it (quantisation). This last process is also tied to the important lossy method of chroma subsampling, where the colour information is given less relevance favouring brightness information. On all these procedures, the limitations of the human sight are taken into account in order to remove every shred of redundant data. A more specific way to compress data is by using previous values to determine the next ones, a technique that wields good results on continuous information like photographic content.

The ruling technique in lossy video compression, called motion estimation and compensation, consists in taking advantage of the repetitive nature of video, and coding the differences between a series of frames instead of each frame independently. Lossless codecs must always use only intra frame compression, thus they are for the most part based on image compression algorithms. A subsequent notion that is not always delivered is that there are two main types of video compression: pure video codecs, made from scratch to compress video, and image codecs extended for video, which taking an image compression algorithm as a starting place add support for video. Pure video codecs can be used in some cases to code single frames but such is nearly always inefficient compared with image codecs and demands unusual settings. Moving JPEG 2000 is an example of an image format (JPEG 2000) used for video purposes; DV is a pure video codec although it just operates with intra coding.

A number of applications that ensure that video files abide by the decoding standards exist and are very heterogeneous in terms of offered features and supported formats. Because of this and allied to the fact that not all of them present trial versions, made comparing them extremely difficult. The choice of software relies purely on its intended use.

For applications where lossless media is required, Motion JPEG 2000 is the right choice to make. Packed with plenty of interesting features, its major drawback is that, in the actual state of hardware technology and the existing software applications, the performance is in some cases insufficient.

If no special features are needed, on lossy environments, besides a good image quality based on inexpensive, low hardware requirements, MPEG-2 is the ideal solution since loads of solutions exist and it is a proven and successful technology. If the hardware requirements are not such an important concern, then MPEG-4 AVC is clearly the path to follow since it offers the best quality/bitrate relation. If the intended goal sits somewhere in between these two formats, VC-1 is the way to go, considering its advantages over DV with only slightly higher system requirements.

3. Digital Video Containers

The concept of these media containing formats has evolved greatly in the last years. These file structures are worthy substitutes of the system layers, which are simply customised to each different format and support only a few kinds of audio and rarely other media.

A video container, or video wrapper, is a multi-format file specification, supporting several video and audio codecs as well as other media that enrich the playback experience. Some system layer designs only provide support for storage in reliable channels whereas video wrapper formats, for the most part, are more flexible and offer mechanisms to compensate less dependable communication channels, by allowing the loss of some packets without disrupting the playback of the media in question. Another feature that differentiates containing formats, is their ability to use references to different essences external to the file itself. All of this turns a wrapper container into an essence agnostic structure.

Still, the key word for container formats today is metadata, which literally means data about data. There are two major types of metadata: structural and descriptive. Older formats focus mainly on the first that represents the way audio, video and other data streams relate to each other, either inside or outside the container. Descriptive metadata is information relating to the several phases of media handling, especially of the capture stage. This makes it extremely important to professionals of cinema and television, which before the introduction of digital media had to rely on handwritten notes or external databases.

In annex, on section A.2. Video Container Formats, there is a compilation of some of the available analysis software for video wrapper formats.

3.1 Definition

A container format is a computer file format that can contain various types of data, compressed or mapped by hundreds of different codecs. The container file is used to store and manage most of the different data types. Simpler container formats can only contain audio codecs, while more advanced container formats can support multiple audio and video streams, subtitles, chapter-information, and metadata, along with the synchronization information needed to playback the various streams together.

Wrapper formats allow much more interoperability that permits file exchanges between different implementations. Additionally they maintain relevant metadata throughout changes made to a file. Some even allow simple or complex edition instructions that change the way a video can be reproduced in pre-determined ways. Most formats reserve some empty or unspecified structures for later innovations to augment the file's capabilities.

The more media formats and features a container supports, the more complexity it will require, so a careful specification must be made to neither over-complicate future implementations nor limit its capabilities. Another issue to take into account is to make the metadata structure broad enough to include various uses but not too extensive so that a smaller company that does not require that much complexity, could still use it.

3.2 Formats

Several standards and non-standards exist for file formats that can contain video and/or audio, differing wildly in supported video and audio codecs, metadata support and other capabilities.

The focus of this document is to introduce the reader, with a certain level of detail, to the inner workings of MXF. This format is taking its place as a format of choice in professional video applications, overtaking the lead QuickTime format has held for several years. There are some other popular container formats, not described here, like AAF (particular of the professional market and quite similar to MXF), AVI, Matroska (an open source alternative), MP4 or RealMedia, that possess strong market presences and as such would be good addictions to this study, but due to the shortage of time could not be included.

3.2.1 Advanced Systems Format

File format that wraps various content bitstreams; data types can include audio, video, script command, JPEG-compressed still images, binary, and other streams defined by developers. This description is focused on the use of the format for audio and video.

Typically, it is a final state format for end user delivery; may be used as a middle-state format, e.g., the video source when producing lower-resolution streaming versions.

Metadata is contained in Header Objects, consisting of many types of sub objects for various kinds of technical metadata. Producers may also include a Content Description Object for "bibliographic" metadata, including "Author", "Title", "Copyright", "Description", and "Rating." Other objects may be used to extend this description and for information about digital rights management and other purposes.

ASF uses an extensible set of GUIDs (Globally Unique IDentifiers) to identify all objects and entities within ASF files, including media types, codec types, error correction approaches, and other elements.

The Content Encryption Object within the header lets authors protect content by using Microsoft Digital Rights Manager version 1. The Extended Content Encryption Object lets authors protect content by using the Windows Media Rights Manager Software Development Kit (SDK). Using this tool, encrypted digital media files can be set to require the acquisition of a license containing a key before the content can be played.

The format was initially developed to support streaming media, and commentators report that *ASF* originally stood for *Advanced Streaming Format*. As potential uses went beyond streaming, it came to be called *Advanced Systems Format*, and it appears to function more or less as a successor-replacement for RIFF.

[All08]

3.2.2 Media eXchange Format

Object-based file format that wraps video, audio, and other bit streams (essences), optimized for content interchange or archiving by creators and/or distributors, and intended for implementation in devices ranging from cameras and video recorders to computer systems. In effect, the format bundles the essences and what amounts to an "edit decision list" (data used by audio-visual content editing systems) in an unambiguous way that is essence-agnostic and metadata-aware.

Although the specification, developed by the SMPTE, allows for complex, multipart objects, most commentators say, "MXF should be seen as the digital equivalent of videotape", an allusion to the videotape's simple, linear structure. There is a close resemblance with AAF that allows for the expression of complex relationships between both formats and wrapping interchange elements of a project for continued production or archiving.

Essences are wrapped in containers, implementations of the MXF Generic Container for specific encodings. Essences are generally internal (within the file itself) which can be either frame wrapped, if the several essences are interleaved together, or clip wrapped if each essence is independent from the others; they may otherwise be external, being identified by references to other files.

Tracks represent the temporal axis and are used to synchronise all the elements associated to the video. Timeline tracks (also known as standard tracks) describe segments that butt against one another to form continuous sequences, event tracks that may have overlapping events referring to the point at which descriptive metadata is valid, while static tracks have no timeline - all of their descriptive metadata applies the entire duration of the linked essence track. Tracks are synchronized by putting them in a package (container for tracks).

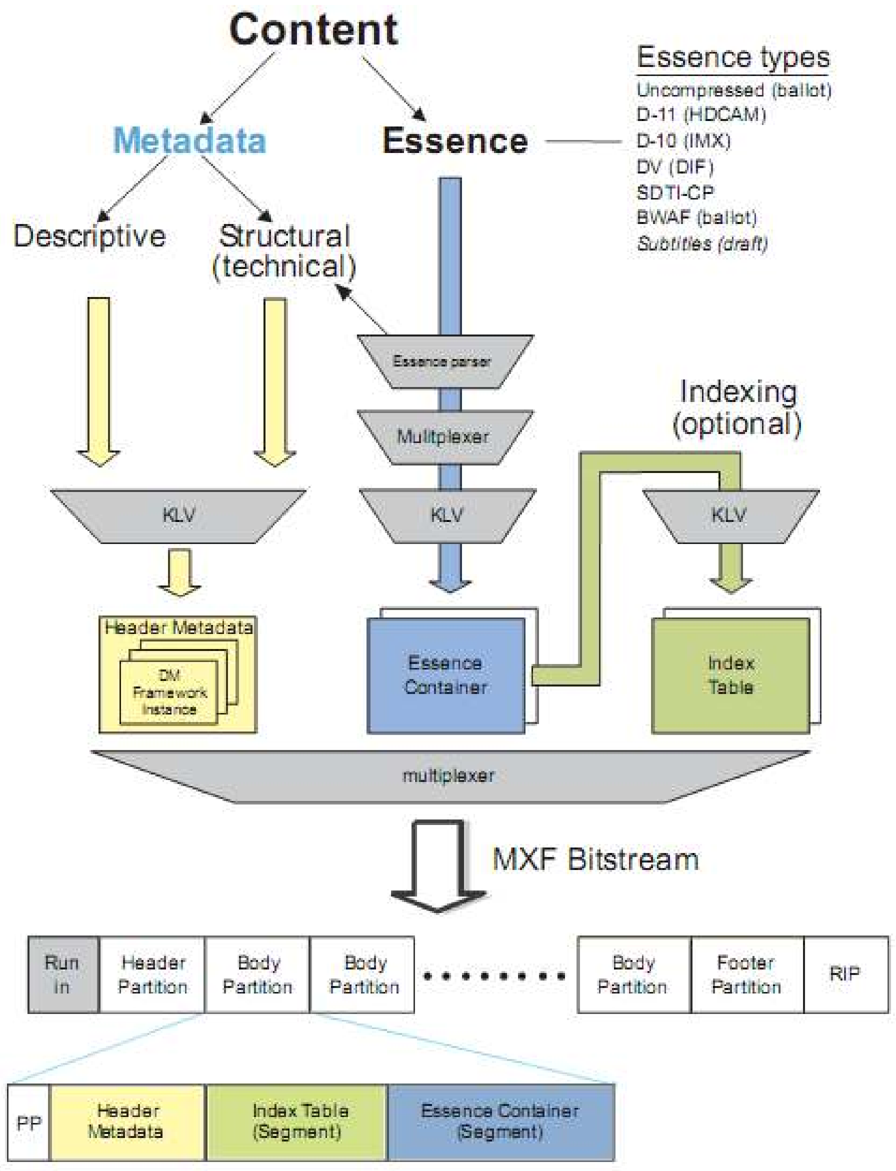


Figure 3.2.31 – MXF general structure.

[Dev02]

3.2.2.1 General Properties and Structures

The following specifications, be it data structures or coding norms, exist throughout an MXF file, spread across the four primary modular components: header, metadata, essence container and index table.

KLV Encoding

Every single element present within an MXF file is coded in the form of Key Length Value written in Abstract Syntax Notation 1 (ASN.1), which guarantees language independency. This type of encoding allows a much faster random access to the file, crucial to processing intensive operations.

Each segment of data is identified with a 16-byte code called Universal Label (UL) that functions as the Key. There are thousands of ULs defined by the SMPTE, but there are still many ULs available for private company use. Whenever possible, pre-defined codes should be used to maximise compatibility between different implementations of MXF software.

After the Key data follows the Length field, coded in ASN.1's BER notation using big-endian, meaning the last byte is the least significant, and allowing value lengths from 1 byte up to 72 Petabytes (!). This required flexibility is the reason why BER notation is used.

The Value field's size is stored in the length field, unless in certain cases in which the specific value is not known beforehand, like when a live incoming bitstream is being packed, a default value is used. In these cases, an additional mechanism must be implemented in the structure to ensure a correct decoding process.

In order to optimize accessing performance, KLV Alignment Grid (KAG) is used, consisting of a property inherent to the partition pack that defines the desired byte alignment of the start of essence elements within a partition. Different sizes can be attributed to different partitions to match the essence type. For example, a high data rate essence will have a larger KAG value than a low data rate essence. For overall performance there should be an integer relationship between all KAG values, and the file overall should respect the smallest of those values. [DW06]

Partitions

Rapid access and partial file restore are two advantages of file partitioning usually associated with streaming applications that need good error resilience and progressive access to the essences, without waiting for the whole file to be transmitted, before being possible to use it.

A Partition Pack always precedes a partition containing the properties about the content of the partition.

[DW06]

Packages and Sets

A Package is a construct that groups together components, with tracks and timelines, to enable synchronization of the essence components. Packages can be nested such that one package may be contained within another package to provide historical information about the essences' origins.

There are three types of packages: Material Package, which synchronises the output of an MXF file playback, Top-Level File Package, that describes the content actually stored or referenced in the file, and Lower-Level Source Package that maintains a history of all the changes and origins of each essence.

A Source Package is used to describe essence or historical essence. They are split into two different subclasses – file packages and physical packages. The first describes the essence stored in a MXF file. A physical package describes the essence before it entered the MXF domain. The historical derivation of an essence is kept through a Source Reference Chain that interconnects all source packages relating to that essence.

Material Packages define the output timeline of an MXF file, synchronising all stored essence. It is a collection of tracks - the structures that relate essence and time separated in three categories: timeline, event and static tracks. The precise relationship between the material package and the top-level file packages is governed by the operational pattern of the file.

Sets are unordered collections of MXF Items (properties), which can be either mandatory, or optional.

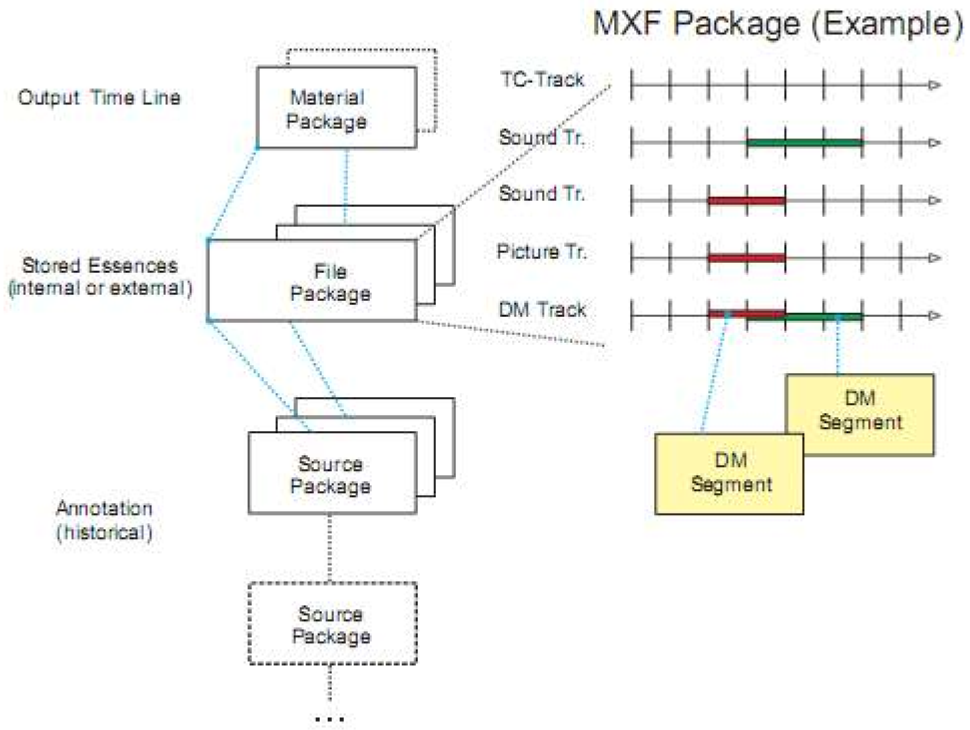


Figure 3.2.32 – MXF logical view.

[DW06]

Footer

Used to close the file and indicate that no further writing is being done to the file. Even though it is closed, it does not mean it is correctly closed, which only happens if the header metadata is properly updated.

The Random Index Pack is an optional partition lookup table found at the end of an MXF file that provides a quick and easy method to find all partitions on the file.

[DW06]

Operational Patterns

These patterns constitute a measure of the MXF files' complexity spread across two axes: one for the extent of editing possible, regarding the contained essences, and another for the level of package handling.

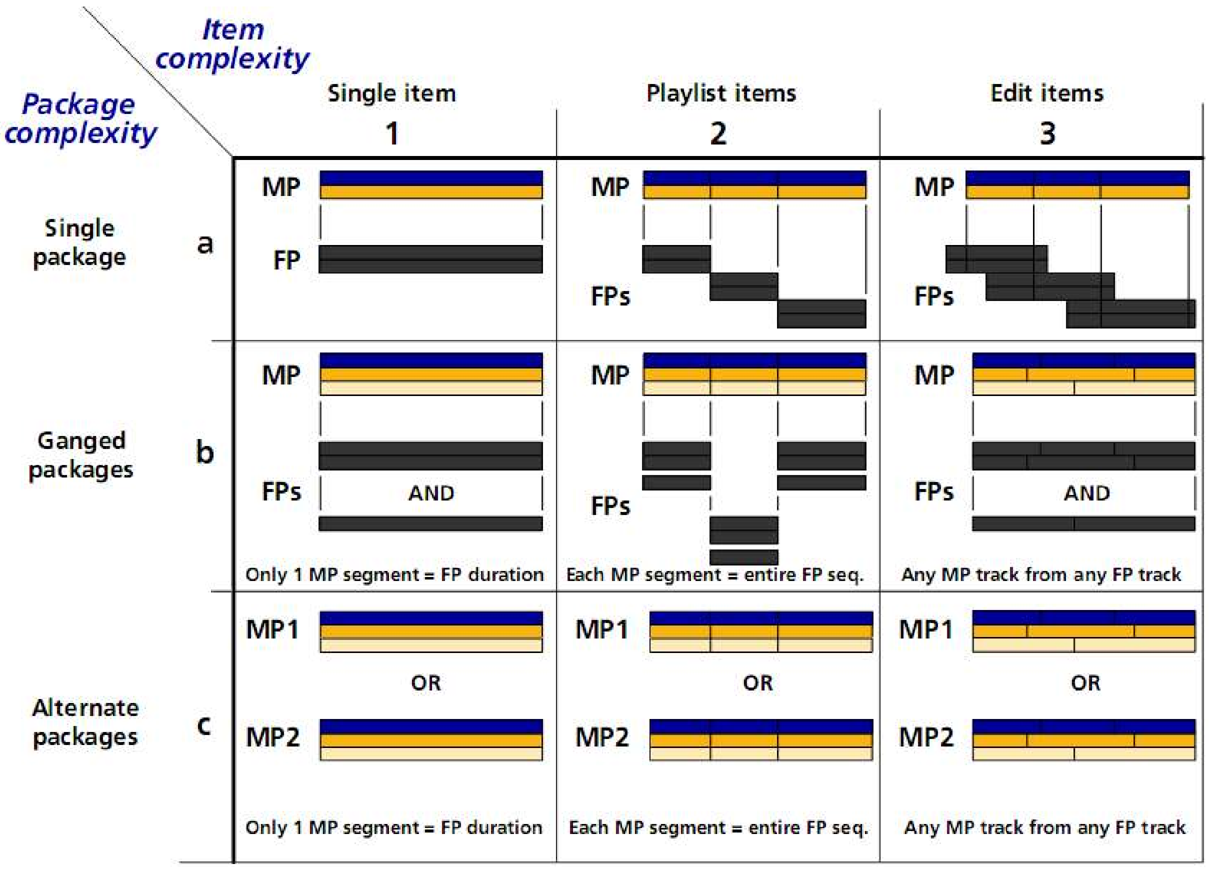


Figure 3.2.33 – MXF profiles.

The standard specifies a number of operational patterns (OPs) intended to accommodate different levels of complexity in a file, e.g., one essence or multiple essences, joint segments or a set of segments from which sub-segment are to be played, and so on. Application Specifications are specific profiles that are constrained to a certain OP, a particular encoding, metadata structure, etc. and other elements.

