Dublin City University

School of Electronic Engineering

Module: `EE554`

Image and Video Compression

Module Notes

Academic Year 2012-2013: Semester 1

Module Co-ordinator:	Prof. Noel E. O'Connor
Lecturers:	Dr. Rafal Kapela, Dr. Kevin McGuinness
Emails:	`Noel.OConnor@dcu.ie`
	`Rafal.Kapela@dcu.ie`
	`Kevin.McGuinness@dcu.ie`

Contents

List of Figures

Acknowledgements

The authors would like to acknowledge and thank Dr. David Sadlier and Dr. Seán Marlow for their contributions to these notes and to the module in general. The authors would also like to thank Dr. Barry McMullin for providing a flexible and powerful framework for the preparation of these notes. Last, but by no means least, the authors would also like to thank Mr. Tomasz Adamek and Mr. Vivien Chappelier for their excellent work on the interactive teaching aids which complement these notes[1]

[1]The development of these teaching aids was funded by the Small Project Scheme of the Office of the Dean of Teaching and Learning.

Chapter 1

Module Overview

These are the lecture notes for module EE554: *Image and Video Compression* for the 2012-2013 academic year. These notes are also available online at:

`http://www.eeng.dcu.ie/ẽe554/notes/`

Please note that these notes may be incrementally revised and extended as the semester progresses.

In this introductory chapter, a general overview of the module is presented in terms of delivery, assessment, etc. A more formal overview, including an indicative syllabus, can be found in the official *Module Specification*.

1.1 Module Co-ordinators

The co-ordinators for this module for the 2012-2013 session are:

Prof. Noel O'Connor	Room L240A	Pho. ext.: 5078	Email: Noel.OConnor@dcu.ie
Dr. Rafal Kapela	Room N207	Pho. ext.: 6007	Email: Rafal.Kapela@dcu.ie
Dr. Kevin McGuinness	Room N207	Pho. ext: 6007	Email: Kevin.McGuinness@dcu.ie

Both co-ordinators have a strong research background in the field of image and video processing – see:

- CLARITY: Centre for Sensor Web Technologies – `http://www.clarity-centre.org/`
- Centre for Digital Video Processing – `http://www.cdvp.dcu.ie`).

1.2 Delivery

This module is delivered via lectures (three hours per week), and a dedicated email conference. Lectures will be based on these notes as much as possible. As such, students should bring a copy of the notes along to each lecture. Any additional material required during the semester (e.g. illustrative graphics, software demonstrations, etc.) will be made available either during lectures or as on-line resources. Notification of the availability of all on-line material will be made via the email conference. In addition to scheduled lectures, students are expected to devote at least three hours per week to independent study of the lecture notes and suitable text-books or reference material.

1.3 The Class Mailing List

This module has an associated class mailing list. All students enrolled in the module are automatically subscribed to this facility. You can post to this mailing list by addressing an email message to:

`ee554@list.dcu.ie`

In effect, this means that you will receive, by email, every message posted to the above address. The conference facility will be used by the co-ordinators to post important information such as notification

of additional on-line resources or time-table changes. In addition, students may use the conference as a means of submitting queries on any aspect of the module. Time permitting, the module co-ordinators will respond to queries but all students are encouraged to contribute with opinions and/or solutions.

In exceptional circumstances, if you have a query relating to the module, but which you do not wish to raise in the public forum of the class mailing list, you can address private email directly to the module co-ordinators.

1.4 Textbooks and Reference Material

Whilst lectures will draw primarily on these notes, additional reading may be required in order to fully understand the material covered. The following textbooks are useful starting points for more information on many of the topics covered in lectures. The texts are all available in the main lending section in the campus library. It is *not recommended* that students purchase any of these texts – there is no one text which adequately covers all aspects of this course, and they tend to be quite expensive! Reference texts particularly suitable for a particular topic will be indicated in lectures as topics are covered.

- **Bhaskaran V., Konstantinides K.,** *Image and Video Compression Standards*, Kluwer Academic, 1997.
- **Ghanbari M.,** *Video Coding: an introduction to standard codecs*, Institution of Electrical Engineers, printed by Short Run Press, 1999.
- **Puri A., Chen T.,** *Multimedia Systems Standards and Networks*, Marcel Dekker, 1999.
- **Rao, K.R., Hwang J.J.,** *Techniques and Standards for Image Video and Audio Coding*, Prentice Hall, 1996.
- **Furht B., Greenberg J., Westwater R.,** *Motion Estimation Algorithms for Video Compression*, Kluwer Academic Pub, 1997.
- **Westwater R., Furht B.,** *Real-time Video Compression: Techniques and Algorithms*, Kluwer Academic, 1997.

1.5 Assessment

This module is assessed via a combination of continuous assessment and a final end of semester exam. In order to **PASS** the module you must receive an *aggregate* mark of not less that 40%. If you **FAIL** the module you will have to repeat the exam in the Autumn diet of exams. There is a single assignment associated with this module. The assignment is in three parts and requires *significant* effort to complete satisfactorily. Further details on the assignment (e.g. deadline, submission format, etc.) and the exam will be provided as the module progresses via the class mailing list.