Possibly the most commonly found MXF profile is the OP-Atom that consists of a specialized OP1a profile limited to a single track, be it sound, video or another media. Groups of these files can be managed as external references by a main MXF file.

[DW06]

3.2.2.2 Header

Contains all required structural metadata that represents a complete description of what the files represents, specifically the synchronisation and timing of the contained essences, and it may be repeated throughout the file. Optionally descriptive metadata may also be included. Its complexity depends upon the profile of the MXF file.

The Primer Pack identifies all local tags used for a local set. Each tag must be unique within the partition in which it is used.

A Run In is an optional field that precedes the header partition pack used to disguise the MXF file as some other file type. It is not considered part of the MXF file and thus it is not coded with KLV. It can only be used with specialized MXF operational patterns and it may not contain the first 11 bytes of the partition pack key.

[DW06]

3.2.2.3 Metadata

Metadata is structured data that describes the characteristics of a resource. It shares many similar characteristics to the cataloguing that takes place in libraries, museums and archives. The term "meta" derives from the Greek word denoting a nature of a higher order or more fundamental kind. A metadata record consists of a number of pre-defined elements representing specific attributes of a resource, and each element can have one or more values.

This is of the utmost interest for any professional user of video since many thousands of different media are at some point stored and need to be easily found for a myriad of distinct uses.

Structural Metadata is the necessary data to correctly playback the contents of an MXF file. It is contained mainly in the header.

It is what tells a MXF reader what, where and how the essences contained are to be played. It handles the various Track Ids, oversees all synchronization issues and keeps information about every relevant component inside or outside the MXF file.

Descriptive Metadata (DM), on the other hand, characterises the essence’s content, not its structure or properties, and is sustained by MXF’s DM framework, as part of the header metadata. It is usually a manually introduced data to describe the whole file, some scenes or a single frame. All kinds of information can be added: where a certain clip was recorded, what objects produced a certain audio essence, the subtitles language, and so on. These structures come as a substitute to what was made before with tape labels.

Descriptive Metadata Scheme – 1 (DMS-1) defines an extensive scheme divided into three different frameworks: Production, Clip and Scene. Production applies to the production as a whole, clip metadata concerns to the content as it was created or captured and finally scene metadata relates to the editorial intent or to individual scenes.

Room for further extensions exist so that new necessities can be accommodated for, in an easy way. This means that if the standard metadata schemes are not enough, the definition of custom structures is possible, granted that companies follow all the conventions that rule MXF metadata specifications.

[DW06]

3.2.2.4 Generic Container

Essence is the name given to any media associated with an MXF file – video, audio, subtitles, pictures, other kinds of data, etc.

A generic container is used to store the different structures within an MXF file. If the content of such container is any kind of essence it is called an essence container and there can be multiple ones within an MXF file or externally referenced. Optionally metadata can be stored, apart from the header metadata, within generic containers or together with the essences themselves.

[DW06]

3.2.2.5 Index Table

This structure exists to perform random access enabling partial restore, scrubbing and other operations. It essentially converts time offsets into byte offsets. The richness of MXF makes the index table's design rather complex.

[DW06]

3.2.3 QuickTime File Format

It is important to clearly differentiate the two aspects around the name QuickTime: its file format and its framework. The first is described in this section and is a wrapper layer, capable of holding different kinds of essences and even some small editing instructions. The second, portrayed later in chapter 4.2.5 QuickTime Framework from Multimedia Frameworks, is a multimedia middleware aimed at playing and editing media. QuickTime is also erroneously associated to a video compression format, which true names range from Cinepak to the Sorenson video codecs. Additionally, it is often confused with the software multimedia player that goes by the name of QuickTime Player that is in fact based on the homonymous framework.

The QuickTime file format is one of the oldest and most multifaceted that uses a track model for organising the temporal data of a movie. A movie can contain one or more tracks - a track is a time-ordered sequence of a media type. The media are addressed using an *edit list,* which is a list of end-points of digital media clips or segments.

This file format wraps video, audio, and other bitstreams. This document is concerned with QuickTime video presentations (generally with a synchronized audio stream), called "movie" by Apple. QuickTime also wraps still images (QuickTime Image Files), animations not recorded as video, virtual reality, etc., and these are not discussed.

Usually used as a final state format for end user delivery and sometimes a middle-state format, e.g., the source when producing lower-resolution streaming versions.

Technical metadata is in the headers for the *atoms* and *container atoms* that comprise a QuickTime file and in the locations in the file. Descriptive ("bibliographic") metadata may be entered within the following annotations: "Album", "Artist", "Author", "Comment", "Copyright", "Creation Date", "Description", "Director", "Disclaimer", "Full Name", "Host Computer", "Information", "Make", "Model", "Original Format", "Original Source", "Performers", "Producer", "Product", "Software", "Special Playback Requirements", "Warning", and "Writer."

QuickTime files may be structured to require end users to enter a *media key* before the file can be played. Newsgroup traffic about *iTunes* includes a statement from a commentator that reports, "*iTunes* uses a DRM system that prevents files to be played on more than 3 platforms and only the iTunes player can cope with that DRM system."

Introduced in 1991, structured for use in Windows in 1994 and in the mid-to-late 1990s, the format influenced the shape of MPEG-4, a testament of the formats flexible and intelligent architecture.

[Bue08] and [Apl07]

3.3 Closing Remarks on Video Containers

A certain confusion frequently arises regarding digital video files: the video compression format is misinterpreted as the containing format and vice-versa. The first information we have access to, on a video file is its extension, which only informs us of the file's outer layer. Sometimes this might be sufficient to acknowledge the compression format, since it is using the default system layer wrapper but most times what we see is the container format.

A video wrapper format offers significant advantages over the coded system layers, not only to fit the needs of video professionals but also to regular consumers. One major advantage is the possibility of encapsulating many distinct formats of different media types within a single container, streamlining the way multimedia can be played. Another major feature is the extended metadata support, which is most valuable in a professional scenario, allowing all kinds of annotations to be intrinsically bound to the essence. An added value of some wrapper formats is the capability of providing the structures for editing different ways for the separate essences to be reproduced, without altering the essences themselves, making it possible to play different presentations from the same set of essences. By enclosing different compression formats in these flexible formats, their distribution becomes independent of the communication channels, due to the ability of various containers to adapt to less reliable transmission environments. Essences need not be contained within a wrapper file, since exterior essences can be mapped as external references. It is also possible to maintain historical information concerning the contained essences.

The biggest drawback about using wrapper formats comes from the obvious increase in complexity and therefore an escalation on the required resources, on both encoder and the decoder sides. Thanks to the rapid development of hardware and the decline of its prices, more and more resources are available, making way to more sophisticated solutions.

The mentioned advantages of this kind of file formats do not exist on all of them. This happens, because depending on the projected usage, some features may or may not be desirable. MXF is ideal for the professional market, since it was thought up from scratch taking into account their needs, and possible future requirements; also, it contains all the mentioned capabilities discussed previously. For a long time QuickTime's aging format has dominated the industrial video market, but MXF offers a fresh take on the new challenges of digital video containment.

For consumer use, several lightweight wrapping formats exist, such as the ubiquitous Audio Video Interleave (AVI) and the more recent Matroska (MKV), but these evidently offer much less features than other highly developed formats. As compensation to this, they also require much fewer resources.

Analyzers for this kind of files are much more specialised than compression format verifiers, and generally only examine the wrapper format itself, not the enclosed essences. Again, the major players of this kind of software, sells to a select group of clients, putting much more emphasis on direct publicity than the information channels I have at my disposal. A list of the found analysing applications is located within the annex A.2. Video Container Formats.

4. Multimedia Frameworks

Playing multimedia files, on a software level, is an intricate process that involves handling different input/output devices, many file formats, reusing common algorithms, different data streams, transmission of such streams across networks, handling buffers of samples, rendering in synchrony and scheduling all processes while maintaining high quality output.

This form of middleware, the multimedia frameworks, exists to allow and facilitate, other software, to play, edit and capture video and/or audio using several modules that interact in succession with each other through channels. It should be added that not only audiovisual material is processed but also any kind of data flow, like subtitles or a music album name, for instance.

This section of the document aims at comparing the different available frameworks over their meaningful features and capabilities.

Many video players available in the market are driven by underlying multimedia frameworks, fact that is frequently ignored by their users. For example Windows Media Player is powered by DirectShow (and in some cases Media Foundation), iTunes and QuickTime Player stand on the QuickTime framework and RealPlayer operates over the Helix framework.

As an alternative to full-blown frameworks that pack many advanced playback and editing features, simpler applications exist focused solely on a straightforward playback of media. These implement every feature and every codec in a much more monolithic fashion, requiring new versions to add new capabilities and formats. The most successful example is the platform-independent Flash Player: a lightweight application that allows multimedia content to run directly from internet browsers. Its importance can be understood by realizing that over 98% of all internet-enabled computers [Ado08] have Flash Player installed, overtaking even Java's PC coverage.

A mixed format between multimedia frameworks and self-contained players exist as an open sourced solution named VLC Media Player, that playback and even encodes media to a number of compression formats. This multi operating system application manages to maintain the plug-in based approach of media frameworks with the compactness of a video player, without more than a small resource footprint. It also has available a video streaming version.

In annex B. Simple Player Source Code, it is possible to find example source code of a simple media player for practically all of the studied frameworks.

4.1 Definition

The general design behind any multimedia framework is a sequence of interconnected *components*, which "pull" or "push" streams of information from one to another through *channels*, generally ranging from file/stream *sources*, to media *processors* and *renderers*. This sequence is usually assembled by a top-level *builder entity*, and it represents one way of handling a certain file or stream.

Source components read information from an address in the network, local files or capture devices onto renderers or, much more often, onto processors.

Processor components are responsible for changing or extracting data from the streams and are composed of several different types:

- Splitters, which accept one intake and usually separate audio from video from other data and output each to a different channel (1 to N);
- Parsers, that extract some kind of relevant information, usually to read audio and video from a container file;
- Encoders and decoders, also known as codecs, take simple data and code it according to a certain compression format or transform coded information into a simpler form (usually raw information ready to be rendered or recoded);
- Multiplexers and demultiplexers, merge into only one from several streams or vice-versa (N to 1 or 1 to N);
- Tees, that multiply a given stream (for example to simultaneously write to a disk and render a stream);
- Effects filters, add different effects to audio, video and other data.
- Colour space converters, to change between several colour spaces like RGB to HSV or YUV;
- Sample-rate converters, recreate the stream with a different sample rate;
- Mixers, allow mixing of different streams of a given kind;
- Video scalers, convert video to different aspect ratios or resolutions.

Some processor components may belong to more than one of the aforementioned types.

Renderer components receive either raw data and render it (on a screen, speakers, etc) or act as sinks by taking data and storing it into a container format (most frequently encoded data).

Certain events must be handled by both the builder entity and the calling application, and function as a means of communication and status reports; in some cases, special buses facilitate the communication process. Reusable buffers are used as a means of conveying data between components.

The developer has the freedom to either "hand pick" which modules to use or to let the framework's builder entity decide the best way to reproduce the input media.

As a side note, I used some original terms to name the various parts that do not correspond to any framework in particular, so that the general definition could be applied to all the frameworks, in an understandable fashion.

4.2 Main Frameworks

Looking at the framework panorama today, the trend on multimedia frameworks is to go cross-platform. From the most used frameworks chosen to be studied in detail, only Microsoft limits its frameworks to Windows, due to its ubiquity. As it is apparent throughout this chapter, certain frameworks present much more information than others do, be it officially or through community websites dedicated to multimedia programming.

4.2.1 DirectShow

DirectShow was born as a stand-alone technology under the name ActiveMovie (codename Quartz), meant to supersede the obsolescent Video for Windows, but it is now part of the DirectX library. This framework was inspired by Clockwork, a modular media-processing framework in which semi-independent components worked together to process digital media streams, loosely inspired by Apple's QuickTime framework. DirectShow was first distributed with its actual name, as part of Windows 98. It is based on COM objects, which makes it language agnostic, meaning it is independent of any programming language.

Component Object Model (COM) is an interface standard for software componentry introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in any programming language that supports the technology. The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one they were created in, even across machine boundaries. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separate from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference counting.

DirectShow is the most used multimedia platform ever, having spawned dozens of companies all around the world, who produce their own filters or base their software on this platform. Because of this, too many filters are available for the same purposes making the selection of the proper ones, a difficult task. DirectShow's merit system is not enough to handle the multitude of available filters. The function *renderfile ()* automatically sets the graph for playback with the available filters, based on this system.

Similar to the general architecture of multimedia frameworks seen before, on DirectShow, several *filters* connect through *pins* to form a *graph*, all of which is controlled through an instance of *filter graph manager*. Filters can belong to one of three major types: *source*, *transform* and *render* filters. The use of the framework itself is free of charge, and some selected modules, not including the graph managers, are even open sourced, but third party filters are often charged for, and are necessary for playback and encoding of newer codecs.

For developers, the structured component architecture of DirectShow provides many useful features and makes it easy to support new and customized data formats or to create custom effects and transforms on standard formats. Still, during the first years many of the details for programming on this platform remained obscure to developers due to incomplete or non-existing documentation. Today that problem has been corrected, thus making it much easier to work with, if knowledge of Window programming paradigm and COM objects is present. However, the documentation has not been maintained during the last years resulting on an aging framework that has been deprecated and "hidden" in the Windows Platform SDK.

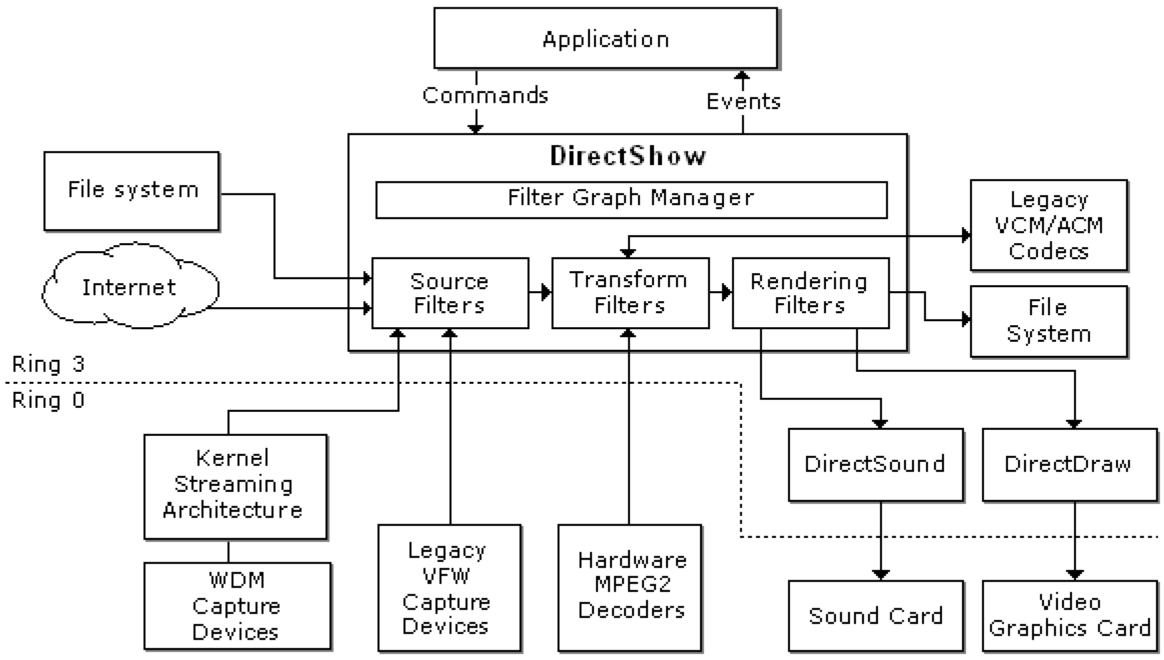


Figure 4.2.34 – DirectShow filter architecture. (rings refer to operating systems' access levels)

Besides being somewhat convoluted to code on, most of the times when uninstalling software that included DirectShow filters, it is not guaranteed that those filters will be unregistered, which can possibly cause future complications.

DirectShow is no longer recommended for game development, as it could be used a few years ago, but facing the advancements in computer graphical processing power, new powerful hardware and the game industry requirements, DirectShow is practically obsolete.

Included within DirectShow's own SDK is a very useful GraphEdit tool. This little application allows developers to test the creation of different graphs using a simple GUI and to check which filters are currently being used (enabled by adding a small portion of code to your program), facilitating the debug process.

There is a very complete information base made available by Microsoft itself on Microsoft Developer Network (MSDN) [MSD08a], as well as on forums about DirectShow [MSD08b] and [TCP08].

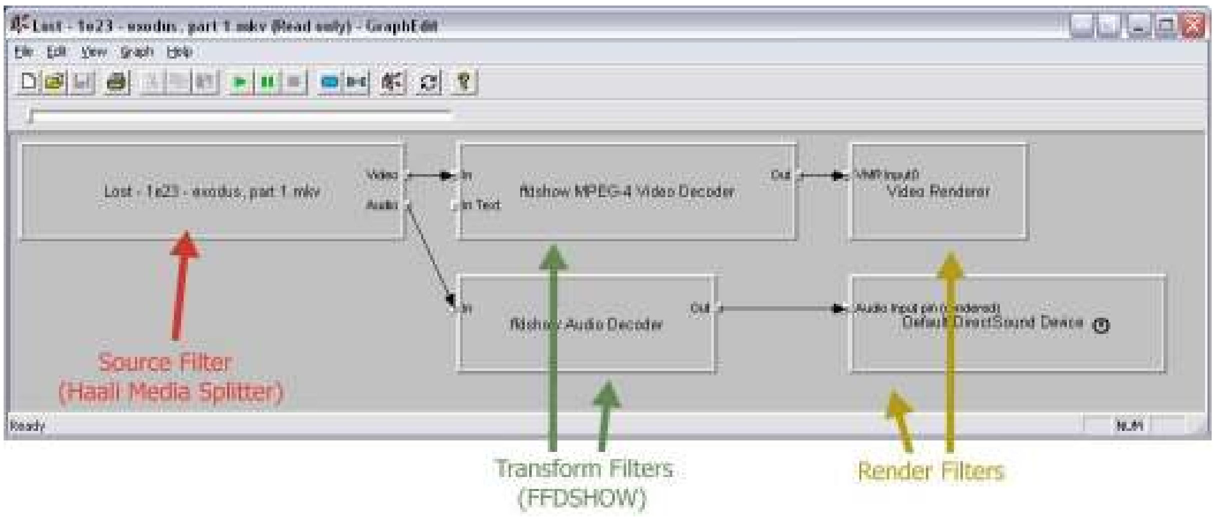

Lost - 1e23 - exodus, part 1.mkv (Read only) - GraphEdit
File Edit View Graph Help
Lost - 1e23 - exodus, part 1.mkv
Video
Audio
In
In Text
ffdshow MPEG-4 Video Decoder
Out
VMR Input0
Video Renderer
In
ffdshow Audio Decoder
Out
Audio Input pin (rendered)
Default DirectSound Device
Ready
NUM
Source Filter (Haali Media Splitter)
Transform Filters (FFDSHOW)
Render Filters

Figure 4.2.35 – GraphEdit's interface, representing a possible graph for an mkv file containing a MPEG-4 video essence.

Microsoft DirectShow Editing Services (DES) is a closed source application programming interface (API) that greatly simplifies the tasks involved in video editing. DES is built on top of the core DirectShow architecture. It abstracts much of the complexity of DirectShow, and provides a set of interfaces designed specifically for manipulating video editing projects. This abstraction level simplifies the development of video editing applications by adding the flexibility needed for nonlinear editing.

DES brings these features to DirectShow:

- A timeline model that organizes video and audio tracks into nested layers, making it easy to manipulate the final production;
- The ability to preview a video project on the fly;
- Project persistence through an XML-based format;
- Support for video and audio effects, as well as transitions between video tracks (such as fades and wipes);
- Over 100 standard wipes, as defined by the SMPTE;
- Keying based on hue, luminance, RGB value, or alpha value;
- Automatic conversion of frame rates and audio sampling rates, enabling a production to use heterogeneous sources;
- Resizing or cropping of video.

[MSD08a]

4.2.2 Gstreamer

Contrary to DirectShow, Gstreamer is open sourced, multi platform and aims mainly at Linux. It was designed to fill the void in Linux multimedia support, following the same architecture behind DirectShow. It is being called the *de facto* standard of multimedia frameworks on Unix based operating systems. Still much work remains to be done.

Gstreamer requires knowledge of GObject programming, which was inherited from its Gnome ancestry, and can be developed in several coding languages such as C, Perl, Python and Ruby. Some more bindings to other languages are in the works by the community.

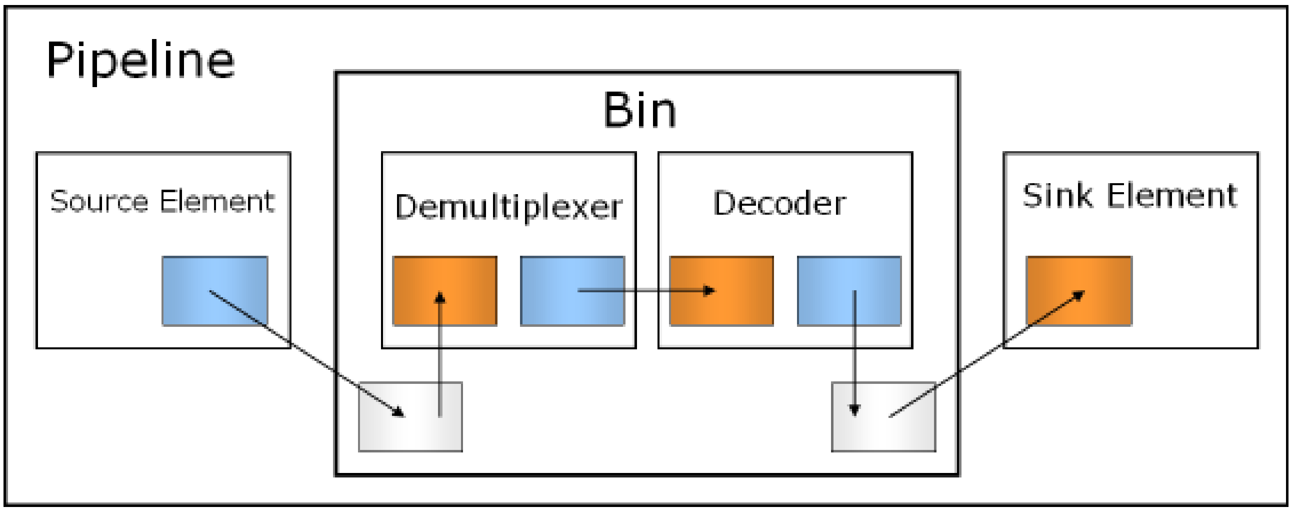


Figure 4.2.36 – Gstreamer's architecture.

The software architecture involves a series of *elements* that connect and communicate through *pads*, coupled in sets called *bins*. Several bins and elements are brought together on a special top-level bin called *pipeline*. Elements can be of three kinds: *sources*, *filters* and *sinks*.

Pads are named *sink pads* if they are inputs or *source pads* if they are outputs; they are separated according to their availability: always, sometimes or on-request. G*host pads* are a particular kind of pads that exist only inside a bin, enabling communication with the exterior and thus separating the bin's output from its elements'.

All pipeline elements are constantly in one of these states: null, ready, paused or playing.

A similar feature to DirectShow's *renderfile()* exists, called autoplugging. It can automatically choose and interconnect the necessary elements for media reproduction.

Differing from Microsoft's flagship framework, it is plug-in based and as such a little more complex but much more flexible and efficient. These plug-ins are elements that can be sorted into three categories: good, bad and ugly plug-ins. The first two are well tested and efficient, the last is not thoroughly tested; the first is under their preferred license (LGPL) and the second might pose problems to distributors seeing as they have licensing costs associated. All plug-ins connect to the core elements, which are an intrinsic part of Gstreamer itself.

Moreover, Gstreamer allows the developers to determine what media types a certain element handles, without having to load it into memory, enabling much more agile processes. Pipelines can be saved in XML files and later reloaded, providing a quick and easy way of debugging and prototyping.

Communication between elements, the pipeline and the top application are made possible through signals sent to a common asynchronous bus set by the pipeline.

From all the evaluated frameworks, Gstreamer engineers' care in design of the thread environment, boosts it as the most efficient among its peers.

This framework is intended to be a testing platform for easy prototyping and release of new codecs, by providing a simple and open sourced environment. Original groundbreaking projects have a fertile soil in which to grow.

One promising project that will enable Gstreamer to play virtually any kind of file is *Pitfdll* – it allows the use of *dll*'s from DirectShow and *qtx*'s from QuickTime as regular Gstreamer plug-ins. If the framework and this project can achieve a stable development stage, DirectShow might be facing a serious adversary.

There are already some projects to bring Gstreamer to the mobile scene, one of which involving Nokia's mobile phone *Nokia 770 Internet Tablet* and its successor, the *Nokia N800*. Gstreamer has network capabilities in distributing media via several different protocols that could open some new markets and business ideas.

Even though many good ideas and a well-structured architecture are a good foundation to an aspiring media framework, Gstreamer still needs to mature some more to be a serious alternative outside Linux environments. The application's Binary Interface, the core connection of the framework to the operating system, changes too often; it is still not stable enough for companies to venture into it. On top of that, so far very few capture devices are supported, hindering a more enterprising adoption of this software.

Gstreamer abides by a very permissive license: LGPL, that makes even distributing software based on the framework, practically free. This is true except for the use of ugly plug-ins that bear individual proprietary licenses that must be paid for separately.

Gstreamer is well documented, and has a very active and knowledgeable community on its official forums, site and Gentoo forums, so getting adequate support is an uncomplicated task.

Gstreamer does much more than just playback - video editors, media servers, music synthesis systems, and many more, have been built on top of Gstreamer such as Totem [Gno08] (a movie player for the GNOME desktop) and Songbird [Pio08] (a cross-platform media player written using the XUL toolkit from Mozilla). It is even possible to run a complicated set of elements to decode a media file through command line, via a specific syntax. This exemplifies decoding an mp3 music file onto alsasink, the audio output element:

```
    $ gst-launch-0.10 filesrc location="concept.mp3" ! decodebin ! alsasink
```

Similar to GraphEdit there is the Gstreamer Editor, that allows developers to see in real-time which elements are being used, and to test, through a simple GUI, how different pipelines behave with a given media.

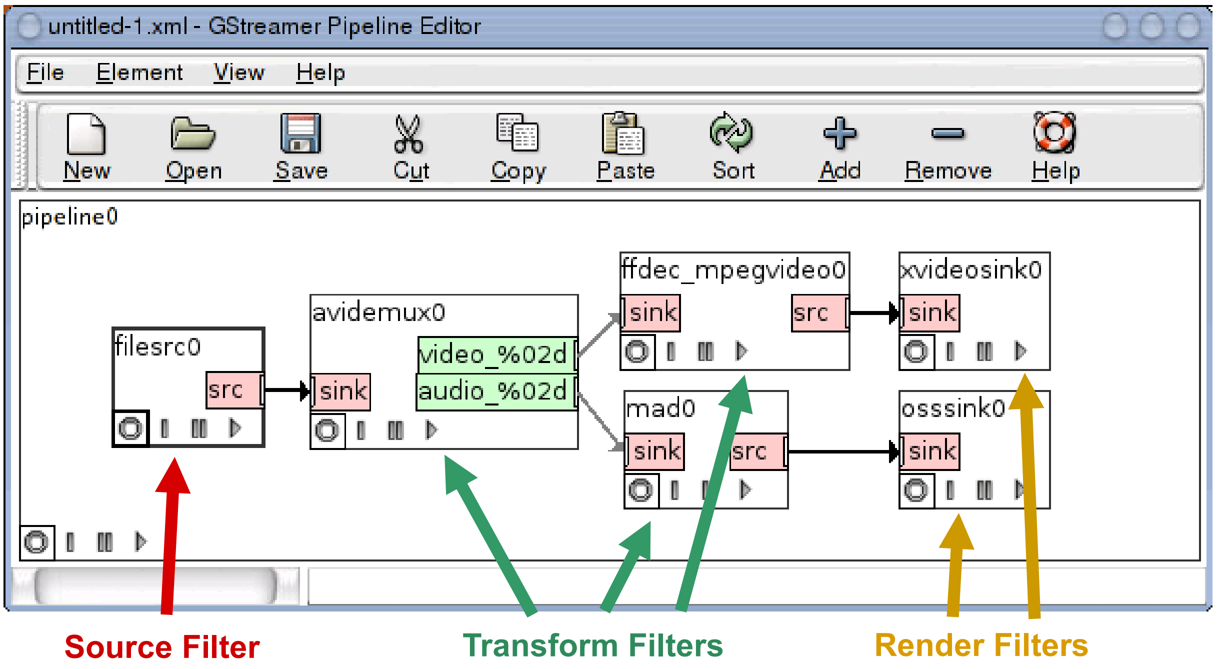


Figure 4.2.37 – Gstreamer Pipeline Editor's interface showing the decoding of an AVI video file.

4.2.3 Helix Framework

In 2002, RealNetworks made a bold move and decided to make RealPlayer's underlying framework open sourced, whilst keeping RealMedia close sourced. This allowed more people and companies to adopt this technology and to develop it further in conjunction with RealNetworks itself. It is the main competitor of Gstreamer and the most distributed framework on the mobile phone scene. Despite that, it still lacks some encoding formats, due mostly to copyright protection.

The Helix platform consists of three components:

- Helix DNA Client - the universal playback engine supporting the decode and playback of any format and on any operating system;
- Helix DNA Producer - the encoding engine and APIs that allow you to convert video and audio into digital media in a streamlined fashion;
- Helix DNA Server - the core engine for digital media delivery that will enable you to build a server for any media format and any operating system you wish.

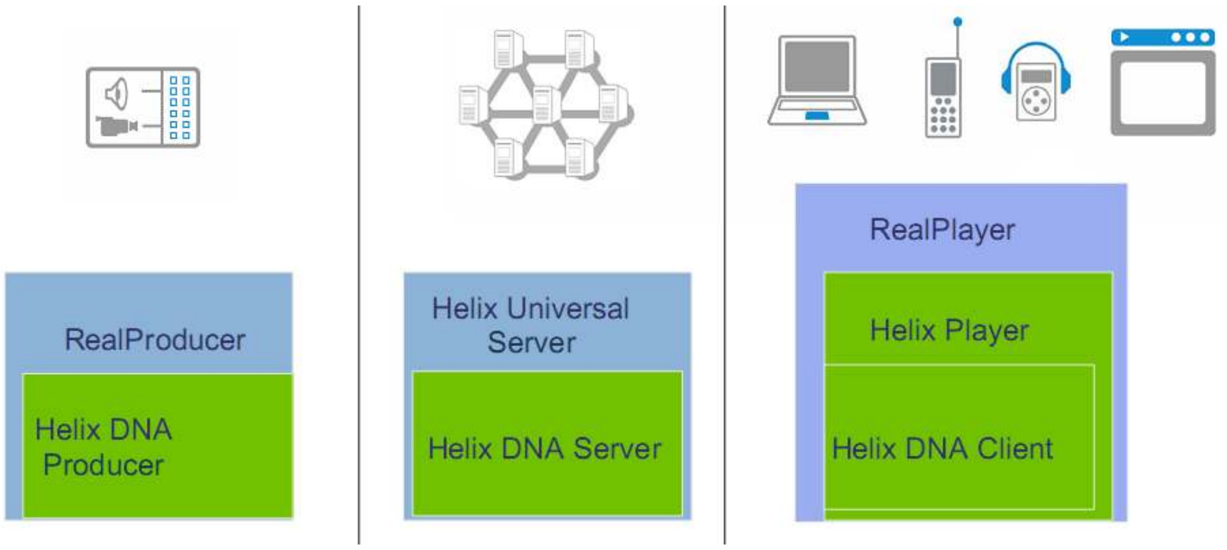


Figure 4.2.38 – Helix Project's components (green) and RealMedia products (blue).

It is generally available for Linux, MacOS, Solaris, OpenBSD, Symbian and Windows, although some components are only supported in some operating systems. Helix is already integrated with Trolltech in the application Qtopia 4.3.0. Cell phones using Symbian, from companies like Nokia, Motorola, Palm, Samsung and Sony Ericsson use it as an all-round media player.

All Helix components are plug-in based, in order to seamlessly handle different types of operations and file formats. These plug-ins can be labelled as one of the following types:

File system plug-in,

Provides access to different types of data storage media, such as a computer's local disks or a database. A file system plug-in creates a file object that components such as file format plug-ins use to access the requested file's data. These system-standard file objects thereby create a virtual file system for accessing file data without regard to data location or storage format.

File format plug-in

Converts a data type from its native format to a packet format that can be streamed. File format plug-ins can also include Adaptive Stream Management rules to change stream data, such as bandwidth, rating, and language.

Rendering plug-in

Receives streamed packets and renders them for playback of a given data type on the Helix DNA Client computer. Every file format plug-in has a corresponding rendering plug-in. The rendering plug-in utilizes the audio and video services provided by the client core.

Generic plug-in

Any plug-in that does not fit in any of the previous categories is referred to as a generic plug-in. Examples of these are view source plug-in, which displays SMIL mark-up and clip information for the content it is playing, or a visualization plug-in that displays audio information in various visual formats.

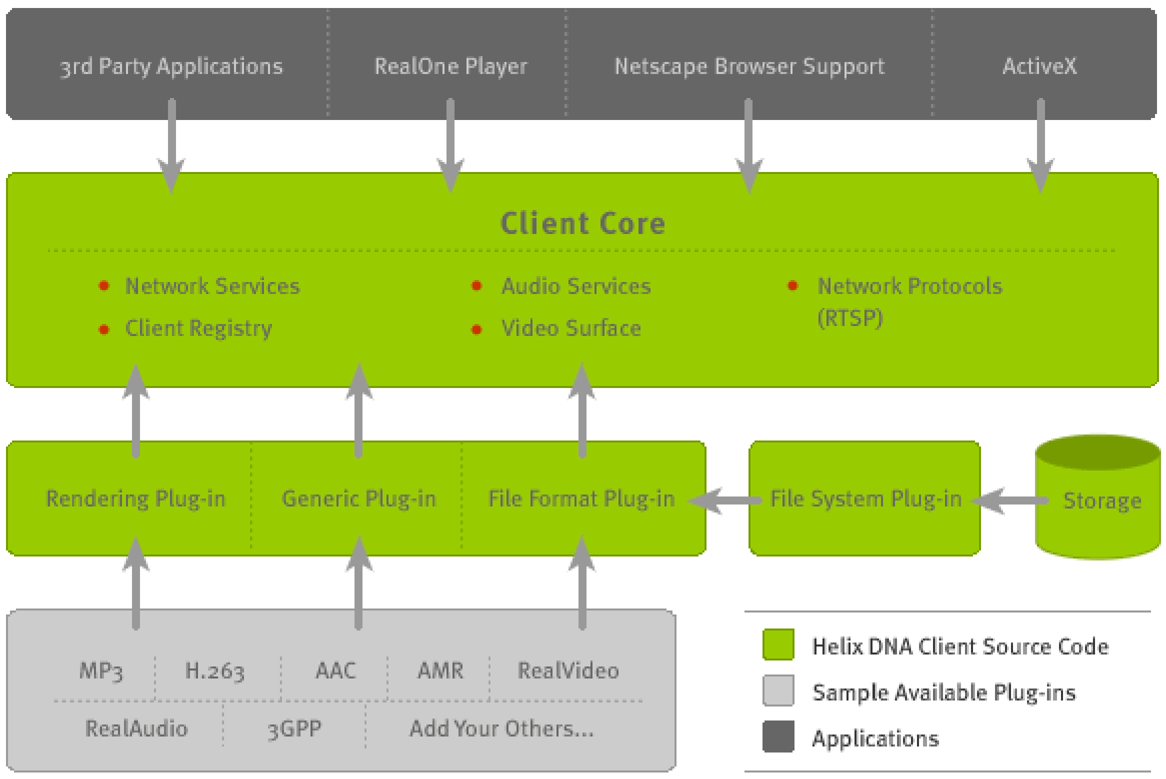


Figure 4.2.39 – Helix DNA Client's architecture.

Helix DNA components architecture is similar to COM, used on DirectShow. Practically only non-proprietary video formats are bundled, but different ones can be developed or purchased.

Helix DRM is a set of tools to a secure delivery platform of live and on-demand media content, constituted by these major components:

- Helix DRM Packager, prevent unauthorized use of content distribution via streaming or download;
- Helix DRM License Server, manage, authorize and report content transactions;
- Helix DRM Client, download and streaming playback of secure formats.

An innovating feature was recently implemented, entitled Synchronized Multimedia Integration Language (SMIL) and is now supported by Helix, introducing these advantages:

- Presentation layout and timing - SMIL allows the arrangement and manipulation of elements of a video, audio or graphics animation presentation to play simultaneously or on a specific time sequence;
- Tailoring a presentation for different audiences - SMIL can stream different clips to audiences based on criteria such as language or available bandwidth, all accessible with just one hyperlink. When a URL is opened, the Helix DNA Client-based application would read the options in the SMIL file and choose the appropriate presentation;
- Flexibility in media organization - A SMIL file lists a separate URL for each clip so presentations can be put together using clips stored on any server. This allows SMIL to eliminate the need to merge multiple clips into a single streaming file.

Licensing is under either RCSL or RPSL, the former for commercial use and the latter for research and recreational use. The commercial license will be available for organizations that develop applications for distribution purposes. Under that scenario, companies will have to pay $0.50 per unit for licensing until a cap of $5 million a year for the client component; the producer component is free of charge. Data about the server is only available on request.

RealNetworks numbers point to more than 135 million devices deployed (most of which thanks to the Symbian mobile operating system) and 125 thousand developers, worldwide. Grants are available to drive research and development, in an attempt to increase these numbers even further.

The source code for a simple player found during the research within the standard Helix DNA Client's SDK [Rea08b], was considered excessively large to be included.

[Rea08a]

4.2.4 Media Foundation

Introduced with Windows Vista, this framework introduces some advantages over DirectShow. DirectX Video Acceleration (DXVA) 2.0 offers more efficient video acceleration, with more robust, streamlined video decoding, and extended use of hardware in video processing, compared to the previous version. With DXVA 2.0, Windows can handle some of the most demanding high-definition content with high quality and improved glitch resilience.

In addition, colour-space information is preserved throughout the video pipeline: users can enjoy video content with full fidelity. Preserving that information also reduces unnecessary colour-space conversions, which frees more cycles to process demanding HD content.

The enhanced video renderer (EVR) provides better timing support, enhanced video processing, and improved glitch resilience.

All the following drawbacks of the previous platform have been eliminated from this newer framework:

- The graph in DirectShow tends to be static. Implementing dynamic graphs and format changes is a complex task;
- The threading model for DirectShow filters is complex and requires a thorough understanding to implement correctly;
- Filters cannot be used easily outside of a DirectShow graph, which ties them to the DirectShow pipeline;
- DirectShow does not readily support protected content.