The allocation of marks between the assignment and the exam is as follows:

End of semester exam:		75%
Assignment:		
	Part 1: Implementation	15%
	Part 2: Investigation	7%
	Part 3: Applications	3%

1.6 Module Resources

1.6.1 EE554 Software Library

In order to assist students in completing the assignments associated with this module, a **C** software library is provided. The library specifies image, sequence and block data stuctures, a pixel data type and a number of functions allowing access and manipulation of these structures including image file input and output. The complete library consists of the following files:

- `ee554.h`
- `ee554.c`
- `ee554_lib.c`
- `ee554_lib.h`

which are available at:

`http://www.eeng.dcu.ie/~ee554/software/ver2.0/`

Also included in this directory are the files `example.c`, `example2.c` and `example.h` which together provide an example of using the EE554 software library. The first example (`example.c`) illustrates thresolding a grey level image in a block-wise fashion in order to produce a bi-level image. The second example (`example2.c`) illustrates calculating the mean image between each pair of images in a video sequence. In this way, the example software illustrates allocating and freeing memory for both individual images and complete video seqeunces, reading and writing PGM images and seqeuences, and accessing and processing image pixel data in a block-wise fashion.

> **Note:** The example programs are quite artificial in nature – e.g. there is no real reason to threshold in a block-wise fashion – but are designed so that they can be used with very little modification in the module assignments.

A `Makefile` for compiling the example software with the GNU `gcc` compiler is also provided. If you are working under Unix/Linux or you are using the recommended CygWin port of gcc this Makefile can be used directly. Alternatively, the software can be compiled using any C compiler. See the supplied `README.txt` file for hints on compiling using a number of different compilers.

Whilst the complete library consists of 4 files (and all of these are needed in order to compile the library), only `ee554.c` and `ee554.h` are of particular interest to the reader, since these contain the high-level functions required in order to complete the assignments. In the following section, a *very brief* overview of these two files is presented. For more information, on all files in the library (including the two examples), the reader is referred to the full documentation for the library, which can be found at the above URL.

Overview of `ee554.h`

The file `ee554.h` defines a data type for pixels, and data structures for images, video sequences, 8×8 blocks of pixels and 16×16 blocks of pixels – see below.

```
typedef int pel;
typedef float block_t[8][8];
typedef int mblock_t[16][16];

struct ee554_image
{
  int width;
  int height;
  pel *pels;
};
typedef struct ee554_image image;

struct ee554_sequence
{
  int length;
  image *images[1000];
};
typedef struct ee554_sequence sequence;
```

A pixel (`pel`) is defined as an integer data type, thereby allowing it to take on both positive and negative[1] values in the integer range of the platform being used. However, pixel values are written to disk (e.g. when writing a PGM image file) as unsigned characters (i.e. 8 bits), thereby restricting pixel values to the range 0 – 255.

An 8×8 block of pixels is defined as a 2-D array of floating point numbers. A 16×16 block of pixels is defined as a 2-D array of integers. This latter is referred to as a *macroblock*[2] (`mblock_t`) in order to differentiate it from an 8×8 block (`block_t`).

The image data structure consists of two integer elements for storing the `width` (i.e. the number of pixels per line) and the `height` (i.e. the number of lines) of an image. The actual pixel data is stored in an array of type `pel` which is `width`×`height` long. The origin of the picture is assumed to be the top left-hand corner of the image and pixel data is stored from left to right row by row starting at the origin.

The sequence data structure consists of an integer element for storing the `length` of the sequence corresponding to the number of images in the sequence. The actual images (up to a maximum of 1000) are stored as an array of pointers to image data structures. Each image can be accessed in the usual way array elements are accessed in C.

Also included in `ee554.h` are the prototypes of the high-level image manipulation functions contained in `ee554.c`.

Overview of `ee554.c`

The file `ee554.c` provides a set of functions which can be loosely categorised as follows:

1. Image functions
   - `image *pgm_load_image(char *filename)`
   - `void pgm_write_image(image *, char *)`
   - `image *clone_image(image * img)`
   - `void free_image(image *)`
   - `int get_image_width(image *)`
   - `int get_image_height(image *)`
2. Sequence functions
   - `sequence *load_sequence(char *seqname)`
   - `void free_sequence(sequence *seq)`
   - `int get_sequence_width(sequence *seq)`
   - `int get_sequence_height(sequence *seq)`
   - `int get_sequence_length(sequence *seq)`
   - `image *get_sequence_images(sequence *seq)`
3. Image utility functions
   - `void sub_image(image *, image *, image *)`
   - `void abs_sub_image(image *, image *, image *)`
   - `void create_diff_image(image *, image *, image *)`
4. Block extraction and insertion functions
   - `void get_block(image *, int, int, block_t)`
   - `void put_block(image *, int, int, block_t)`
5. Macroblock extraction and insertion functions

[1] This is necessary when subtracting two images to form a prediction residual, for example

[2] This is slightly mis-leading since this does not correspond to a real macroblock since there are no chrominance pixels included

- `void get_mblock(image *, int, int, mblock_t)`
- `void put_mblock(image *, int, int, mblock_t)`

6. PSNR function
   - `double calc_PSNR_image(image *, image *)`
7. Entropy function
   - `double calc_entropy_image(image *, int)`
8. Printing functions
   - `void print_block(block_t)`

1.6.2 Image and Video Test Suite

For the purposes of completing the first assignment, a number of test images are provided. The images are drawn from test suites commonly used in image processing and compression experiments. The images are in the Portable Grey Map (PGM) format (type P5). **C** functions for reading and writing this image format with the ee554 image data structure are provided in the software library. The test images are available at:

`http://www.eeng.dcu.ie/ẽe554/testdata/testimg`

For the purposes of completing the second assignment a test suite of video sequences is provided. The sequences used are drawn from those used in video standardisation activities. The sequences are supplied as four zip files (for.zip, mad.zip, tab.zip and coa.zip). Each file contains 50 successive *luminance* frames from a 25Hz video sequence. Each image is in PGM format (as above), and the following file naming convention is used:

`[root name][frame number (starting from 000)].pgm`

For example, for041.pgm is the 42^{nd} frame of the "Foreman" test sequence, mad000.pgm, is the 1^{st} frame of the "Mother and Daughter" test sequence, coa049.pgm is the 50^{th} frame of the "Coastguard" test sequence and tab001.pgm is the 2^{nd} frame of the "Table Tennis" sequence. The sequences are available at:

`http://www.eeng.dcu.ie/ẽe554/testdata/testimg`

1.6.3 Image Viewing Software

If working on a PC running Windows 9x or Windows NT, the application recommended for viewing PGM images is IrfanView32, which is a very fast freeware (for non commercial use) 32-bit graphic viewer. It is available at:
`http://www.irfanview.com`
If working on a Unix/Linux platform, then a number of image viewers supporting PGM should be already installed.

1.6.4 Compiler

The **GNU C Compiler** (`gcc`) is recommended for use with this module. For 32-bit PC platforms running Windows-9x or Windows-NT, the **Cygnus GNU-Win32** port of `gcc` is recommended:

`http://www.cygwin.com`

In addition to `gcc` itself, some implementation of the **C** Standard Library is required. Again the **GNU C Library** is recommended (this is included with the **gnu-win32** package[3]).

[3]In fact, gnu-win32 is a comprehensive package, with many additional tools over and above the compiler.

Chapter 2

The Challenge of Image and Video Compression

2.1 Introduction

The proliferation of personal computers, in conjunction with the continuous evolution of digital networks, has pushed the world towards a new digital age. Nowhere is this more prevalent than in the area of telecommunications and multimedia. The introduction of the telephone revolutionised the world by allowing instant voice communication. The expansion of digital communication services now means we can easily capture and exchange audio, visual and data-based information at a moment's notice. The digital revolution is being fueled by the growth in mobile communications technology. Initially introduced as a business person's toy, smartphones are now in huge demand from all sectors of society, allowing everyone to become a content producer. Looking to the future and the continuous evolution of mobile networks, it is clear that interactive multimedia applications are here to stay.

The arrival of this new digital and mobile age has brought with it many technological challenges. Multimedia data must be represented in a manner suitable for meeting the requirements placed upon it. It must be compressed to minimise storage costs and to ensure that it can be transmitted at the data rates possible on the networks (mobile or otherwise) which will carry the data. It needs to be protected against corruption. It should be processed in such a way as to facilitate reuse. In certain cases it must be encrypted. Sometimes, in order to protect ownership rights and for other legal considerations, it is necessary to incorporate non-removable qualities in the data. Digital visual information, corresponding to still images and video sequences, in particular, presents significant challenges, consisting as it does of large volumes of information to be processed in order to meet these requirements. In this module, we consider aspects of image and video compression in the context of some of these requirements, where compression refers to the process of obtaining as compact a representation as possible of the visual content.

2.2 Image and Video Fundamentals

An analogue image signal is generated when a camera scans a two-dimensional scene and converts it to an electrical signal. A video signal (e.g. corresponding to a moving scene), simply consists of successive scans producing multiple images. In this case, each image is referred to as a *frame* and the number of frames scanned per second is referred to as the video *framerate* and is measured in Hertz (Hz). If a frame is generated from a single scan of the scene, then this is referred to as *progressive* video. In this case, scanning starts at the top-left hand corner of the scene and proceeds line by line, ending at the bottom-right hand corner. Alternatively, two different images can be scanned at two different times, each with half the vertical spatial resolution, to produce two *fields*. The lines of these two fields can then be interleaved to produce a single frame. This is called *interlaced* video. Interlaced video is a convienent

Table 2.1: Video Data Rates

Format		Resolution	bits/pixel	frame rate (Hz)	Bandwidth (bps)
HDTV	(4:4:4)	1280×720	24	50	
CIF	(4:2:0)	352×288	8	30	
QCIF	(4:2:0)	176×144	8	8.33	

Note: bps = bits/second

way of doubling the perceived video framerate, without doubling the bandwidth required[1].

The scanning process actually generates three signals corresponding to the three primary colours red (R), green (G) and blue (B). In order to produce a digital image, these signals can be sampled and each sample quantised to certain number of bits. Each sample is referred to as a *picture element*, or *pixel* for short. Pixels are the atomic units of image processing and image/video compression. An uncompressed colour image consisting of 576 lines with 576 pixels per line represented with 8 bits/pixel requires, 576 × 576 × 8 × 3 = 972 KBytes of storage.

For video applications, the RGB signals are normally transformed into a new colour space called YUV, before being sampled. In this new space, the Y signal represents the *luminance* (or intensity) component of each image (thereby ensuring compatability with black and white video) whilst the U and V signals represent the *chrominance* (or colour) components. Most applications require standardised video formats. One commonly used format in video conferencing applications is called the *Common Intermediate Format* (CIF) which specifies progressive YUV video, with the U and V components horizontally and vertically subsampled by a factor of two[2]. The spatial resolution of Y component of CIF is 352×288 pixels (i.e. 352 pixels per line, 288 lines per frame) and the framerate is 30 Hz. The lower resolution *Quarter-Common Intermediate Format* (QCIF), which has a Y component resolution of 176×144, is used for very low bitrate applications. In fact, in order to further reduce the bandwidth required in very low bitrate applications, very often the framerate of the source video is lowered by dropping frames[3].