Much advancement has been made, but this middleware is still very immature – it does not provide some important features such as media capture and editing, as well as some popular media types playing, and it involves a somewhat lower level programming than DirectShow.

It encapsulates the same architecture behind DirectShow, as its core, adding Digital Rights Management on top of it, and improving its inner workings.

Media sources read files or streams onto *Media Foundation Transforms* (MFT) that process them and delivers them into *media sinks*. If the content is copyrighted then the source *filter* is replaced by Input Trust Authority and the sink by Output Trust Authority. *Media Session* controls the sequences of filters, or *topologies*, being hardware level functions relegated to the platform layer.

New filters implemented today should be made in the form of MFTs, since they use the most advanced features of Media Foundation and are still compatible with DirectShow. DMOs and MFTs use the same basic architecture, so updating a DirectShow DMO filter should usually be a straightforward process. If only a "regular" DirectShow filter needs to be updated, some harder reengineering might be necessary.

Video editing is not the primary focus of MF. If a wrapper were to be made that encapsulated regular DirectShow filters, to allow MF to treat them as MFTs, Media foundation would have a very big push towards convincing developers to leave DirectShow.

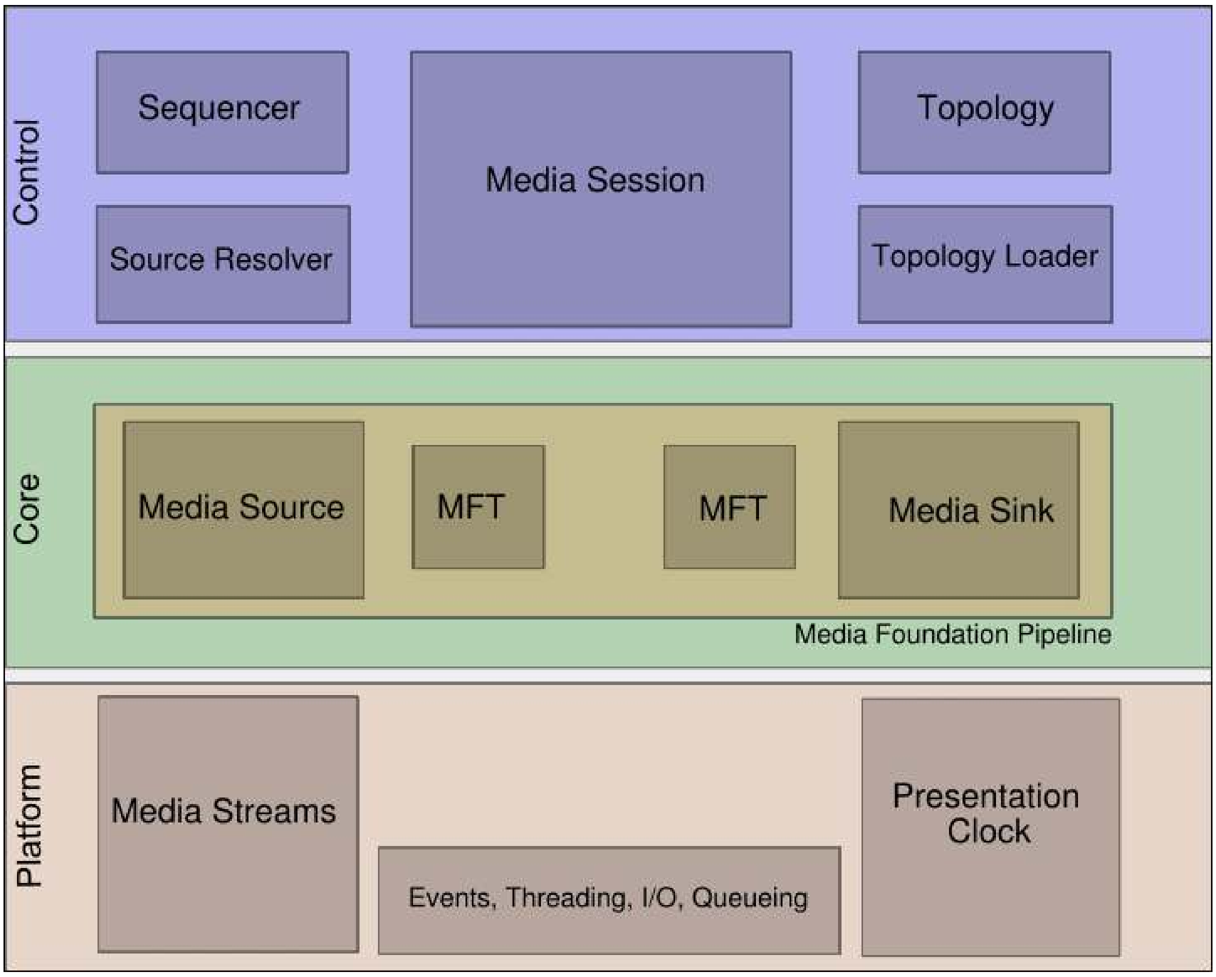


Figure 4.2.40 – Media Foundation's architecture.

Not ready yet to begin serious mass implementation. As of now, it requires more low level programming than DirectShow, although more high-level tools are expected to be released.

Documentation, support and communities are still lagging behind, especially because there was not much preoccupation in preparing the initial change from DirectShow. This is understandable, since the previous framework dominates the market completely and will not lose its leading position any time soon.

Besides the official documentation [MSD08c] and MSDN forums, some extra help and guidance can be found at [MFD08].

4.2.5 QuickTime Framework

Apple's QuickTime is the most widely used cross-platform multimedia framework available today. It is important to separate all concepts associated with the term QuickTime: the wrapper file format (considered in Video Container Formats chapter 2.3), the multimedia framework discussed here and all other software made by apple, based on this framework with similar names.

The architecture behind QuickTime middleware has practically remained unchanged since its first release in 1992. If DirectShow influenced most of the modern frameworks, QuickTime provided the original ideas for Microsoft's framework. The reason the same architecture resisted the pass of time is a very modular and well-structured design that allows some very complex tasks.

On the top level, *Tool Sets,* such as Movie Toolbox or image compression manager, support a spectrum of operations on media, such as playing a video, transcoding an audio file or applying one or another type of filter to some media. At a lower level, *Components* perform all different kinds of functions, and are selected and controlled by the Tool Sets. QuickTime often has multiple components of a given type. For example, QuickTime has many image decompressor components. They all have the same type: 'imdc'. Each 'imdc' component has a subtype that specifies the kind of compression it understands, such as JPEG, and a manufacturer code that distinguishes among components of the same subtype. For example, the image decompressor for JPEG supplied by Apple has the type, subtype, and manufacturer codes of 'imdc', 'jpeg', 'aapl'.

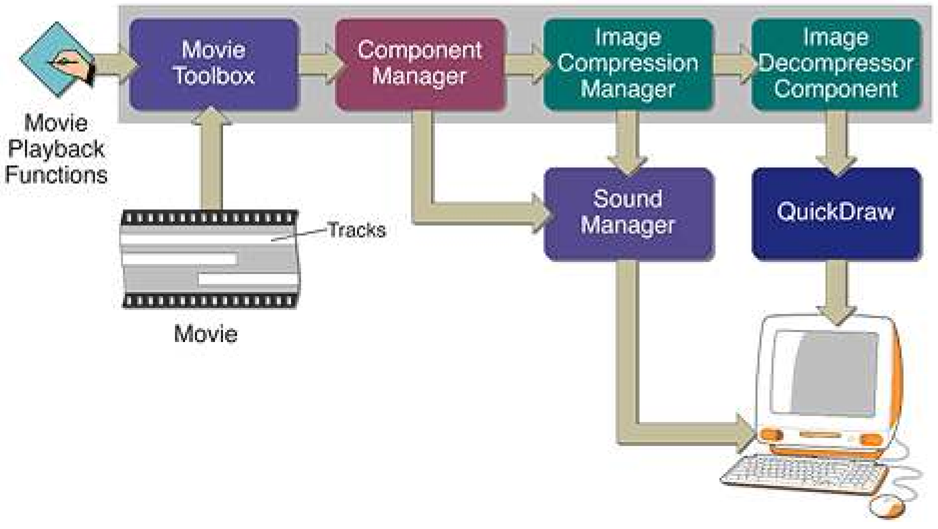


Figure 4.2.41 – QuickTime Framework's architecture.

Components work on a plug-in basis making it simple to incorporate your own implementation for a feature not covered by the existing ones. This way the framework can support more formats, either by implementing them or by buying additional plug-ins to Apple itself or to third party companies. The component's code can be available as a system wide resource or in a resource that is local to a particular application. Each QuickTime component supports a defined set of features and presents a specified functional interface to its client applications. Applications are thereby isolated from the details of implementing and managing a given technology. For example, you could create a component that supports a certain data encryption algorithm. Applications could then use your algorithm by connecting to your component through the Component Manager, rather than implementing the algorithm again.

The handling of a media file or stream is done through a data structure called movie that describes what media to present, where the media samples are located, and how and when to present them (duration, sequencing, compositing, layering, rotation and scaling, sound volume, and so on). A QuickTime movie may contain several tracks. Each track refers to a media that contains references to the movie data, which may be stored as images or sound on hard disks, removable media, network addresses or other devices. The data references constitute the track's media. Each track has a single media data structure. A movie data structure is always internally used, independently of the file type, since every aspect of the framework is oriented towards the QuickTime wrapper format.

QuickTime is starting to turn its attention towards open standards rather than proprietary coding formats, following the trend that can lead to a much more open market and rapid development of new and interesting ideas.

It performs worst on Windows OS, compared to MacOS, since its architecture is Mac oriented, making rendering a more complex task on Windows due to differences between the operating systems.

The licensing terms regarding the distribution of software built upon this framework are not present on product’s site, and through some interactions with Apple’s licensing team I could only come to the already known conclusion that the developing of software based on QuickTime is free of charge; I implied that there are costs associated by its distribution.

There is little information regarding this framework besides that which is available through Apple’s site and online documentation [APL07], that makes it a daunting task to find solutions for unusual problems. This can be explained by the seemingly over-protective attitude of Apple towards their products, even open ones.

4.3 Other Frameworks

More and more frameworks are coming into play, since the concept has been proven to work quite well. With that in mind, making an in-depth study of all of them would be extremely time consuming and unreasonably out of scope, but there are a few that even though they are not as important nowadays, as the ones mentioned before, they played or will play an important role in a near future.

4.3.1 Java Media Framework

Sun's Java has an excellent PC coverage throughout the world. Being practically platform independent and open sourced, a multimedia framework written on it could reach, in a standard, consistent way millions of systems, giving it a chance to become one of the most widely used multimedia frameworks. Having that much potential, it is hard to understand what went wrong after its release in 1997, but some of the reasons include a complex architecture, based on a state machine with many different states and the lack of support for the most frequently used video and audio formats.

Incidentally, it has been "dead", as far as Sun's support is concerned, since 2004, meaning there were no updates to the source code, made public. Still some community driven efforts have been made to resurrect this middleware solution from its ultimate demise [FMJ07].

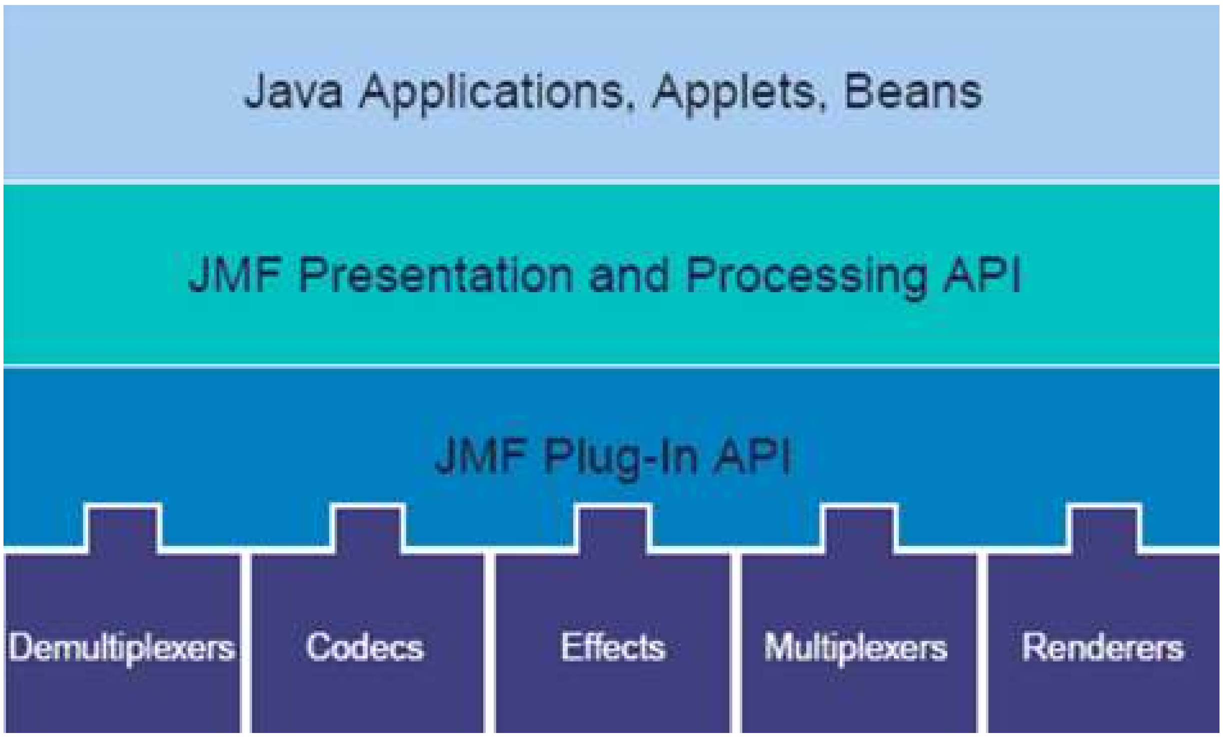

Java Applications, Applets, Beans
JMF Presentation and Processing API
JMF Plug-In API
Demultiplexers
Codecs
Effects
Multiplexers
Renderers

Figure 4.3.42 – Java Media Framework's layers.

4.3.2 Network-Integrated Multimedia Middleware

Since there is a strong trend towards networked multimedia devices, like networked cameras, audio devices, PDAs, cellular phones, PCs, etc., centralized approaches like most multimedia frameworks today, are becoming inadequate in some cases. In contrast, the Network-Integrated Multimedia Middleware (NMM) offers a multimedia architecture, which considers the network as an integral part and enables the intelligent use of devices distributed across a network.

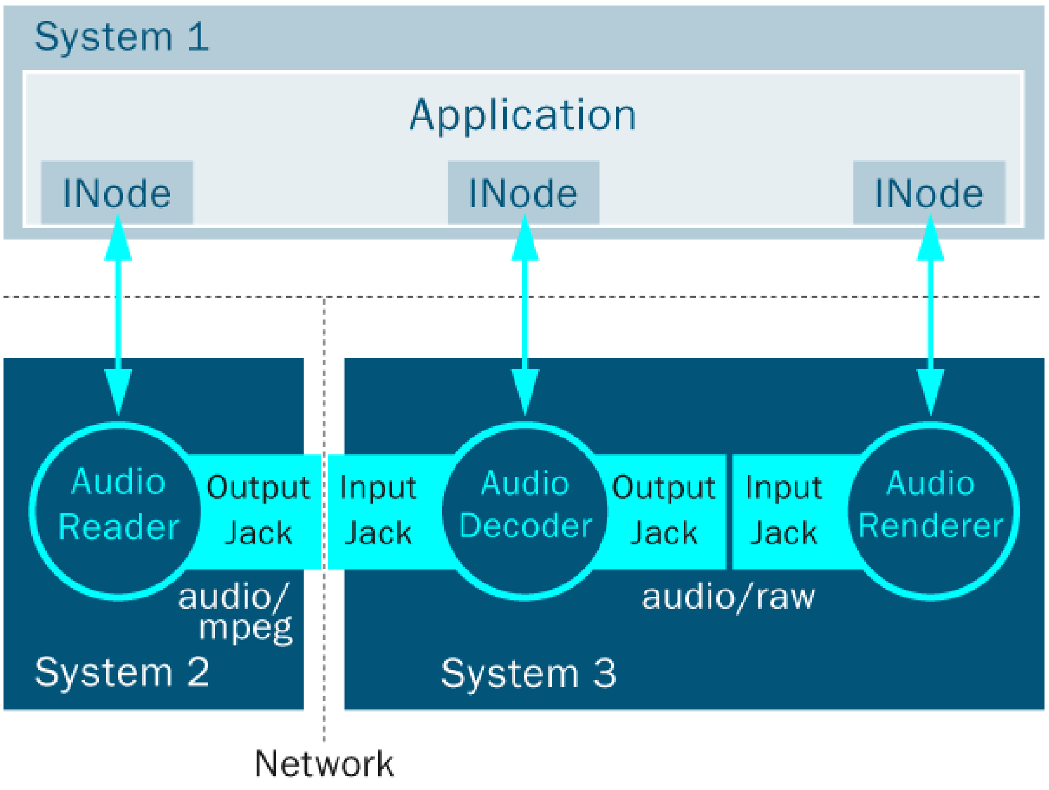


Figure 4.3.43 – Network-Integrated Multimedia Middleware's architecture.

NMM is available for Linux, Windows, MacOS, as well as other operating systems. Its unified architecture offers a simple and easy to use interface for applications to integrate multimedia functionality. Therefore, it can be used as an enabling technology for locally operating multimedia applications, but more importantly for all kinds of networked and distributed multimedia systems - spanning from embedded and mobile systems, to PCs, to large-scale computing clusters.

Its main feature is the possibility to, out of the box and in an uncomplicated manner, spread the media handling processes over different systems, permitting new solutions to be achieved easily and efficiently. For example, it is possible to read a file from a wireless storage device, remote control the playback from a cellular phone and have the output sound "follow" you around the different rooms of a house through just a few lines of code, and the appropriate hardware. Another example (also possible with Gstreamer through an existing plug-in) is to develop a video wall easily: splitting a video source among several screens connected by several PCs networked together, to exhibit a larger display area.

Each system keeps its own clock synchronized with all others, providing a simultaneous playback. Different formats and features are supported via plug-ins.

[Mot08]

4.3.3 MPEG-21

The MPEG group created this standard under the name of multimedia framework, although it is not of the same ilk of the previous frameworks. In fact, this framework is not able to reproduce any type of media and it does not have a fixed implementation; it is a set of tools and standards to orient the design and implementation of all media related software that will guarantee a consistent environment in which media to be transactioned. It is meant to be enclosed within every step between capture and consumption of media, thus facilitating every kind of media transaction by maintaining many types of metadata, additional data and focusing on copyright protection.

There are two kinds of entities declared: digital items and users. Digital Item is a combination of resources (such as videos, audio tracks, images, etc), metadata (such as MPEG-7 descriptors), and structure (describing the relationship between resources). Users can be any entity or person involved in the aforementioned transactions: companies, directors, packagers or consumers.

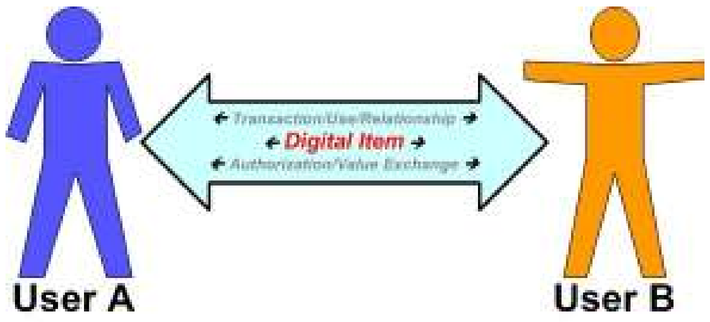


Figure 4.3.44 – A MPEG-21 Transaction between different users.