2.3 Image and Video Compression

Irrespective of the format used, the sheer volume of information to be processed when dealing with digital visual content[4] is substantial. Table 2.3 gives the data rates for three different uncompressed video formats suitable for applications corresponding to High Definition Television (HDTV), video conferencing, and video over a mobile network. This illustrates the need for efficient compression of video data. Luckily, image and video data is amenable to compression due to the considerable amount of *redundancy* present in such data. This redundancy takes the following forms:

- **Spatial:** within an image or a single frame of video there is usually significant *spatial* correlation between neighbouring pixels, which can be exploited for compression purposes.
- **Temporal:** between successive frames in a video sequence, there is usually a significant amount of *temporal* correlation which can be exploited for compression purposes.
- **Perceptual:** the human visual system is less sensitive to certain types of information and very often this information may be removed from the image/video (thereby obtaining compression) without a noticeable loss of quality.

[1]The first field produces every other horizontal line on the display. The second field then fills in the missing lines, lighting up just before the first field fades.

[2]The chrominance components are subsampled since the human eye is less sensitive to colour resolution

[3]Subjective tests have shown that the lowest framerate which still produces "acceptable" results is 5 Hz. Typically framerates of 15 Hz or 8.33 Hz are used for very low bandwidth applications.

[4]For the purposes of these notes, the term *visual content* is used to refer primarilly to still images and video sequences of natural scenes and not to other forms of visual content such as synthetic 3-D wire-frame models

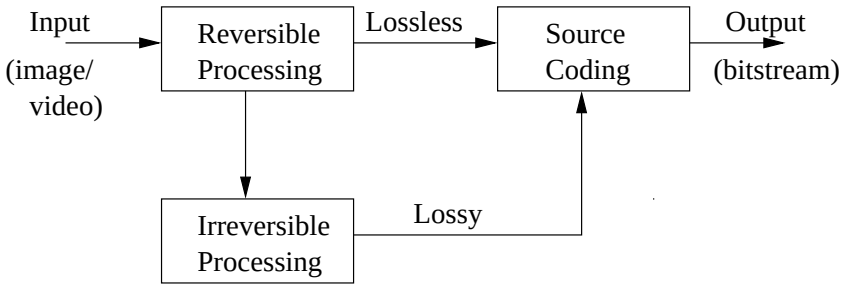


Figure 2.1: Relationship between and lossless and lossy compression

2.3.1 Lossless and Lossy Compression

There are two types of compression possible and the choice of which is used typically depends on the target application. Sometimes in still image compression applications, it is required that the output of the decoder is an exact replica of the original input to the encoder. In other words, the reconstructed image must have the exact same pixel values as the original image at every location. This means that the compression approach should not introduce any coding noise. An example of such an application is medical imaging (e.g. digitised X-rays) whereby legal considerations imply that no distortion whatsoever be introduced by the encoding process. In this case, *lossless* compression must be used. In this approach, the spatial correlation which exists in an image is exploited to obtain a more compact representation of the image without effecting its content.

More often than not, some distortion or coding noise can be tolerated. This is particularly true of video applications. In this case, a *lossy* compression approach can be employed. In such an approach, certain information in the input signal is actually discarded. Thus, the reconstructed image is not an exact replica of the original. However, the perceptual redundancy of the input is exploited by discarding only the information which is the least visually important to a human observer, thereby minimising the reduction in image quality. Of course, after this information is discarded, the remainder of the signal is losslessly encoded as compactly as possible by exploiting both spatial and temporal correlation. Since information is being "thrown out" during a lossy encoding process, successive applications of the process to the same image or video sequence significantly degrades image quality over time. The relationship between lossless and lossy encoding is illustrated in Figure 2.1.

An example of exploiting spatial correlation is using neighbouring pixels to predict the current pixel (or group of pixels) and only encoding the *prediction residual* (also referred to as the *prediction error*). When coding video sequences, temporal redundancy is exploited by using previous images to predict the current image and again only encoding the prediction residual. An example of exploiting perceptual redundancy would be removing the high frequencty components of the image or prediction error prior to lossless encoding.

2.4 The Role of International Standards

See Chapter 1 in [1] and Section 1.1 in [2]

Chapter 3

Information Theory and Lossless Compression

3.1 Introduction

This chapter focuses on the techniques used in lossless compression. By necessity, this requires an overview of the fundamentals of information theory. The chapter proceeds by first introducing the concept of entropy and building on this to introduce the concept of efficient source coding. Shannon's Lossless Coding Theorem is presented and it's importance is illustrated. The two approaches to source coding, Huffman and arithmetic coding, are then described. Finally, the chapter finishes with a very brief overview of existing international standards for lossless compression of still images. It should be noted that whilst only one chapter in these notes is devoted to lossless compression, the remainder of the notes focusses on tools and techniques, which when combined with the techniques described in this chapter, form the basis of all lossy compression standards.

3.2 Entropy

Let x be a discrete random variable with a set of possible *outcomes* $A_x = \{a_1, a_2, \ldots, a_i, \ldots, a_N\}$ having probabilities $p_1, p_2, \ldots, p_i, \ldots, p_N$, where $P\{x = a_i\} = P\{x_i\} = p_i$, $p_i \geq 0$ and $\sum_{i=1}^{N} p_i = 1$

In information theory terms, the random variable x and its associated probability distribution $p_1, p_2, \ldots, p_i, \ldots, p_N$ (i.e. the set of probabilities assigned to the set of outcomes) constitute an *information source*. The set of possible outcomes $A_x = \{a_1, a_2, \ldots, a_i, \ldots, a_N\}$ is sometimes called an *alphabet* with each element referred to as a *symbol*

A measure of the *information* contained in an outcome, termed the *self information*, was introduced by Hartley in 1932:

$$I(x_i) = \log_2 \frac{1}{p_i} = -\log_2 p_i \text{ bits}$$

In plain English, this means that the amount of information contained in an outcome is determined not only by the outcome itself, but also by how probable that outcome is. Intuitively, this should make sense to us. Consider, for example, the result of the National Lottery for a particular set of numbers as an information source with two possible outcomes: "win" or "no win". The outcome "no win" contains very little information, since it is an extremely *probable* event (unfortunately!). The outcome "win", the other hand, is an extremely *improbable* event and therefore contains a significant amount of information. This indicates that outcomes which are certain to occur convey no information i.e. if $p_i = 1$ (with $p_j = 0, \forall j \neq i$), then $I(x_i) = 0$[1].

$I(x_i)$ is only a measure of the information contained in a single outcome. The average information content of the source as a whole is defined as the *entropy*[2]:

[1] This of course gives rise to (terrible) information theory jokes such as $I(\text{death}) = 0$ and $I(\text{taxes}) = 0$.

[2] The term entropy was originally used in thermodynamics where it refers to a measure of the uncertainty or disorder of a system

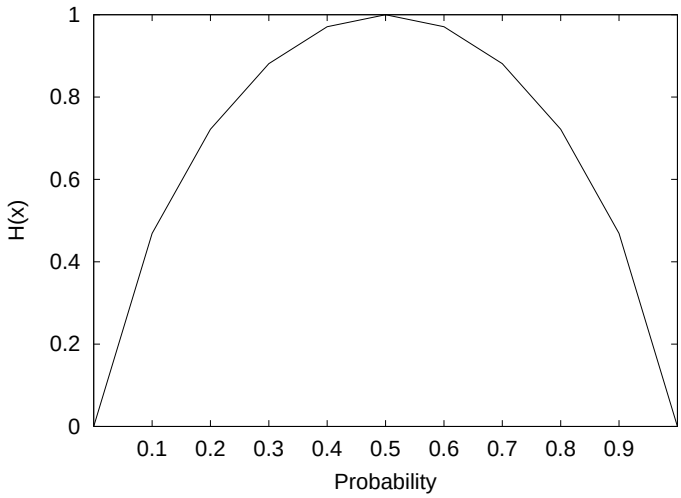


Figure 3.1: Entropy of a Binary Symmetric Source (BSS)

$$\begin{aligned} H(x) &= E\{I(x)\} \\ &= \sum_{i=1}^{N} p_i I(x_i) \\ &= -\sum_{i=1}^{N} p_i \log_2 p_i \text{ bits/symbol} \end{aligned}$$

with the convention for $P\{x_i\} = p_i = 0$ that $p_i \log_2 p_i = 0$ since $\lim_{y \to 0^+} y \log_2 y = 0$. Note that entropy is measured in bits/symbol which is interpreted as the average information required for each outcome (or each symbol in the source's alphabet).

A Binary Symmetric Source (BSS)

A Binary Symmetric Source (BSS) is an information source where there are only two possible outcomes, i.e. if x_i can only take values a_1 and a_2. If $P\{x = a_1\} = p$ then $P\{a_2\} = 1 - p$. In this case, the entropy is:

$$H(x) = -p \log_2 p - (1-p) \log_2 (1-p)$$

This function is illustrated in Figure 3.1. From this, we can infer the following *general* properties of $H(x)$:

- $H(x)$ is a continuous function of $P\{x_i\}$
- $H(x)$ is non-negative and is 0 only for cases where $P\{x_i\} = 0$ for all values of x_i except one
- $H(X)$ is maximum when all outcomes are equally likely.
- $H(x)$ increases with N

The above properties indicate that if an information source has N symbols (possible outcomes), then:

$$0 \leq H(x) \leq \log_2 N$$

with maximum entropy obtained when all N outcomes have equal probabilities. We define *redundancy* $R(x)$ as the difference between an information source's maximum entropy and the actual amount of information in the source.

Table 3.1: Codewords for Example 3.1

Symbol	Prob.	Codeword
s_1	$p_1 = 0.25$	00
s_2	$p_2 = 0.25$	01
s_3	$p_3 = 0.25$	10
s_4	$p_4 = 0.25$	11

Table 3.2: Initial codewords for Example 3.2

Symbol	Prob.	Codeword
s_1	$p_1 = 0.5$	1
s_2	$p_2 = 0.25$	01
s_3	$p_3 = 0.125$	011
s_4	$p_4 = 0.125$	010

$$\mathrm{R}\{x\} = \log_2(N) - H(x)$$

Example 3.1 $N = 4$, $p_1 = p_2 = p_3 = p_4 = 0.25$ *The entropy is* $H(x) = 4 \times (-0.25)\log_2 0.25 = 2$ *bits/symbol . In other words, if we were to transmit these symbols, then in order to convery the required information, we would need to allocate two bits for each symbol.*

Example 3.2 $N = 4$, $p_1 = 0.5, p_2 = 0.25, p_3 = p_4 = 0.125$ *The entropy is* $H(x) = -(0.5)\log_2 0.5 - (0.25)\log_2 0.25 - (0.25)\log_2 0.125 = 1.75$ *bits/symbol . In this case, due to the different probabilities of the symbols, less than two bits per symbol are required on average.*

3.3 Source Coding

If we are to allocate a certain number of bits to a symbol, then clearly we should decide on what those bits will actually be (i.e. the specific sequence of 1's and 0's to use for each symbol). In effect we must assign each symbol a binary codeword. For Example 3.1 of the previous section we might assign these codewords as illustrated in Table 3.1.