MPEG addresses the following key items in order to provide an interoperable multimedia framework. These areas are:

- **Digital Item Declaration**, a uniform and flexible abstraction and interoperable schema for declaring Digital Items;
- **Digital Item Identification and Description,** a framework for identification and description of any entity regardless of its nature, type or granularity;
- **Content Handling and Usage,** provide interfaces and protocols that enable creation, manipulation, search, access, storage, delivery, and (re)use of content across the content distribution and consumption value chain;
- **Intellectual Property Management and Protection,** the means to enable intellectual property rights on content to be persistently and reliably managed and protected across a wide range of networks and devices;
- **Terminals and Networks,** the ability to provide interoperable and transparent access to content across networks and terminals;
- **Content Representation,** how the media resources are represented;
- **Event Reporting,** the metrics and interfaces that enable Users to understand precisely the performance of all reportable events within the framework.

MPEG-21 is suitable for many types of applications. Some of the use cases that the standard has been developed to satisfy include digital libraries, broadcast usage, Publishing, music/video releases, asset management, cataloguing in publication, trade transactions and content filtering.

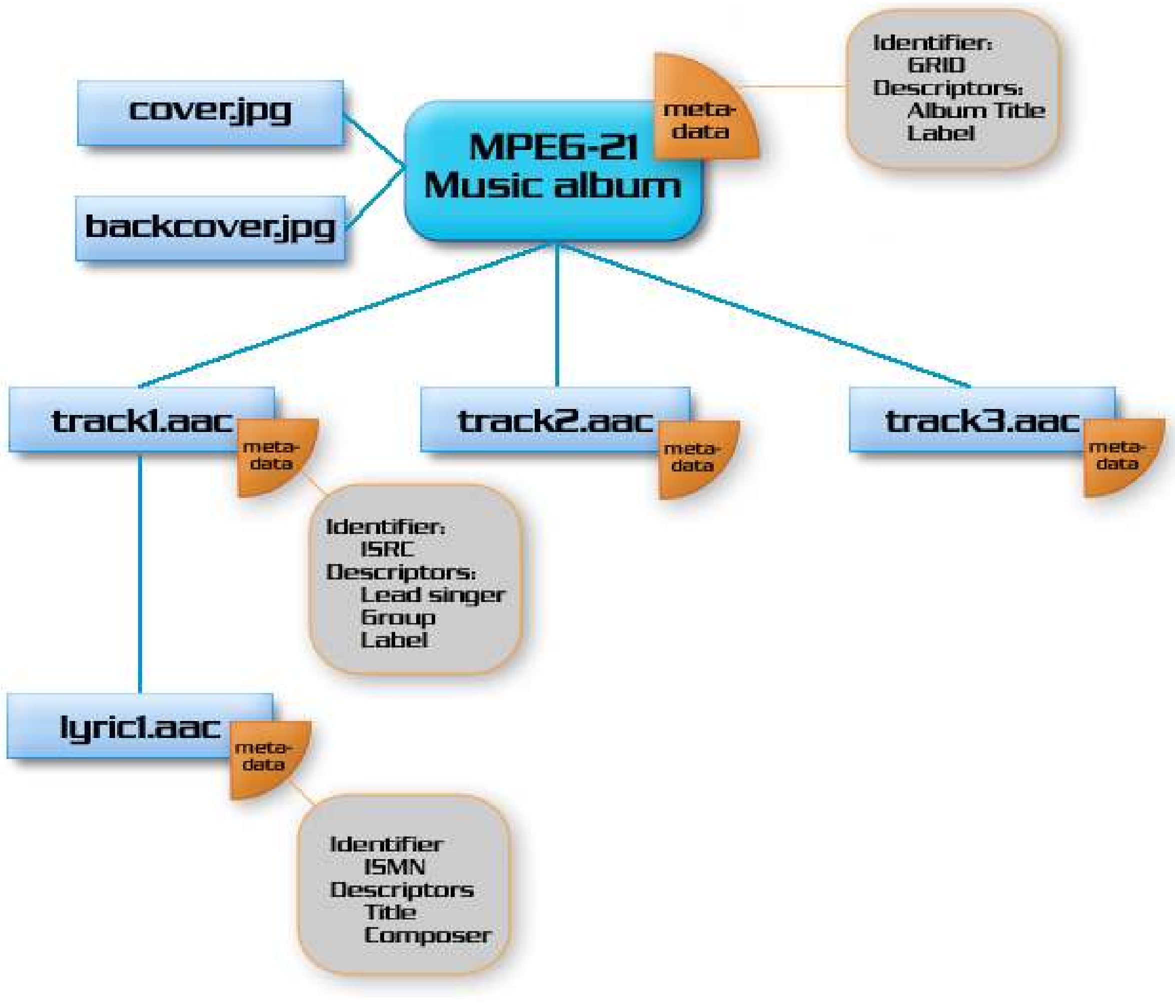


Figure 4.3.45 – Example of a Digital Item: a music album.

[Tim07]

4.4 Framework Comparison

In order to adequately choose on which media platform to implement a new video handling software, several items should be taken into account, ranging from how many users already possess, use or develop a given framework to its various codecs support. The following tables condense such information in an easily readable arrangement.

Some of the presented information is based on properties demonstrated on several applications that draw on these frameworks as a basis for their media handling processes and the middleware developers' claims. Ideally, it would be supported by a well-defined benchmark applied to all frameworks, but such exercise has never been done before and it was not possible to undergo during the brief period dedicated to this broad project.

4.4.1 Frameworks' Status

First, a study of the market presence, current and future projection, the state of the frameworks development and an analysis on their features and resource efficiency.

	Supported Platforms	Distribution	Future Use	Maturity	Performance
DirectShow	Windows in between 98 and Windows 7[2]	Every Windows from 98 to Vista	Stalled	Becoming Obsolete	Less good
Gstreamer	Linux, OpenBSD And Windows	Mainly Linux	Increasing	Immature Version 0.10	Very good
Media Foundation	Windows Vista and Windows 7	Every Windows Vista	Slowly increasing	Young	Good on SD, slow with HD
QuickTime	MacOS and Windows	Every Mac	Stalled	Mature Version 7.4.1	Good
Helix DNA	BSD, Linux, MacOS, Symbian and Windows	Strong on mobile	Increasing	Gold Version 1.0.9	Good

Table 4.4.8 – Main frameworks' general characteristics.

Gstreamer Linux support includes Fedora, Red Hat, Debian and Gentoo versions. Helix DNA support varies depending on the considered module: the Helix Client is released for Linux, Solaris, SunOS and Symbian; Helix Producer works on Windows, MacOS X and Linux; and Helix Server operates on Linux, Solaris and Windows.

The distribution information and future use were based on the latest statistics concerning operating systems usage [Ref08], because DirectShow, Media Foundation and QuickTime frameworks are shipped embedded with all Windows since 98, Windows Vista and all Mac OS respectively. Truthfully, plenty of Windows users work with QuickTime's framework or even Gstreamer, Linux users vary from Helix DNA to Gstreamer, a few MacOS users utilize Helix DNA and there are other markets besides the computer's, such as the mobile devices'. Such discrepancies have been taken into account by estimation based on the number of projects each framework has on a given platform and the relative measure of its success.

DirectShow is currently the most used multimedia framework but its architecture began, in

[2] Windows 7 is Microsoft's codename for the upcoming operating system that will succeed Windows Vista expected to be released in the beginning of the year 2010

the later years, to show signs of aging when facing the newer codecs requisites, as a consequence of the lack of Microsoft's interest in updating it. Still many companies' businesses are founded around this framework, which maintains a very active community and a long list of derived products. Apple's even older framework still keeps up with the new generation of codecs thanks to a finely tuned architecture that allows complex operations and to frequent upgrades. Media Foundation is still very "green" and does not possess many available codecs, which is crucial to this kind of middleware. It is currently present in many computer systems, but not widely used.

Maturity assessment was based on the status and activity level of the development of each framework, how old they are, their version number and their features related to the current state of multimedia handling requirements. Helix DNA is currently just past its first solid release and is being used in several projects; there is still a lot to be done but this feature rich framework has the possibility to firmly implant itself on the market. Gstreamer is a melting pot of ideas and concepts, and adopted the philosophy of slowly but steadily developing itself. Instead of rushing to a major release, preparations are being made to turn this framework into a stable, reliable foundation; because of this, it is taking longer to consolidate its core, which is at the same time preventing serious investments on it, but that resolution may pay off on long run.

Performance rating is in terms of speed, memory and processing requirements and available features; supported by the reviews of developers that worked with the frameworks and through the analysis of the architecture sustaining them.

4.4.2 Development

The cost of distributing software built on top of these frameworks, the required coding languages for implementing them, the number of companies and developers using these middleware solutions and the documentation explaining how they work and how to program them is discussed in this chapter.

	Licensing	**Coding Languages**	**Current Development**	**Documentation**
DirectShow	None required	Language agnostic	Large, not increasing	Thorough documentation
Gstreamer	LGPL plus proprietary codecs	C, Perl, Python and Ruby [3]	Medium, very active community	Very well documented
Media Foundation	None required	Language agnostic	Small, slowly increasing	Scarcely documented
QuickTime	Proprietary	C, Objective-C, C++ or Java[4]	Mainly from Apple	Good documentation
Helix DNA	RCSL or RPSL	C++	Medium and active community	Documented but unorganised

Table 4.4.9 – Main frameworks' development status.

3 Java, .NET and C++ bindings are currently being developed

4 Also Perl, Python and Ruby through extensions to Cocoa, MacOS's Objective-C's framework

There are no development costs for any of these frameworks, but the distribution is only free of charge for Microsoft's frameworks. The use of plug-ins, namely codec decoders and encoders is often paid for. Gstreamer packs the most liberal license, enabling software to be sold freely, as long as it links dynamically to the framework; otherwise, it requires the software to ensue the same LGPL license. Helix DNA distribution follows their own licensing terms that are detailed on the previous 2.3 chapter. QuickTime contains the biggest amount of included formats but does not specify, at any level, the costs of distribution.

Thanks to the employed COM architecture, both Media Foundation and DirectShow are independent of any coding language; still the most used is C++.

Current development indicates the communities and the work being done on each framework: DirectShow has a huge community but part of it is starting to transfer to the young Media Foundation community. Gstreamer has a considerable and very active community, as opposed to QuickTime that tends to rely on its own work force. Helix DNA has a sizeable and active community.

Media Foundation's documentation is one of the causes holding it back from a larger adoption since its still incomplete. Both DirectShow and QuickTime benefit from their older age with a very thorough documentation. Gstreamer is exceptionally documented on all aspects ranging from the framework's architecture to the implementation of new elements. Helix DNA's documentation focuses on the architecture and lacks some proper organisation.

4.4.3 Supported Formats

In the ensuing tables, we will look at which formats are supported by which frameworks. Third-party software components are reusable software developed to be either freely distributed or sold by an entity other than the original vendor of the development platform. Additional formats may exist, distinct from those shown here, from third parties that I did not have contact with, since there is no compilation of all existing plug-ins for these frameworks.

	DirectShow	**Gstreamer**	**Media Foundation**	**QuickTime**	**Helix**
AAC	Third Party	Supported		Supported	Decoding
APE	Third Party	Supported			
Apple Lossless	Third Party			Supported	
ATRAC	Third Party				
FLAC	Third Party	Supported			
MIDI	Supported	Supported		Supported	
MP3	TP E and S D	TP E and S D	Supported	Supported	Supported
PCM	Supported	Supported	Supported	Supported	Supported
OGA	Third Party	Supported			Supported
TAK	Third Party				
TTA	Third Party	Supported			
Wav	Supported	Supported	Supported	Supported	Supported
WMA	Supported		Supported		Decoding

Table 4.4.10 – Supported audio formats.

"E" stands for encoding, "D" for decoding, "S" for supported, "TP" for third-party and blank for not supported. Not all possible audio formats are presented here but only the most popular and relevant formats used in both professional and consumer markets. The same applies for the video and container formats displayed in the next two tables.

Regarding the professional market, Pulse Code Modulation (PCM) and Waveform (WAV) are the most used formats, which are frequently extended by Audio Engineering Society (AES3) and Broadcast Wave Format (BWF), respectively. MPEG Layer 3 (MP3) is undoubtedly the most popular consumer audio format, spawning millions of devices capable of handling it.

	DirectShow	**Gstreamer**	**Media Foundation**	**QuickTime**	**Helix**
Cinepak	Supported	Supported			
Dirac	Third Party				
DV	Supported	Supported		Supported	Encoding
H.261	Supported	Supported		Supported	Supported
H.263	Supported	Supported		Supported	Supported
H.264	Supported	S D and TP E		Supported	Supported
JPEG 2000	Third Party	Supported		MacOS only	
MPEG 1	TP E and S D	Third Party		Supported	
MPEG 2	S E and TP D	Third Party		Third Party	
MPEG 4	Supported	Third Party		Supported	Supported[5]
QuickTime	Third Party	Decoding		Supported	
Real Video	Third Party	Third Party			Supported
Theora	Third Party	Supported			Supported
VC-1	Supported		Supported		Supported
WMV	Supported		Supported		Supported

Table 4.4.11 – Supported video formats.

Within the professional video industry, the most relevant formats are JPEG 2000 in its Motion JPEG 2000 variant, MPEG-2 as the DVD and digital broadcasting technology, MPEG-4 as the new leading video format, especially in the efficient H.264 variant (also on the consumer market), and DV in its DVCPRO and DVCAM versions.

[5] Source code is provided, but not its binary form, due to copyright restrictions.

	DirectShow	Gstreamer	Media Foundation	QuickTime	Helix
3GP	Third Party			Supported	Decoding
AIFF	Supported			Decoding	Decoding
ASF	E and D	Third Party	Supported		Decoding
AU	Supported			Decoding	Supported
AVI	E and D	Supported		Decoding	Supported
MKV	Third Party	Supported			
MOV	Third Party	Decoding		Supported	Supported
MP4	TP E and S D	Third Party		Supported	Supported
MXF & GXF	Third Party				
OGG	Third Party	Supported			Supported
Real Media	Third Party	Third Party			Supported
VOB	Supported	Third Party			

Table 4.4.12 – Supported container formats.

On television and cinema markets, the historically relevant wrapper formats are VOB used on DVDs and MOV, as a versatile digital container. Nowadays the market is shifting towards more powerful and comprehensive formats as MXF, closely related to AAF, and GXF.

On the consumer market AVI, the WMV versions of ASF and MOV have dominated the market, besides the obvious VOB format from DVDs. Presently MKV is taking over AVI's place, by packing much more features and eliminating some commonly know problems of its predecessor. 3GP is widespread on mobile devices as MP4, based on the MOV format, and is steadily increasing its market presence.

4.5 Market Analysis

Indubitably, DirectShow is the most used multimedia framework throughout the world. All that is needed to realize this is to check the usage of operating systems and observe that nearly 90% of all personal computers run Windows operating systems [Ref08]. Media Foundation, the new media framework iteration from Microsoft, exists in a small percentage of those computers, but due to an obvious lack of supported formats, its predecessor is still used for almost the totality of the media handling tasks. Surprisingly enough, this panorama is not bound to change in the next 2-3 years. DirectShow is the base of too many software solutions, both professional and consumer oriented, and its intended replacement is more focused on playback rather than the rest of the media handling functions.

Outside Windows operating systems, almost every Mac computer uses QuickTime, and all have it installed by default. Being the second most used OS and an important name in the video industry gives it a prominent position over other frameworks. The current tendency of MacOS systems sales increase is going to boost QuickTime's usage.

Linux systems have two main platforms – Helix Project and Gstreamer - that will compete for any market left vacant by the previously stated frameworks. Helix is more established in the market, having already many projects with a few companies, fruit of RealNetworks support. Gstreamer is progressing slowly, but its devoted focus to detail on the architecture, will result in a well-structured and stable platform, with many innovative capabilities and relatively easy to implement. Even so, it still has a long way to go before becoming a serious alternative due to the slow solidification of its core elements.

Ultimately, the choice of this or that multimedia framework depends on the number, and quality, of the video and audio codecs. With the current trend of standardising media compression formats this becomes a slightly less problematic issue, but there are still licences to be paid in some cases, and there will always be the need for developers willing to implement new and old formats.

4.6 Final Thoughts on Multimedia Frameworks

It is stimulating to watch major companies starting to opt for open sourced applications and open standards, because it enables knowledge to be shared and developed in a much faster fashion, much more widely. Furthermore, taking measures to enable their software to run in multiple operating systems is a step closer to bringing free operating systems to a similar level as their paid counterparts, while promoting the reuse of code and a greater interoperability.

The biggest obstacle that a framework faces is the lack of support for media transformations and more importantly for media formats. Directly related, is the distribution of the platform that influences the investment on development for the framework. All the studied middleware solutions have the potential to perform all desired features capably, provided there is a plug-in written for it, or someone willing to implement one.

Most users will not leave their already working systems (DirectShow) in the short run, but will surely venture into other kinds of frameworks if they offer sufficient performance and feature gains. On the user market, Media Foundation will surely benefit from an increasing Vista distribution and the forthcoming Windows iterations, making it a plausible alternative to the implanted frameworks, if sufficient formats are supported and the documentation for its development is mature enough.

Professional video market will surely stick to the stable and proven technologies, with large turn around cycles, thus venturing onto neither Gstreamer nor Helix DNA, but might turn its attention to Media Foundation, due to its improvements over DirectShow. One such advancement is a better DRM support that will play a major role in the upcoming audiovisual material financial transactions, crucial to the entertainment business, especially taking into account that we are headed for a massive multimedia exchange market. If there is a niche of the professional market that wants to turn you're their backs away from Windows operating systems, surely Gstreamer or Helix DNA are the way to go, since both also have a good support for copyrighted material, and not as draconian as in Media Foundation.

My recommendations for developers already using DirectShow planning to implement for Windows in the next 2-3 years (expected date for the release of the new Windows 7) is to keep use DirectShow but if developing new filters upgrade to the new DMO architecture, which can later be easily upgraded into Media Foundation MFTs. Later implementations for performance hungry applications should be made with Media Foundation in mind, implementing DMO/MFT hybrids if enough knowledge of the new framework is still not implanted enough.

If the goal is to have multiplatform capabilities use Helix DNA; it provides plenty of features, is well delimited and is stable enough to allow long term projects.

If no prior knowledge of multimedia frameworks exists, and speed is of the essence, Gstreamer is the framework to choose, because of its simple architecture and excellent documentation. Also it is the most innovative framework, merit of its active community. The only drawback is the fact that its implementation is still not static, but several projects already exist based upon it. It is very efficient, performance wise.

QuickTime is the best choice for overly complex features, but is to some extent limited to its own format (.mov). It bundles many features and codecs in the standard version, without the need to purchase additional software (although this option still exists).

5. Practical Implementation

Due to the abundance of programming languages, used coding styles and platforms, it was not possible to code all frameworks pragmatically. The only middleware I had programming contact with was DirectShow. The company in which this document was produced, bases many of its products on this framework, because of its reach, existing libraries, communities and its flexible architecture. Taking advantage of the existing experience in the company on dealing with DirectShow based media applications, DirectShow was chosen to be used with C++ language.

In this context, one of MOG Solutions' clients, RTP (Radio Televisão Portuguesa, the first Portuguese public television broadcaster) requested additional support for WMV playback on the tool MOGPlayer [MOG08], since it was implemented only to read MXF and GXF files. It was required not only to open local files but also remotely hosted videos. MOGPlayer is an integrant part of several other tools, so to simplify its testing, it was encapsulated as an ActiveX object, that runs on Internet Explorer browsers.

The developing environment used consisted of Visual C++ .NET 2002 as IDE, MogBuilder (a customized version of Jam) as the compiler and Eclipse for browsing and committing to the CVS code repository, all on Windows XP. Additionally, GraphEdit was used for filter tracking and debugging.

Since all filters already exist for reproducing this video format, I had the chance to be introduced, somewhat gently, to DirectShow programming. The complexity level would have increased exponentially if writing a new filter were required, since it demands a much deeper knowledge of the framework, filter internal architecture (COM objects) and the specific video format decoding. Still it was not as easy as simply using *renderfile()* and letting *Filter Graph Manager* choose the correct filters for rendering both video and audio.

There are several filters with similar capabilities and these are differentiated from each other by DirectShow's merit system. The problem is that, what might fit one particular use, might not fit another, and DirectShow does not take that into account: it simply chooses the filters he considers most suited for playback. Added to that, some filters may have unexpected behaviours, so a careful selection is usually preferable. Each filter had to be added individually to the graph, from source filter, to decoders, to video and audio renderers, so their interfaces could be correctly exposed.

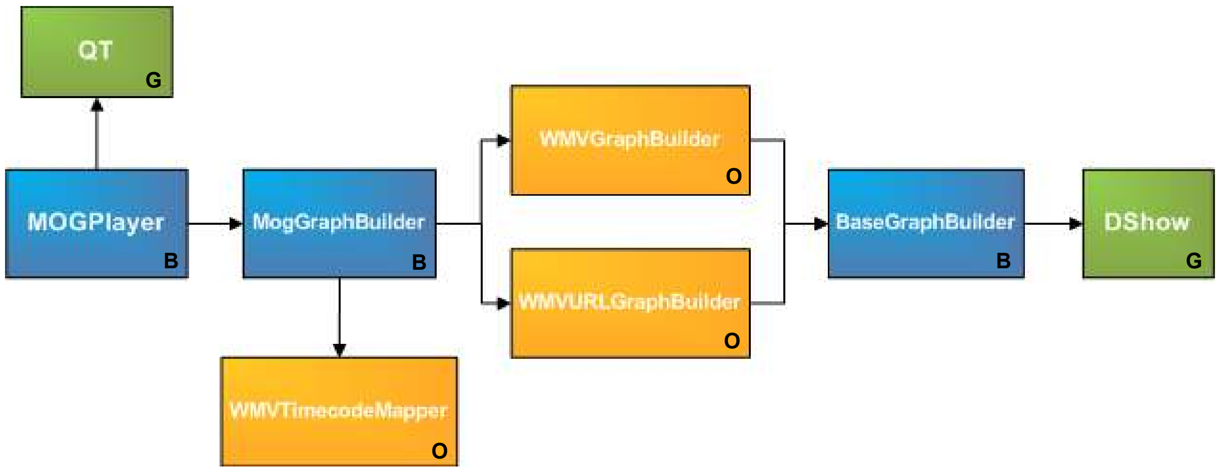


Figure 5.46 – MOGPlayer classes and libraries (letters added for black and white readability).

MOGPlayer's playback of media is fundamentally based on the classes show on Figure 46. The green colouring represents libraries external to MOG Solutions; blue already existing classes edited by me; and orange the newly created classes.

QT is the library responsible for all interface elements, and apart from learning how it worked, it was not within the span of my implementation to change or add anything to the appearance of the application. DShow is the header file, linking to the DirectShow API, which is responsible for all low-level system processes for loading, decoding and rendering the WMV files.

MOGPlayer comprises the application itself, and manages all requested actions through the QT interface. It uses the class MogGraphBuilder to control all processes related with handling the video and audio sections – it is in charge of all the logic behind the player. WMVTimecodeMapper is responsible for all time to timecode and vice-versa calculations.

WMVGraphBuilder and WMVURLGraphBuilder, both inherit from BaseGraphBuilder and are responsible for selecting the specific source filters for reading the file, and in this case, of dividing it into video and audio streams, since it acts as both source and parser. The difference between the two classes is just the location of the file: local or remote (the URL version). There were other existing classes that allowed more types of media to be handled, such as MXF and GXF, but with the use of multiple inheritance, changing their code was unnecessary.

BaseGraphBuilder acts akin to a layer on top of DirectShow, containing higher-level functions that are commonly used, much like an API. Most relevant to this work is its function *renderSplitterOutputPin*, that given an output pin from a filter, tries to complete the graph for the type of media present – it adds the *a priori* stipulated filters necessary to decode and render that media.

The filters' UUIDs (Universal Unique IDentification) were discovered using GraphStudio, a more advanced clone of GraphEdit that provides a graphical interface to the auto-rendering feature available in DirectShow. Some trials were made because more than one source filter was usable, but not all provided the necessary features: seeking and frame-by-frame playback.

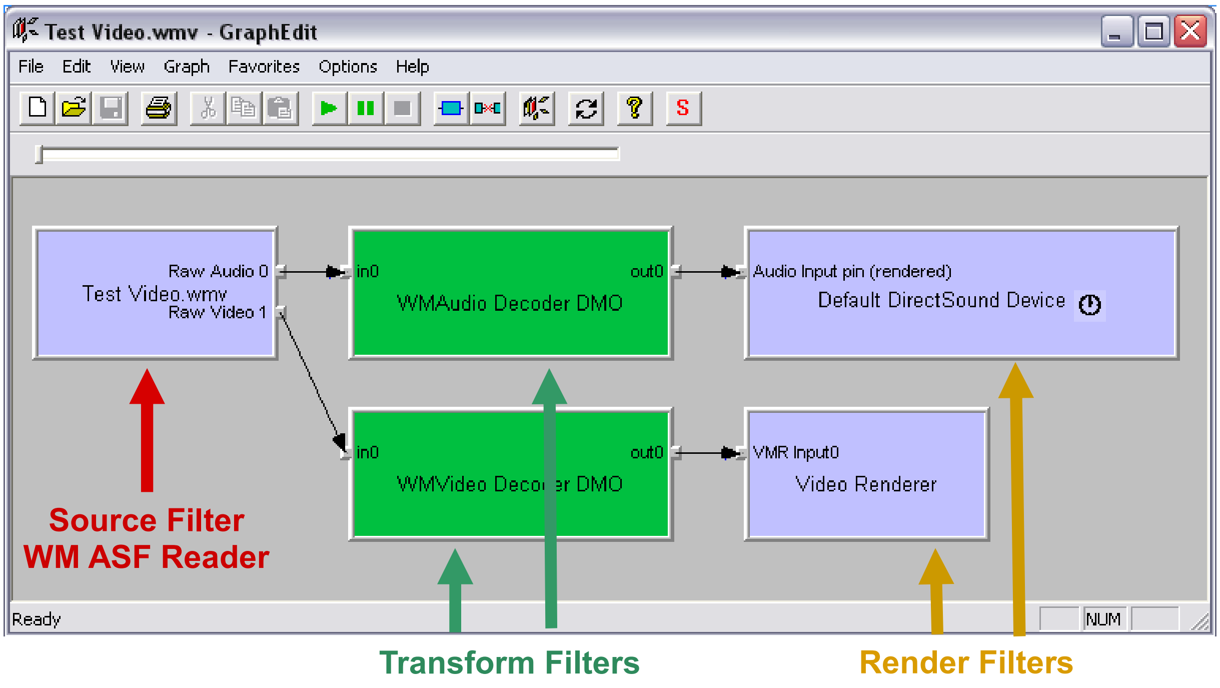


Figure 5.47 – GraphEdit's capture of the filters used in MOGPlayer for local WMV rendering.

Later, after a testing version was sent to the client, it was brought to my attention that seeking was not functioning correctly in a certain number of computers. As a first approach, more specific logging was added to try to obtain more information about the filter incompatibility, but all seemed to match exactly, between systems where it worked properly and those where it did not. After a couple of days of exhaustingly testing and debugging several aspects of the software, the problem was traced to an incompatibility of the default renderer with certain video accelerator cards, within some PCs. In order to bypass this complication some solutions were found, that due to their nature involving the installation of additional software on the client's machine, were set aside. At length, the solution was to use a different renderer for WMV files, that even though it has some minor aesthetic drawbacks, was more reliable during playback. The Default Video Renderer was replaced by the Video Renderer.

Later some additional features were asked for: correct support for mark in and mark out and timecode extrapolation (WMV does not natively support timecode). Mark ins and outs are used to select a clip from within a larger video, more precisely the timestamps at which it starts and end. Timecode is a standard to represent frame timing that instead of representing it in the form of Hour:Minute:Second:Millisecond, substitutes milliseconds for frame number that ranges from 0 to frame rate minus 1, in the form Hour:Minute:Second:Frame.

Before each release to the client, an install executable was made with Ghost Installer containing all necessary *dll*'s and other files. Then the implemented features were comprehensively tested in a clean install of Windows XP on an available test machine.

After all problems were found and corrected, positive feedback was sent from the client and MOG Player was ready for release, with n important new feature - WMV advanced playback.

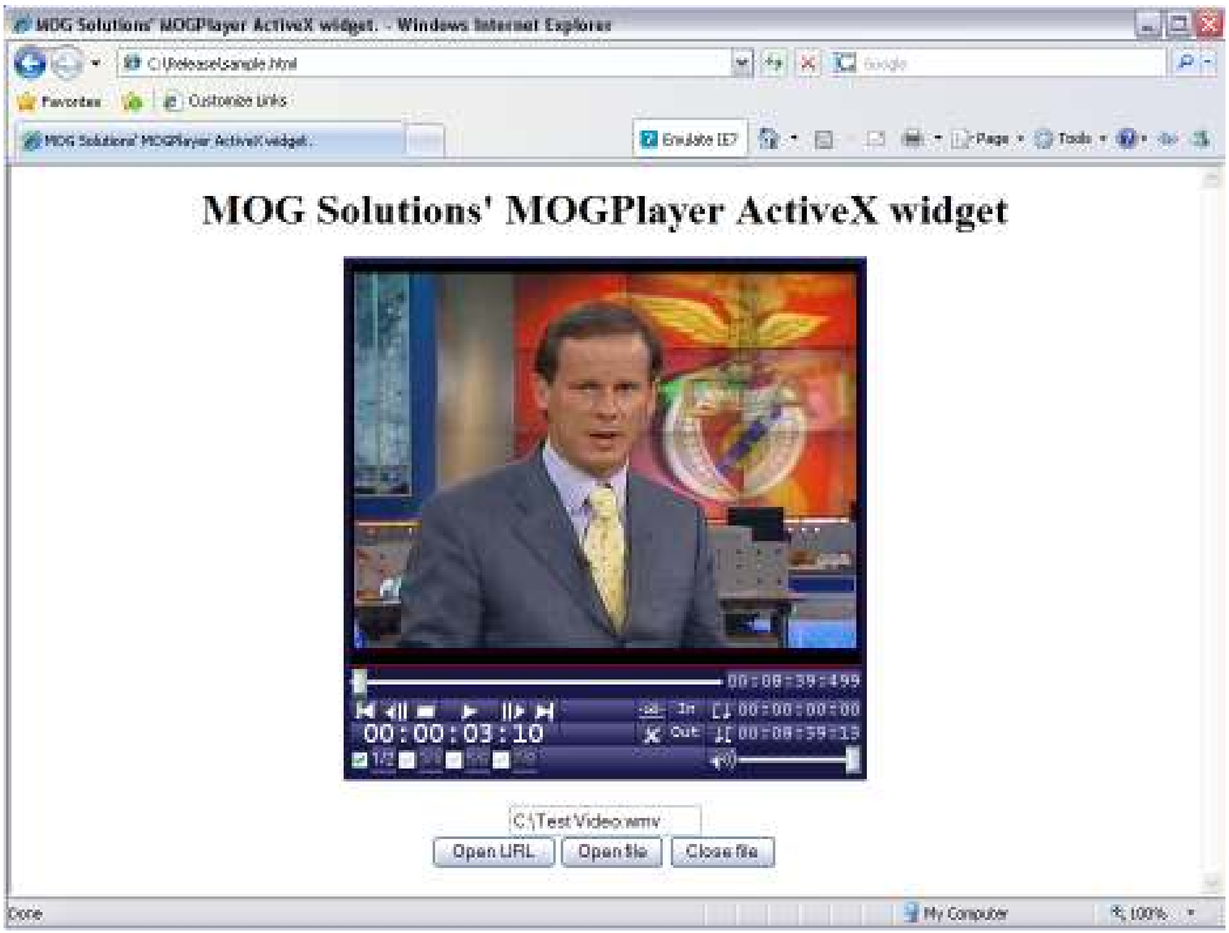


Figure 5.48 – Latest version of MOGPlayer reproducing one of the supplied test media.

6. Conclusions

In this chapter, I will disclose what new information has been created and summed up through the development of this document, the achieved results and propositions of future addendums to the produced product.

6.1 Original Work

Being this project an assessment of the state of the art of digital video compression, container formats and multimedia frameworks, most of the data referred here already exists in the form of books, sites and articles. My contribution consists in the introductions, conclusions and most explanations that were written with foundations on the initially made research and on the wise words of MOG Solution's workers that incorporate years of expertise, while programmers and marketers of professional video software solutions.

A great deal of documents has already been written about all the matters discussed on the Video Compression sections. Notably, one of the main sources for this work [Sym04] covers a large part of the aforementioned topics, but often on a very technical level, sometimes disregarding the implications of the actual implementation and market adoption. Here, the valuable input of experienced professional video software developers, add to the pertinence of this document's content. All exercises exemplifying the compression algorithms were devised by me, to maintain a coherent explanation throughout the chapters 2.1.1.1 Run-Length Encoding and 2.1.1.2 Entropy Encoding.

Most of the Video Containers section is a simplification of the norms and standards of the various container formats. The main interest lays on the introducing and concluding chapters that depart from the plain enumeration of features and specifications.

A couple of documents about these middleware solutions were found during the compilation of this work that merely scratch the surface of this extensive matter, both from a multimedia programming class lectured at the Finish Telecommunications Software and Multimedia Laboratory: [Her07] and [Vuo06]. To my knowledge, there are no, previously made, comprehensive comparison between the most important multimedia frameworks available today. There are indeed detailed works concerning each of the frameworks individually, but none to overview all the relevant alternatives.

Many applications exist, in numerous platforms, that relying on a multimedia framework, can output WMV video and associated audio. A significantly lesser number is capable of using advanced playback features, like frame-by-frame playback. While not being a groundbreaking achievement this development exercise made it possible for me to gain a much better insight on the complexities of using a multimedia framework as DirectShow.

A great care was taken to maintain, throughout this multi themed document, a consistent language and style, since the contained information was siphoned from a multitude of distinct sources, from various knowledge areas and different levels of formality and technical detail.

6.2 Results

During the elaboration of this document, I had contact with the practical implementation of certain features on a few multimedia frameworks allowing me to have a better idea of the complexity involved with developing an application that uses such middleware and the greater difficulty of building a working element such as a container parser or a video decoder.

Almost all of the proposed objectives have been accomplished and some useful extra information has been added, especially on the subject of multimedia frameworks, which ended up including the most innovative work. Unfortunately, the objectives of comparing the different analysing applications, for both video compression and containing formats, could not be satisfactorily achieved, since not many worthy solutions were possible to find, due to some seclusion to more personal publicity mediums, and the ones found had little characteristics in common. The solution for the spread of information through the company passes by using this document to determine the main point of focus and later follow on its references and the gained knowledge to venture towards more technical details.

Two important facts to retain are the, sometimes masked, differences between image video formats and native video formats, and between system layers and container layers.

The main characteristics that must be taken into account for consciously choosing a video compression format are the wanted level of coding and decoding and complexity, the availability (and sequentially the price range) of the format’s codecs and/or hardware, the involved bitrates and visual quality requirements.

For video containers it all comes down to the inverse relationship between the amount of codecs and the depth of metadata supported versus the level of software and hardware complexity.

When deciding a framework to develop into, the main arguments hinge on the targeted operating system(s) and the codecs/transforms needed. There are other software solutions that, while not having the same framework structure, can nonetheless provide most of the features available through these middlewares.

Although the initiation on C++ and DirectShow programming took longer than I expect, due to the rather complex already existent application and the incompatibility issues of some DirectShow filters with a number of graphic accelerating cards, the outcome of the programming exercise was very good. It fulfilled all the stipulated requirements, maintaining its integrity, even after a thorough battery of tests.

More detailed conclusions, relating to the more specific matters studied throughout the document, are present at the end of each section: 2.4 Concluding Thoughts on Video Compression, 3.3 Closing Remarks on Video Containers and 4.6 Final Thoughts on Multimedia Frameworks.

The biggest difficulties I faced during the elaboration of this project were mainly caused by its large amount of included subjects. This very broad scope did not allow me to go a bit more in-depth on certain subjects, as I would like to. Even though sacrificing some detail, the fact that my time was not enough to go deeper into the matters at hand, still made it possible to write about the most important aspects and to introduce the reader to the key elements regarding those topics. This allows someone interested in knowing more about a certain theme to focus his search for information. In addition, the referenced works contain additional information that permit an excellent starting point for further pursuit of more technical information.

6.3 Future Developments

The matters of video compression algorithms and formats are much wider than the information collected here. To make for a more complete base of information variants of the algorithms and other codecs could be included to the list of studied themes. Still for this document, all necessary information, for someone that needs not to linger on technical details, is available within. Much like compression formats, more containing formats would further complete the available knowledge, but for the professional market, only AAF would make a worthy extra.

To allow a more up-to-date analysis of the video compression formats available today the examination of a lossy video wavelet based compression format, would be valuable. This kind of format will, in due time, become the compression standard substituting the previous DCT based algorithms. It has not been done during this work because no proper implementations of these codecs are ready, being the current ones either too slow or incomplete.

If more applications for testing formats specifications were available, a proper comparison could have been made. By directly contacting companies devoted to the implementation of this kind of software, more trial versions will surely be made available.

My biggest regret was to not have had the time and resources to methodically benchmark the different multimedia frameworks for a more apposite comparison. A properly define array of tests could more objectively verify the performance these frameworks provide on the various operating systems, media processes types and distinct codecs. Yet this is not necessarily needed for a correct sampling of the multimedia platforms, because their properties, supported codecs, operating systems and operations make it possible to choose, with a very high certainty, which middleware better fits the reader's needs.

During the implementation of the DirectShow based WMV player, I had contact with the frameworks inner processes and structure. I would like to delve into the actual development of a coding or decoding filter to better understand the complexities behind it and to improve on my knowledge of compression formats.

Glossary

Application Programming Interface – A language that enables communication between computer programs, in particular between application programs and control programs.

Bit-plane – Part of a digital medium (such as image or sound) it is a set of bits having the same position in the respective binary order. For example, for 16-bit data representation there are sixteen bit-planes: the first bit-plane contains the set of the most significant bit and the 16th contains the least significant bit.

Blur – Image filter designed to render obscure by making the form or outline of an object confused and uncertain; to make indistinct and confused.

CIF – Common Intermediate Format, also known as FCIF (Full Common Intermediate Format), is a format used to standardize the horizontal and vertical resolutions in pixels of YCbCr sequences in video signals, commonly used in video teleconferencing systems. It was first proposed in the H.261 standard. It stands for the video resolution 352 × 288.

Codec – Combination of the words en**cod**er/**dec**oder. It is a device or software that enables compression and/or decompression of digital data, usually audio or video, based on a certain compression format.

Colour Space – A colour model that is an abstract mathematical model describing the way colours can be represented as tuples of numbers, typically as three or four values or colour components.

Compression Ratio – The quantification of the reduction in data-representation size produced by a data compression algorithm. Equals the uncompressed size divided by the compressed size and it is usually expressed as 5:1 (5 to 1).