The process of replacing symbols with binary codewords is known as *source coding*. In general, the objective of source coding is to reduce the amount of bits required to convey the information contained in the source. In order to measure the effectiveness of source coding we introduce a measure for the average coding rate which we can then compare to the entropy of the information source. The average coding rate is defined as:

$$R = \sum_{i=1}^{N} p_i \times l_i \text{ bits/symbol}$$

where l_i is the length in bits of the codeword for s_i. Using this measure, the codewords in Table 3.1 gives $R = 0.25 \times 2 \times 4 = 2$ bits/symbol which equals H(x) in this case (since all symbols have equal probabilities), thereby indicating efficient source coding.

For Example 3.2, of the previous section, rather than assigning a fixed length (two bit) codeword to each symbol, we might decide to assign *variable length* codewords since we know that the information source needs less than two bits for each symbol on average. A typical assignment is illustrated in Table 3.2. In this case $R = (0.5 \times 1) + (0.25 \times 2) + (0.125 \times 3 \times 2) = 1.75$ bits/symbol , which again equals $H(x)$ for this information source, since all the symbol probabilities are exact negative exponents of two. Again, this indicates efficient source coding.

On closer inspection, however, we might notice that there is a problem with the codewords in Table 3.2, particularly if we were to encode and transmit a sequence of these symbols. In such a scenario, the first step on the the decoding side, would be to *parse* the received bitstream. Parsing refers to the

Table 3.3: Improved codewords for Example 3.2

Symbol	Prob.	Codeword
s_1	$p_1 = 0.5$	1
s_2	$p_2 = 0.25$	01
s_3	$p_3 = 0.125$	001
s_4	$p_4 = 0.125$	000

process of breaking a bitstream up into its component codewords before decoding each codeword to an individual symbol. An *instantaneously parseable codeword* is one which can be parsed as soon as the decoder receives the last bit of the codeword. Such codewords must obey the *prefix condition*, which states that the codeword cannot appear as the initial part (prefix) of another codeword. The codewords in the above table do not satisify this condition. For example, if the sequence transmitted were s_1, s_4, s_3, s_2 then the bitstream would be 101001101. The decoder would be able to recognise the first '1' as the codeword indicating symbol s_1, but would subsequently be unable to distinguish '01' as being either a codeword in it own right indicating s_2, or as part of the codewords for s_3 or s_4. This problem would not arise if we used the codewords of Table 3.3 which are instantaneously parseable.

Shannon's Lossless Coding Theorem[3], also known as the noiseless source coding theorem, states that an instantaneous code (i.e. made up of instantaneously parseable codewords) can be found which encodes a source of entropy $H(x)$ with B_s bits/symbol such that:

$$B_s \geq H(x)$$

Typically, the longer the sequence of input symbols, the closer B_s will be to $H(x)$. Note that whilst this does not tell us how to find this code, it is nonetheless an extremely useful result (see section 3.3.2).

3.3.1 Huffman Coding

The codewords in Table 3.3 are in fact the *Huffman Codes*[4] for the associated information source. Generating Huffman codes requires generating a Huffman code tree as follows:

1. Form the tree:
   (a) Sort the symbols by their probabilities
   (b) Merge the two smallest probabilities by adding them and produce a new node in the tree
   (c) Repeat until only a singe node is reached
2. Assign bits:
   (a) Traverse the tree from the root to the leaf nodes assigning each branch encountered a one or zero.

Symbol codewords are obtained from the bits assigned to the branches encountered when traversing the tree from the root to the leaf node corresponding to the symbol.

Example 3.3 *Consider the 8×8 block of pixels shown in Figure 3.2. The probabilities associated with these pixel values can be calculated as shown in Table 3.4. Sorting the probabilities, and combining is illustrated in Figure 3.3(a). The assignment of bits is illustrated in Figure 3.3(b) and the resultant codes are shown in the rightmost column of Table 3.4*

[3]Claude Shannon, who singlehandedly developed the fundamentals of information theory, was an American mathematical engineer who worked in Bell Labs in the 1940's. He passed away on Feb. 24th 2001.

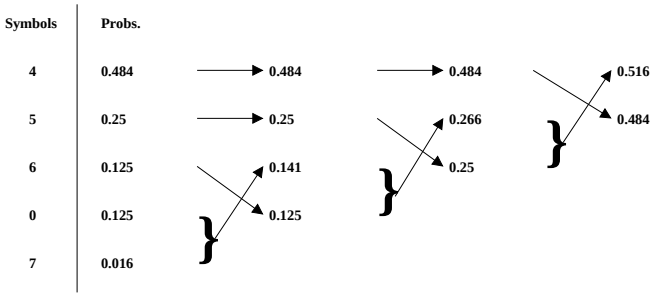
[4]First published in 1952, Huffman coding was developed by David Huffman, a student at MIT, in response to a challenge from his professor. As a result, Huffman was not required to sit the final exam for the module!

4	4	4	4	4	4	4	0
4	5	5	5	5	5	4	0
4	5	6	6	6	5	4	0
4	5	6	7	6	5	4	0
4	5	6	6	6	5	4	0
4	5	5	5	5	5	4	0
4	4	4	4	4	4	4	0
4	4	4	4	4	4	4	0

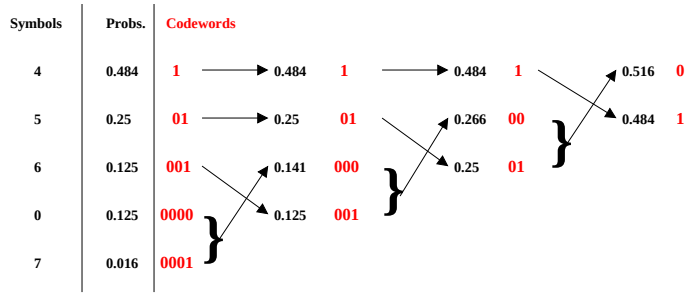
Figure 3.2: An 8×8 block of pixels from a grey-level image

Table 3.4: Probabilities and codes for the pixels of Figure 3.2

Grey Level	Num. of Pixels	p_i	$-p_i \log_2 p_i$	$p_i l_i$	Code
0	8	0.125	0.375	0.5	0000
4	31	0.484	0.507	0.484	1
5	16	0.25	0.500	0.5	01
6	8	0.125	0.375	0.375	001
7	1	0.016	0.095	0.064	0001
			1.852	1.923	



(a) Sorting and combining probabilities



(b) Assigning bits

Figure 3.3: Generating the Huffman codewords for Example 3.3

Table 3.5: Example LUT for Huffman decoding

L-bit code	l_i	Symbol
000	3	s_4
001	3	s_3
010	2	s_2
011	2	s_2
100	1	s_1
101	1	s_1
110	1	s_1
111	1	s_1

Exercise 3.1 *Calculate the Huffman codes for the 5 symbols* a, b, c, d, e *with probabilities* $0.25, 0.25, 0.2, 0.15, 0.15$. *Calculate the entropy of the information source and the average coding rate using Huffman coding. Calculate the efficiency of these codewords.* ***Solution:*** *Codewords:* $01, 10, 11, 000, 001$*, Entropy:* 2.285*, Rate:* 2.3 *bits/symbol*

Huffman coding assumes that both the encoder and the decoder have the symbol-to-codeword mapping table. The Huffman encoding process is very straighforward since the required mapping is directly available. The decoding process, on the other hand, is slightly more complex. The two main approaches to decoding are described in the following sections.

Serial Bit Decoding

This approach requires that the decoder have available the Huffman code tree. The input stream is read one bit at a time, each bit indicating what branch in the tree should be taken (starting from the root). Each bit is discarded after being read. When a leaf node is reached, then the corresponding symbol is deocoded. The process then continues, starting from the root node again. This approach has a fixed input data rate, but a variable symbol decode rate.

LUT-based Decoding

This approach requires construction of a LUT (look-up-table) with 2^L entries where L is the length of the longest possible Huffman code. If c_i is the codeword for symbol s_i, and c_i is l_i bits long, then an L bit address into the LUT is formed for s_i, the first l_i bits of which are c_i, whilst the remainder will be all possible combinations of ones and zeros. In this way, each s_i has 2^{L-l_i} entries in the table. For each entry in the LUT, the corresponding value of l_i is also stored. Decoding then proceeds as follows:

1. Read L bits from the bitstream into a buffer
2. Use the buffer as an adress into the LUT - this gives symbol s_i
3. Remove the first l_i bits from the buffer (corresponding to c_i) and append $L - l_i$ bits from the bitstream leaving L bits in the buffer once again
4. repeat the previous two steps until no more symbols need to be decoded.

An LUT for the codewords in Table 3.3 are shown in Table 3.5. The advantage of this approach is that it is fast with a fixed symbol decode rate (albeit with a variable input data rate). A disadvantage is the large amount of storage required for large alphabets.

Exercise 3.2 *Create an LUT for the Huffman codes of Exercise 3.1 and Example 3.3*

3.3.2 Huffman Coding and Symbol Grouping

Consider an information source with two symbols s_1 and s_2 , the probabilities of which are $p_1 = 0.8$ and $p_2 = 0.2$, respectively, giving $H(x) \approx 0.72$. Using Huffman encoding, we have to assign a single

Table 3.6: Huffman codes for groups of symbols

Symbols	Prob.	Codeword
s_1, s_1	$p_1 = 0.64$	0
s_1, s_2	$p_2 = 0.16$	11
s_2, s_1	$p_3 = 0.16$	100
s_2, s_2	$p_4 = 0.04$	101

bit codeword to each of the outcomes giving an average coding rate of $R = (0.8 \times 1) + (0.2 \times 1) = 1$ bit/symbol . This indicates quite inefficient source coding!

This situation can be improved if we decide to encode successive *groups* of outcomes. For example, we may decide that the successive outcomes s_1, s_1 are more likely than s_1, s_2, which are equally as likely as s_2, s_1, both of which are more likely than s_2, s_2. We might assign suitable probabilities and calculate Huffman codes as illustrated in Table 3.6. The codes of Table 3.6 give an average coding rate of $R = (0.64 \times 1) + (0.16 \times 2) + (0.16 \times 3) + (0.04 \times 3) = 1.56$ bits per *two* symbols, which is equivalent to 0.78 bits/symbol. Clearly, by grouping successive symbols we have made the source coding more efficent.

Incidentally, from Shannon's theorem, we know that the best we can do in this case is 0.72 bits/symbol, which is only an improvement of 8% (approximately). We might therefore decide not to devote any extra effort to further improving this by extending the approach to groups of three. This serves to underline the importance of Shannon's theorem. An alternative statement of Shannon's theorem is:

$$\min\{R\} = H(x) + \epsilon$$

where ϵ is a positive value which can be made arbitrarily as close to 0 at the cost of encoding delay.

3.3.3 Arithmetic Coding

There are a number of disadvantages associated with Huffman encoding, which led to the development of a new approach to source coding. In this section this new approach is described.