Dead-zone Quantization – Where the quantum around the zero value is defined to be as twice as wide (i.e., a width of two from -1 to +1) as the rest of the other quanta, each of which has the width of one. This technique is effective in filtering out noise signals due to the high probability of noise signals occurring near the zero value, that is, in the [-1, +1] region.

Deblocking – Process applied to blocks in decoded video to improve visual quality and prediction performance by smoothing the sharp edges which can form between blocks when block coding techniques are used. The filter aims to improve the appearance of decoded pictures.

Decorrelation – A general term for any process that is used to reduce autocorrelation within a signal, or cross-correlation within a set of signals, while preserving other aspects of the signal. A frequently used method of decorrelation is the use of a matched linear filter to reduce the autocorrelation of a signal as far as possible. Since the minimum possible autocorrelation for a given signal energy is achieved by equalising the power spectrum of the signal to be similar to that of a white noise signal, this is often referred to as signal whitening

Denoising – The extraction of a signal from a mixture of signal and noise by removing the latter.

Framework – A set of common software routines that provides a foundation structure for developing an application.

Halftone – A reprographic technique that simulates continuous tone imagery through the use of equally spaced dots of varying size. It can also be used to refer specifically to the image that is produced by this process.

Macroblock – A term used in video compression, which represents a block of *x* by *y* pixels, usually a square. Each macroblock contains a number of Y blocks and fewer Cb blocks and Cr blocks relative to the YCbCr colour space used. Macroblocks can be subdivided further into smaller blocks, called partitions.

Middleware – Software layer, usually between an application and the OS.

Multiplexing – The term used to refer to a process where multiple analogue message signals or digital data streams are combined into one signal over a shared medium. The aim is to share an expensive resource.

Operating System – Software designed to control the hardware of a specific data-processing system in order to allow users and application programs to make use of it, usually by exposing the necessary services in the form of APIs.

PAL-SD – Short for Phase Alternating Line – Standard Definition, it is a colour encoding system and a video used in broadcast television systems in many parts of the world. Also called 576i it features 576 lines and 720 of horizontal resolution in digital mode and 625 by 704 in analogue.

Pixel – Short for **pic**ture **el**ement, (using the common abbreviation "pix" for "pictures") is a single point in a capture device or graphic image.

QCIF – Quarter CIF. To have one fourth of the area as "quarter" implies, height and width of the frame are halved. It stands for the video resolution 176 × 144.

QVGA – The Quarter Video Graphics Array (also known as Quarter VGA or qVGA) is a popular term for a computer display with 320 × 240 resolution. The name is derived from the fact that it offers 1/4 of the 640 × 480 maximum resolution of the original IBM VGA display technology.

[Hig02]

Bibliography

All online references were confirmed available as of July 5th 2008.

Digital Video Compression

- [Abr63] Norman Abramson. **Information Theory and Coding**. New York: McGraw-Hill, 1963. pp. 33-38.
- [Ber06] Marcel Berberich. **MPEG2 Profiles** (Online) July 20th, 2006. www.reelport.info/board/viewtopic.php?id=8
- [Bog04] Dumitru Bogdan**. An analyzer for MPEG 2 Program Stream files** (Online) April 12th, 2004. www.codeproject.com/KB/audio-video/program_stream_analyzer.aspx
- [Com04] Computer Desktop Encyclopaedia. **Chroma subsampling** (Online) 2004. www.computerlanguage.com/index.htm
- [Ele08] Elecard. **StreamEye Tools** (Online) 2008. www.elecard.com/products/products-pc/professional/streameye-tools
- [Fog96] Chad Fogg. **MPEG-2 FAQ** (Online) April 2nd, 1996. bmrc.berkeley.edu/research/mpeg/faq/mpeg2-v38/faq_v38.html
- [Fun99] Thomas Funkhouser. **Image Sampling and Reconstruction** (Online) September 24th, 1999. www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/sampling/index.htm
- [Gra06] Grass Valley. **AVC Analyzer** (Online) December 6th, 2006. catalogs.infocommiq.com/AVCat/images/documents/pdfs/CDT-2008D.pdf
- [Hab71] Ali Habibi. **Comparison of nth-order DPCM encoder with linear transformations and block quantization techniques**. IEEE Transactions on Communications, December 1971, pp. 948-956.
- [Hen97] Chua Heng. **Image Compression: JPEG** (Online) May 19th, 1997. pascalzone.amirmelamed.co.il/Graphics/JPEG/JPEG.htm
- [Hig02] HighDef. **The HighDef Glossary of Terms** (Online) 2002. www.highdef.com/library/glossary.htm
- [Hun03] Jill Hunter. **Digital cinema reels from motion JPEG2000 advances** (Online) June 1st, 2003. www.eetimes.com/in_focus/mixed_signals/OEG20030106S0034
- [Ill08] Illustrate. **Spoon's Audio Guide** (Online) 2008. www.dbpoweramp.com/spoons-audio-guide-starting.htm
- [Ima08] ImageMagick Studio LLC. **Introduction to Motion Picture Formats** (Online) 2008. www.imagemagick.org/script/motion-picture.php
- [Kno00] Adolf Knoll. **Efficiency of wavelet conversion** (Online) August 31st, 2000. digit.nkp.cz/knihcin/digit/vav/wavelet/Efficiency-of-wavelet-conversion.html
- [Koe02] Rob Koenen. **Overview of the MPEG-4 Standard** (Online) March 2002. www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm
- [Lib08] Library of Congress. **Format Descriptions for Moving Images** (Online) April 2nd, 2008. www.digitalpreservation.gov/formats/fdd/video_fdd.shtml
- [Lo] Victor Lo. **MPEG-2** (Online) www.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/mk/mpeg2beg/beginnzi.htm

- [LW07] Jay Loomis and Mike Wasson. **VC-1 Technical Overview** (Online) October 2007. www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx
- [Mar01] Dave Marshall. **Video and Audio Compression** (Online) October 4th, 2001. www.cs.cf.ac.uk/Dave/Multimedia/node200.html
- [Min04] Mindego. **MPEG Analyzer** (Online) July 10th, 2004. www.mindego.com/products/analyzer.php
- [Mir08] MiraVid. **MSight Overview (H264, VC-1, MPEG 1/2/4), and Transport Stream Analyzer** (Online) April 21st, 2008. www.miravid.com/product_MSight.htm
- [MV03] Detlev Marpea and George Valeri. **Performance evaluation of Motion-JPEG2000 in comparison with H.264/AVC operated in pure intra coding mode** (Online) October 17th, 2003. www.f4.fhtw-berlin.de/~barthel/paper/spie03_marpe_et_al.pdf
- [Pen07] Fernando Pereira and Paulo Nunes. **Levels for MPEG-4 Visual Profiles** (Online) 2007. www.m4if.org/resources/profiles/index.php
- [SE01] Andrea Silva and Juliana Eyng. **Wavelets e Wavelet Packets** (Online) June 23rd, 2001. www.inf.ufsc.br/~visao/2000/Wavelets/index.html
- [Shi02] Yoshiaki Shishikui. **High-efficiency coding of video signals** (Online) November 12th, 2002. www.nhk.or.jp/strl/publica/bt/en/le0009-1.html
- [Sim08] **Simple Directmedia Layer** (Online) www.libsdl.org
- [Sun08] Sunray Image. **YUV Player, YUV Converter, YUV Analyzer and YUV Editor** (Online) June 9th, 2008. www.sunrayimage.com/index.html
- [Sym04] Peter Symes. **Digital Video Compression**. United States of America: McGraw-Hill, 2004.
- [TaS07] Stewart Taylor. **Compression for High-Quality, High Bandwidth Video** (Online) October 23rd, 2007. softwarecommunity.intel.com/articles/eng/1619.htm
- [Tho08] Thomson. **Real-time MPEG-2 Transport Stream Analyzer** (Online) 2008. www.thomsongrassvalley.com/products/tbm/mercury
- [Wil05] Adam Wilt**. DV Technical Details** (Online) August 28th, 2005. www.adamwilt.com/DV-tech.html

Digital Video Containers

- [AAA07] Laurent Aimar, Roberto de Amorim and Daisuke Anayama. **Matroska** (Online) 2007. www.matroska.org
- [All08] All-Streaming-Media.Com. **Windows Media: Advanced Systems Format (.ASF files)** (Online) 2008. all-streaming-media.com/faq/streaming-media/Format-Advanced-Systems-Format-ASF-files.htm
- [Bue08] Tom Buehler. **QuickTime** (Online) May 25th, 2008. people.csail.mit.edu/tbuehler/video/codecs/quicktime.html
- [Dev02] Bruce Devlin. **MXF - the Material eXchange Format** (Online) July 12th, 2002. www.ebu.ch/en/technical/trev/trev_291-devlin.pdf
- [DW06] Bruce Devlin and Jim Wilkinson. **The MXF Book**. Burlington, United States of America: Focal Press, 2006.
- [Hoo97] Simon Hooper. **Quicktime Developer's Tools** (Online) May 12th, 1997. www.prenhall.com/divisions/ESM/app/hooper/html/tools.html
- [Ins07] Institut für Rundfunktechnik. **MXF Test Centre** (Online) November 28th, 2007. ftp.irt.de/IRT/mxf/tools/analyzer/comparison.html

- [MeC07] MediaCross. **PRISMA-PRO MXF - Analyzer** (Online) November 30th, 2007. www.mediacross.tv/eng/Prisma-Pro.htm
- [Mic08] Microsoft. **Windows Media ASF Viewer 9 Series** (Online) 2008. www.microsoft.com/windows/windowsmedia/forpros/format/asfviewer.aspx
- [San06] Ernesto Santos. **AAF, MXF, XML…Putting it All Together**. Portugal, 2006.
- [San07a] Ernesto Santos. **MXF – SMPTE Material eXchange Format**. Portugal, January 17th, 2007.
- [San07b] Ernesto Santos. **Operational Patterns… the MXF Flavours?** Portugal, 2007.
- [SMP04a] SMPTE. **SMPTE 377M: The MXF File Format Specification (The Overall Master Document).** 2004.
- [SMP04b] SMPTE. **SMPTE EG41: MXF Engineering Guide (How to Use MXF).** 2004.
- [SMP04c] SMPTE. **SMPTE EG42: MXF Descriptive Metadata (How to Use Descriptive Metadata in MXF).** 2004.
- [TaC03] Chris Taylor. **Introduction to metadata** (Online) July 29th, 2003. www.library.uq.edu.au/iad/ctmeta4.html

Multimedia Frameworks

- [Ado08] Adobe Systems Incorporated. **Flash Player Penetration** (Online) March 2008. www.adobe.com/products/player_census/flashplayer/
- [Apl07] Apple, Inc. **QuickTime Architecture** (Online) 2007. developer.apple.com/documentation/QuickTime/index.html
- [CoW08] CompWisdom. **Topic: Gstreamer** (Online) 2008. www.compwisdom.com/topics/GStreamer
- [FMJ07] FMJ Development Team. **FMJ Project** (Online) October 2007. fmj-sf.net
- [Gno08] Gnome. **What is Totem?** (Online) June 19th, 2008. www.gnome.org/projects/totem
- [Her07] Carlos Herrero. **Different Operating Systems and their Multimedia Support** (Online) September 27th, 2007. www.tml.tkk.fi/Opinnot/T-111.5350/2007/MM_OS.pdf
- [MFD08] **Media Foundation Development** (Online) 2008. www.bokebb.com/dev/english/2028/thread1.shtml
- [Mot08] Motama. **Network-Integrated Multimedia Middleware** (Online) 2008 www.motama.com/nmm.html
- [MSD08a] MSDN**. DirectShow** (Online) 2008. msdn.microsoft.com/en-us/library/ms783323(VS.85).aspx
- [MSD08b] MSDN. **DirectShow Development** (Online) 2008. forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=129&SiteID=1
- [MSD08c] MSDN. **Microsoft Media Foundation SDK** (Online) 2008. msdn.microsoft.com/en-us/library/ms694197(VS.85).aspx
- [Pio08] Pioneers of the Inevitable. **Songbird** (Online) 2008. getsongbird.com
- [Rea08a] RealNetworks. **Helix DNA Architecture** (Online) February 25th 2008. helix-client.helixcommunity.org/Architecture.html
- [Rea08b] RealNetworks. **Helix Client and Server Software Developer's Guide** (Online) June 6th 2008. common.helixcommunity.org/2003/HCS_SDK_r5/helixsdk.htm
- [Ref08] Refsnes Data. **OS Platform Statistics** (Online) May 2008. www.w3schools.com/browsers/browsers_os.asp

- [Sti08] Thomas Stichele. **Gstreamer Documentation** (Online) June 18th, 2008. gstreamer.freedesktop.org/documentation
- [Str03] Daniel Strigl. **DirectShow MediaPlayer in C#** (Online) December 16th, 2003. www.codeproject.com/KB/directx/directshowmediaplayer.aspx
- [TB08] Wym Taymans and Steve Baker. **GStreamer Application Development Manual (0.10.17.1)** (Online) May 25th 2008. gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/index.html
- [TCP08] The Code Project. **Audio & Video - DirectShow** (Online) 2008. www.codeproject.com/KB/audio-video/index.aspx?#Audio&Video–DirectShow
- [ThC00] Chris Thompson. **DirectShow for Media Playback in Windows** (Online) July 28th, 2000. www.flipcode.com/archives/DirectShow_For_Media_Playback_In_Windows-Part_I_Basics.shtml
- [Tim07] Christian Timmerer. **The MPEG-21 Multimedia Framework** (Online) June 23rd, 2007. multimediacommunication.blogspot.com/2007/06/mpeg-21-multimedia-framework.html
- [Ura08] Uraeus. **Nokia releases GTK+ and GStreamer based Internet tablet** (Online) May 25th, 2008. gnomedesktop.org/node/2265
- [Vuo06] Petri Vuorimaa. **Different Operating Systems and their Multimedia Support** (Online) September 9th, 2006. www.tml.tkk.fi/Opinnot/T-111.5350/2006/MM_OS.pdf

Implementation

- [MOG08] MOG-Solutions. **theScribe LITE** (Online) 2008. www.mog-solutions.com/produtos.php?ID=105

Annexes

A. Analysis Software

1. Video Compression Formats

Even though a video format specification bounds all aspects of decoding a certain compression format it does not in any way limit the encoder, so developers are free to introduce all kinds of improvements, provided that the end stream decodes according to the format layout. This leads to very different encoders, which may or may not follow the specification thoroughly. When this happens, small errors can be introduced by mistake or omission in the files, being practically impossible to discover them without the proper tools. This is especially harmful in a professional environment when using video editing tools, transcoding to another format or broadcasting. Some applications will not work properly if the inputted file does not abide by the specification.

When an error is detected in a media handling application and debugging is needed or simply when a quality screening process is required, the problems can be detected rapidly and accurately by the analysing software. That said, StreamEye tools possesses the largest number of features, MSight the most compression formats supported and YUVTools is recommended for uncompressed files analysis.

A comparison between all of them cannot be appropriately made because even though some features are shared others depend heavily on for what purposes the application is needed.

Some of these tools could not be tested because no sample application was supplied; also, there are other software solutions not taken into account, that offer the same or better capabilities, but that are not disclosed to the general public, being secluded to the professional industry, particularly in cinema and television conventions and fairs.

1.1 StreamEye Tools

This software pack from Elecard, includes a few utilities, each specialising in different fields of video quality control and file format compliancy. The main **StreamEye** application provides the user with a visual representation of the encoded video features and a stream structure analysis of MPEG-1/2/4 or AVC/H.264 Video Elementary Streams (VES), MPEG-1 System Streams (SS), MPEG-2 Program Streams (PS) and MPEG-2 Transport Streams (TS).

StreamEye Tools feature:

- Navigation and display of media stream picture-by-picture (I, P, B).
- Display of the time, type, size and number of a frame in a stream, and its properties: size, type, PTS, decoding order and offset from the file beginning.
- Display the bit rate (declared in the sequence header) and a calculated bit rate.
- Display a diagram representative average bit rate (moving average).
- Display detailed information about macroblocks in MPEG-1/2/4 and H.264 video streams, as well as bit rate lines and of the current frame.
- Frame-accurate positioning.
- Display of the stream and gathering of statistics relating to the entire file.

The Elecard **Stream Analyzer** is designed for syntax analysis of encoded media streams and presentation of the analysis log in a human readable form. It operates with VC-1, MPEG-1 Video/Audio, MPEG-2 Video/Audio, MPEG-4 Video, AAC, AC-3 and AVC/H.264 files.

Features:

- Selection of packets in a text by PID and StreamID
- Stream viewing in the HEX mode
- Storing the information about the stream and currently selected packets into a .TXT
- Search of elements by offset, PID and text
- Error report generation (Transport Packet Counters and Start Indicators)
- Calculation of the overhead in transport and program streams

Elecard **Buffer Analizer** allows analyzing of the decoder video buffer parameters. It supports MPEG-1 System Stream, MPEG-2 Program and Transport Stream, MP4 Stream, VC-1 Video and AVC/H.264 video stream.

Features:

- Displaying (and saving to .TXT file) of the following frame information:
    - Frame size and the frame number in the stream order
    - The time of frame removal from the buffer
    - Padding size and frame type (for MPEG-2 and AVC streams)
    - The time of frame arriving into the buffer (AVC)
- Ability to select a stream for analysis, if the file contains several streams
- Ability to save the analysis report to .TXT file which contains the following:
    - Buffer size, bitrate value and frame rate
    - Detected buffer overflow and underflow errors
    - Total and average padding (for MPEG-2 and AVC streams)
    - Bitrate type (CBR, VBR) and specific format information
    - Plotting of the buffer fullness curve and ability to save chart as a bitmap file
- Ability to interrupt and resume stream parsing.

[Ele08]

1.2 AVC Analyzer

An audio/video codec analyzer with off-line analysis capabilities oriented for VC-1 and H.264 video analysis. Idealised for designers, system integrators, and network operators, our AVC (audio/video codec) analyzer performs in-depth syntax analysis and decoding for H.264 and VC-1 files. It can be used for a host of applications, including standards compliance and interoperability testing, new codec development, media provisioning, quality assessment of H.264 and VC-1 encoded video, and transmission troubleshooting.

Features:

- In-depth H.264 and VC-1 syntax analysis.
- Multiple A/V formats conformance testing.
- Dynamic buffer analysis.
- Video information overlay
- A/V synchronization.
- Live capture of IP traffic.
- Bit-stream navigation and editing down to MB level.
- AAC, Dolby AC-3, MPEG-1 and MPEG-2 audio analysis.
- Extraction and analysis of media streams from IP capture files

[Gra06]

1.3 MSight

MSight provides users with the ability to "see it all" with instant access to the lowest level syntax elements present in the supported compression standards. MSight is prepared for dealing with HD content and the volumes of digital media being produced. Video and audio can be processed very quickly and problems can be identified, debugged, and fixed in the shortest possible period. It supports VC-1, H.264/AVC, MPEG-4 part 2, MPEG-1/2.

This product can guarantee quality levels and debugging of products that are closer to release, when errors become sparse and require larger streams to detect. This also makes it useful for checking long-term bitrate characteristics of encoding algorithms for refinement.

Highlights:

- Support for all major media standards and future extensibility as new support is added
- System level analysis (transport and program including graphical buffer analysis)
- Real-time playback with instant forward and backward navigation and seeking slider
- HD analysis, conformance and performance testing (including VOD CEP 2.0)
- Closed Caption and XDS data decode and display over video in real-time
- Scripting support for automated regression testing
- Video quality comparison to YUV baseline
- Extraction of video or audio elementary stream from container file formats
- Plug-in interface; custom IDCT algorithms for bit accurate comparison with encoder
- Faster than real-time compliance testing. Greater number of video streams can be effectively processed for regression testing resulting in improved QA effectiveness
- Instant fast seeking to the exact location of any error identified in a compliance test to allow full inspection of the stream data there
- Access to relevant statistics and analytics required to tune compression algorithms
- Compare different compression algorithms or codecs to evaluate coding efficiency
- Data overlaid on top of video for quick analysis

[Mir08]

1.4 Mindego Analyzer

Mindego develops software tools to help technical professionals work more productively and knowledgeably with international media standards. The Mindego Analyzer is a commercial software application designed with these goals in mind, providing discovery, analysis and measurement capabilities for stored MPEG-4 media data. The Mindego Analyzer is a MPEG-4 and AVC analyzer (H.264 analyzer) in one package.

The Mindego Analyzer gives media professionals the ability to analyze and explore data files coded to the MPEG-4 (ISO/IEC 14996) specification and related industry interoperability guidelines, such as ISMA and 3GPP, that incorporate elements of the MPEG-4 standard. The software operates on the Windows platform. Its purpose is to provide fast, yet comprehensive insight into the structure, contents and coding of MPEG-4 files and bitstreams.

The MA-4600 is a full-featured media file and bitstream analyzer, designed by and for MPEG-4 technical professionals. In addition to MP4 file format, MPEG-4 video, and AVC bitstream browsing capabilities it offers syntactic and semantic tests for conformance, buffer analysis, and in-depth analysis at the macroblock level.

[Min04]

1.5 YUVTools

This analysing software package lets the user play many different YUV/RGB formats, with full playing controls, as well as two layers of grids overlaid to indicate macro block (MB) and block boundaries in MPEG 1/2/4 and H.264. The user can also convert one YUV/RGB format to another, compare two YUV files and display their difference, and calculate the PSNR. YUVTools includes YUVPlayer, YUVConverter, YUVAnalyzer and YUVEditor.

The main features of YUVTools are:

Supports FOURCC: when one is selected, the equivalent formats are set to match it:

- UYVY, UYNV or Y422 (YUV422, U-Y-V-Y, progressive, packed)
- IUYV (YUV422, U-Y-V-Y, interlaced, packed)
- YUYV, YUNV, V422 or YUY2 (YUV422, Y-U-Y-V, progressive, packed)
- YVYU (YUV422, Y-V-Y-U, progressive, packed)
- IYU2 (YUV444, U-Y-V, progressive, packed)
- YV16 (YUV422, Y-U-V, progressive, planar)
- YV12 (YUV420, Y-V-U, progressive, planar)
- IYUV or I420 (YUV420, Y-U-V, progressive, planar)

It can accept (play, convert and analyse) the combination of following 4 formats with predefined or arbitrary resolution:

- in 4:4:4, 4:2:2 or 4:2:0 YUV sample format, or in 4:4:4 RGB format;
- in different component order, like YUV, YVU, UYV, RGB, BGR, etc.
- in progressive (one single field) or interlaced (two fields) format;
- in planar (YYY...UUU...VVV...) or packed (YUV, YUV....) pixel format;

Since the raw video data is headless and with no format information saved in the file, a format guess function is provided when the video data is loaded to play, convert and analyse. For each format combination, an image preview (first frame), a format diagram (illustrating the data structure) and a raw data dump map will be displayed interactively, to help the users to try the combination of above format, until the right format is obtained.

YUV Converter:

- It can be used to convert any combination of above format to another;
- It can convert a sequence of BMP files into a single YUV/RGB format, or vice versa;
- It can scale up/down if the input and output have different resolution
- Two YUV files can be joined into one file.
- One area of all frames can be cropped and saved into a new YUV file.
- Vertically flipping all frames in original file and saved into a new YUV file.

YUV Editor:

- Ability to select one block, and zoom in to check each pixel values, in both RGB and YUV, and change the pixel values, to create especial colours or patterns in block
- Overlay two layers of grid, which can be used to show MB (Macro Block, 16x16 in MPEG2) and Block (8x8 in MPEG2), in two different colours;
- Command line options for playing and converting to batch processes or scripting.

YUV Player:

- When playing a video, it can reverse play and frame by frame forward/backward;
- When playing a video, it can display all YUV components, or each one separately;
- User can open and play multiple video files at same time;

YUV Analyzer:

- A PSNR tools can be used to calculate the PSNR of two input video file;
- Two videos can be compared frame by frame and the difference (residual) displayed
- The brightness of the difference image can be adjusted, in case the difference is very small; the comparison can be applied to all YUV components, together or separately;
- Data Overlay: Other information can be overlaid on top of video display while it is playing. For each macroblock, one dimensional (like block type) or two dimensional information (like motion vector) can be overlaid;

[Sun08]

2. Video Container Formats

As referred in the previous chapter, these applications provide the ability to swiftly trace errors in a file format according to its official definition. This is progressively more important due to the ever-increasing complexity of the containing formats, notably with the inclusion of more complex and detailed metadata as well as new video functionalities.

From the following software applications, two were not tested: a trial version of Prisma-Pro MXF Analyzer was not available for download and Movie Analyzer only runs on Macintosh operating systems, which I do not have at my disposal. The only comparison possible from the lot of discovered programs, would be between the two MXF analysers but since only one of them is available that assessment could not be made.

2.1 Windows Media ASF Viewer 9 Series

This free of charge application is a tool for inspecting the contents of files (such as .asf. .wma, and .wmv file) to make sure they meet the requirements of the Advanced Systems Format (ASF) specification. This tool is helpful for independent software vendors, content providers, and device manufacturers who want to make sure that their files have the proper structure and that they will playback correctly.

ASF Viewer can be run in batch mode or from the user interface and displays several different kinds of information regarding the file’s structure and content, with the possibility to save that information reports into HTML files.

[Mic08]

2.2 PRISMA-PRO MXF - Analyzer

Prisma-Pro is a highly reliable software tool designed to analyse MXF-Files. Further analysis includes, bug identification, conformance checks and validation to the data model as defined by the MXF-Standard. A broad choice of parameters allows the analysis of a wide range of existing formats. The file under test can be additionally analysed by user-defined criteria.

Prisma-Pro is suitable to analyse MXF-Files from AVID, Grass Valley, Panasonic, Sony and others in SD and HD including video essence, audio essence, Metadata, Operational patterns etc. Prisma-Pro features a quick analysis and an intelligent traffic light system to display the corresponding messages

Prisma-Pro Deep-Analysis provides referencing of tested files to the existing SMPTE-Standard. Allocation of errors and other messages are precisely defined by the Byte-Offset. All results are automatically saved in a log file. Files are available in XML, Word and Excel.

[MeC07]

2.3 IRT - MXF Analyser

The MXF Analyser is based on the MXF::SDK developed by IRT and MOG Solutions. The full version of the analyser is being implemented to support in-depth analysis of the:

- KLV layer
- Partition multiplex
- Metadata (decoding and analysis)
- Index Tables
- Essence Containers and their payload

The total structure of the MXF file (including the contents of the header metadata for each partition) is exported as an instance of the XML Schema and can be further validated using XML tools.

IRT presents the MXF Analyser Professional as a tool for thorough analysis and validation of MXF-files, and for easy integration into IT-based systems. It is well suited to support operational validation of MXF-files in production facilities. Due to the controllable depth of analysis and its flexibility, the MXF Analyser can also be used to develop MXF-file analysis applications for engineering or laboratory use. The MXF Analyser Professional is shipped as a DLL. The shipped package contains a simple GUI program (similar to the light version's GUI) as well as a command line demo application, which both constitute readily usable MXF analysis tools.

For self-test a light version with limited features is available for free download. The executable has been built on Windows XP using Visual Studio.NET 2002. The light version of the MXF Analyser does not validate the relationship of Metadata attributes with respect to the Essence of the file. Nor does it validate Essence Containers, Index Tables and the format of Essence streams.

[Ins07]

2.4 Movie Analyzer

Movie Analyzer provides information about movie tracks. It displays duration, creation and modification dates, matrix information, poster time, file resource size, video track information such as number of samples, codecs used, frame rate, data size, audio track information such as samples, volume, channels, compression used, track information, data layout in the file, and edits in the movie.

This software also displays a graphical representation of the 'playability' of the movie showing data throughput, the interleave factors and other useful information.

[Hoo97]

B. Simple Player Source Code

Different implementations of a simple player on the studied main frameworks are provided in the following pages. Code may not be complete since it has not been thoroughly tested; these are examples available on online documentation and SDKs of each framework.

1. DirectShow

Playing a file using DirectShow in C++ is illustrated in the SDK documentation as:

```
#include <dshow.h>

void main ( void )
{
     IGraphBuilder *pGraph = NULL;
     ImediaControl *pControl = NULL;
     ImediaEvent *pEvent = NULL;

     // Initialize the COM library.
     HRESULT hr = CoInitialize ( NULL );
     if ( FAILED (hr) )
     {
          printf ("ERROR - Could not initialize COM library");
          return;
     }

     // Create the filter graph manager and query for interfaces.
     hr = CoCreateInstance (CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
                           IID_IGraphBuilder, (void **)&pGraph);
     if ( FAILED (hr) )
     {
          printf ("ERROR - Could not create the Filter Graph Manager.");
          return;
     }

     hr = pGraph -> QueryInterface (IID_IMediaControl, (void **)&pControl);
     hr = pGraph -> QueryInterface (IID_IMediaEvent, (void **)&pEvent);

     // Build the graph
     hr = pGraph -> RenderFile (L"C:\\Example.avi", NULL);
     if ( SUCCEEDED (hr) )
     {
          // Run the graph.
          hr = pControl -> Run ();
          if ( SUCCEEDED(hr) )
          {
               // Wait for completion.
               long evCode;
               pEvent -> WaitForCompletion (INFINITE, &evCode);
          }
     }

     pControl -> Release ();
     pEvent -> Release ();
     pGraph -> Release ();
     CoUninitialize ();
}
```

The equivalent code in C# using *interop* and *quartz.dll*:

```
using System;
using QuartzTypeLib;

class PlayFile
{
     public static void Main( string [] args )
     {
          FilgraphManagerClass graphClass = null;
          try
          {
               graphClass = new FilgraphManagerClass ();
               graphClass.RenderFile ( args[0] );
               graphClass.Run ();
               int evCode;
               graphClass.WaitForCompletion ( -1, out evCode );
          }
          catch ( Exception ) {}
          finally
          {
               graphClass = null;
          }
     }
}
```

2. Gstreamer

Simple player code snippet in C++ with Gnome Objects:

```
#include <gst/gst.h>

gint main ( gint argc, gchar *argv [] )
{
     GMainLoop *loop;
     GstElement *play;
     GstBus *bus;

     /* init GStreamer */
     gst_init (&argc, &argv);
     loop = g_main_loop_new (NULL, FALSE);

     /* make sure we have a URI */
     if (argc != 2)
     {
          g_print ("Usage: %s <URI>\n", argv[0]);
          return -1;
     }

     /* set up */
     play = gst_element_factory_make ("playbin", "play");
     g_object_set (G_OBJECT (play), "uri", argv[1], NULL);
     bus = gst_pipeline_get_bus (GST_PIPELINE (play));
     gst_bus_add_watch (bus, my_bus_callback, loop);
     gst_object_unref (bus);
     gst_element_set_state (play, GST_STATE_PLAYING);

     /* now run */
     g_main_loop_run (loop);
     /* also clean up */
     gst_element_set_state (play, GST_STATE_NULL);
     gst_object_unref (GST_OBJECT (play));

     return 0;
}
```

3. Media Foundation

A simple URL player:

```
#include <stdio.h>
#include <conio.h>
#include <assert.h>
#include <mfidl.h>

int main (int argc, const char* argv[])
{
     HRESULT hr = S_OK;
     IMFAttributes* pConfiguration = NULL;
     IMFMediaSession* ppMS = NULL;
     IMFSourceResolver* pSourceResolver = NULL;
     IUnknown* pSource = NULL;
     HWND hVideoWnd;                 // Window for video playback.

     // Start up Media Foundation platform.
     CHECK_HR (hr = MFStartup (MF_VERSION));

     // Create the media session.
     CHECK_HR (hr = MFCreateMediaSession (pConfiguration, ppMS);

     // Create a media source.
     hr = MFCreateSourceResolver (&pSourceResolver);
     if (SUCCEEDED (hr))
     {
          MF_OBJECT_TYPE ObjectType = MF_OBJECT_INVALID;
          hr = pSourceResolver -> CreateObjectFromURL (
               argv [1],                   // URL of the source.
               MF_RESOLUTION_MEDIASOURCE, // Create a source object.
               NULL,                       // Optional property store.
               &ObjectType,                // Receives the object type.
               &pSource                   // Receives pointer to the source.
          );
     }

     // Create partial a topology connecting the media source to renderers.
     IMFTopology *pTopology = NULL;
     IMFPresentationDescriptor *pPD = NULL;
     DWORD cStreams = 0;  // Number of streams in the source.

     // Create a new topology.
     CHECK_HR (hr = MFCreateTopology(&pTopology));

     // Create the presentation descriptor for the media source.
     CHECK_HR (hr = pSource -> CreatePresentationDescriptor (&pPD));

     // Get the number of streams in the media source.
     CHECK_HR (hr = pPD -> GetStreamDescriptorCount (&cStreams));

     // For each stream, create the topology nodes and add them to the
     topology.
     for (DWORD iStream = 0; iStream < cStreams; iStream++)
     {
          CHECK_HR (hr = AddBranchToPartialTopology(pTopology, pSource, pPD,
          iStream, hVideoWnd));
     }
     (*ppTopology) -> AddRef ();


     // Queue the topology on the media session.
     ppMS -> SetTopology ();
     // Play the media for 10 seconds
     ppMS -> Play ();
     sleep (10);

     // Close media session
     ppMS -> Close ();

     // Shutdown media source
     pSource -> Shutdown ();

     // Shutdown media session
     ppMS -> Shutdown ();

     // Shutdown Media Foundation
     MFShutdown ();
}
```

4. QuickTime

Source code on MacOS:

```
#define doTheRightThing 5000

void PlayMyMovie (movie, aMovie)
{
     WindowPtr      aWindow;
     Rect           windowRect;
     Rect           movieBox;
     Movie          aMovie;
     Boolean        done = false;
     OSErr          err;
     EventRecord    theEvent;
     WindowPtr      whichWindow;
     short          part;

     err = EnterMovies ();
     if (err) return;

     SetRect (&windowRect, 100, 100, 200, 200);
     aWindow = NewCWindow (nil, &windowRect, "\pMovie", false,
noGrowDocProc, (WindowPtr)-1, true, 0);

     GetMovieBox (aMovie, &movieBox);
     OffsetRect (&movieBox, -movieBox.left, -movieBox.top);
     SetMovieBox (aMovie, &movieBox);

     SizeWindow (aWindow, movieBox.right, movieBox.bottom, true);
     ShowWindow (aWindow);
     SetMovieGWorld (aMovie, (CGrafPtr)aWindow, nil);

     StartMovie (aMovie);

     while ( !IsMovieDone(aMovie)  )
     {
          // Handle resize and update events...
          // Handle drag events
          MoviesTask (aMovie, DoTheRightThing);
     }

     DisposeMovie (aMovie);
     DisposeWindow (aWindow);
}
```

Same application on Windows OS:

```
// Resource identifiers
#define         IDM_OPEN 101

// Global variables
char            movieFile[255];       // Name of movie file
Movie           theMovie;             // Movie object
MovieController theMC;                // Movie controller

int CALLBACK WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine, int nCmdShow)
{
     InitializeQTML(0);               // Initialize QTML
     EnterMovies();                   // Initialize QuickTime

     // Main message loop
     ExitMovies();                    // Terminate QuickTime
     TerminateQTML();                 // Terminate QTML

} /* end WinMain */

LRESULT CALLBACK WndProc (HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     MSG          winMsg;
     EventRecord  qtmlEvent;
     int          wmEvent, wmId;
     // Fill in contents of MSG structure

     // Convert message to a QTML event
     NativeEventToMacEvent (&winMsg, &qtmlEvent);

     // Pass event to movie controller
     MCIsPlayerEvent (theMC, (const EventRecord *) &qtmlEvent);

     switch ( message )
     {
          case WM_CREATE:
               // Register window with QTML
               CreatePortAssociation (hWnd, NULL);
               break;

          case WM_COMMAND:
               wmEvent = HIWORD(wParam);  // Parse menu selection
               wmId = LOWORD(wParam);

               switch ( wmId )
               {
                    case IDM_OPEN:
                         // Close previous movie, if any
                         CloseMovie ();

                         // Get file name from user
                         if ( GetFile (movieFile) )
                         {
                              // Open the movie
                              OpenMovie (hWnd, movieFile);
                         }
                         break;
                    default:
                       return DefWindowProc (hWnd, message, wParam, lParam);
               } /* end switch ( wmId ) */
```

```
              break;
         case WM_CLOSE:
              // Unregister window with QTML
              DestroyPortAssociation (hWnd);
              break;
         default:
              return DefWindowProc (hWnd, message, wParam, lParam);

     } /* end switch ( message ) */

     return 0;

} /* end WndProc */

BOOL GetFile (char *movieFile)
{
     OPENFILENAME ofn;

     // Fill in contents of OPENFILENAME structure
     if ( GetOpenFileName(&ofn) )      // Let user select file
          return TRUE;
     else
          return FALSE;

} /* end GetFile */

void OpenMovie (HWND hwnd, char fileName[255])
{
     short theFile = 0;
     FSSpec sfFile;
     char fullPath[255];

     // Set graphics port
     SetGWorld ( (CgrafPtr)GetNativeWindowPort (hwnd), nil);

     strcpy (fullPath, fileName);                // Copy full pathname
     c2pstr (fullPath);                          // Convert to Pascal string

     // Make file-system specification record
     FSMakeFSSpec (0, 0L, fullPath, &sfFile);

     // Open movie file
     OpenMovieFile (&sfFile, &theFile, fsRdPerm);
     // Get movie from file
     NewMovieFromFile (&theMovie, theFile, nil, nil, newMovieActive, nil);

     CloseMovieFile (theFile);                       // Close movie file

     theMC = NewMovieController (theMovie, ... );  // Make movie controller

} /* end OpenMovie */

void CloseMovie (void)
{
     if ( theMC )                          // Destroy movie controller, if any
          DisposeMovieController (theMC);

     if ( theMovie )                       // Destroy movie object, if any
          DisposeMovie (theMovie);

} /* end CloseMovie */
```

5. Java Multimedia Framework

Here is the concise source code of a basic player:

```
import javax.media.*;
import javax.swing.*;
import java.awt.*;
import java.net.*;
import java.awt.event.*;
import javax.swing.event.*;

public class JMFTest extends JFrame
{
     Player _player;
     JMFTest ()
     {
          addWindowListener ( new WindowAdapter ()
                              {
                                 public void windowClosing ( WindowEvent e )
                                 {
                                      _player.stop ();
                                      _player.deallocate ();
                                      _player.close ();
                                      System.exit ( 0 );
                                 }
                              }
          );

          setExtent ( 0, 0, 320, 260 );
          JPanel panel = (JPanel)getContentPane ();
          panel.setLayout ( new BorderLayout () );
          String mediaFile = "vfw://1";
          try
          {
               MediaLocator mlr = new MediaLocator ( mediaFile );
               _player = Manager.createRealizedPlayer ( mlr );
               if (_player.getVisualComponent () != null)
                    panel.add ("Center", _player.getVisualComponent ());
               if (_player.getControlPanelComponent () != null)
                    panel.add ("South", _player.getControlPanelComponent
());
          }
          catch (Exception e)
          {
               System.err.println ( "Got exception " + e );
          }
    }

    public static void main (String[] args)
    {
         JMFTest jmfTest = new JMFTest ();
         jmfTest.show ();
    }
}
```