The first drawback of Huffman Coding is that it requires that an integral number of bits be allocated for each symbol. This implies a lower bound of 1 bit/symbol, and makes the approach (on its own) unsuitable for encoding bi-level images[5], for example.

Another drawback is that Huffman encoding does not easily facilitate adaptive coding since the probability model and the coding process are interdependent. Adaptive coding is where frequency counts of the symbols are updated as a sequence is processed and the probabilities associated with each symbol are updated accordingly. In this way, the source coding adapts to the particular sequence of symbols to be encoded. In Huffman coding, any change in symbol probabilities requires that all the Huffman codewords be re-calculated.

The final drawback of Huffman encoding is that the approach is only optimal when the symbol probabilities are negative exponents of two. In general, this is not the case.

Example 3.4 *Assume we have three symbols* $\{a, b, c\}$ *with probabilities* $\{0.9, 0.05, 0.05\}$ *respectively. This gives the Huffman codes illustrated below:*

Symbol	*Prob.*(p_i)	$\log_2(\frac{1}{p_i})$	*Huffman Code*
a	*0.9*	*0.152003*	*0*
b	*0.05*	*4.321928*	*10*
c	*0.05*	*4.321928*	*11*

This means the entropy is $H(x_i) = \sum_{i=1}^{N} -p_i \log_2 p_i = 0.9 \times 0.152003 + 2 \times (0.05 \times 4.321928) \approx 0.569$ *bits/symbols . However, the average coding rate is* $R = \sum_{i=1}^{N} p_i \times l_i = 0.9 \times 1 + 2 \times (0.05 \times 2) = 1.1$ *bits/symbol!*

[5]Bi-level images are images in which pixels can only take on only two values, '1' or '0', corresponding to 'black' and 'white' respectively – see section 3.4

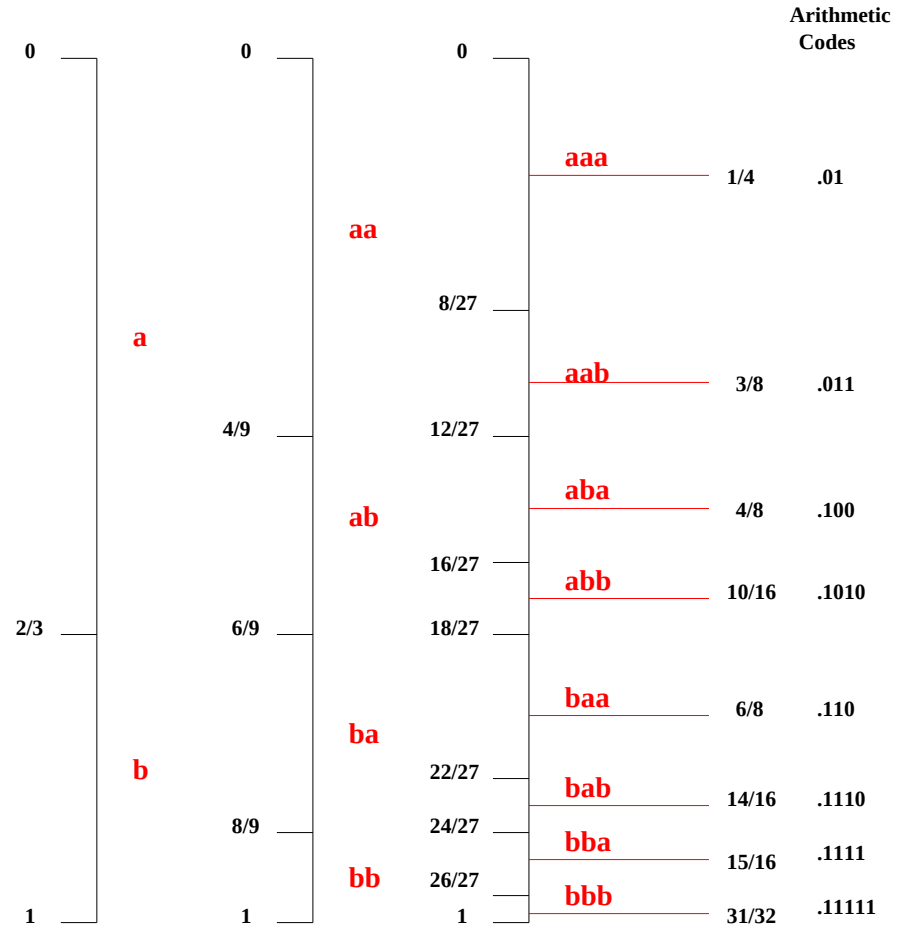


Figure 3.4: Arithmetic encoding example

Example 3.5 *If we have two symbols $s_1 = a$ and $s_2 = b$ with probabilities $p_a = 0.75$ and $p_b = 0.25$, then $H(x) \approx 0.81$ bits/symbol, but with Huffman coding we must allocate at least one bit per symbol. Thus, if we were to transmit the message 'aaaaab', we would require 6 bits.*

In the 1970's a new form of source coding, called *arithmetic coding* was developed which addresses these limitations of Huffman coding. Arithmetic coding benefits from treating groups of symbols (c.f. section 3.3.2 for Huffman coding) whilst maintaining a symbol-by-symbol approach. The approach assigns a single codeword to each group of symbols, where the codeword represents a half-open interval on $[0.0, 1.0)$. By assigning enough precision bits, one interval can be distinguished from another, thereby allowing the sequence of symbols to be decoded. Symbols with higher probabilities correspond to larger intervals, thereby requiring less precision bits. Arithmetic encoding is best understood using the following example.

Example 3.6 *If we have two symbols $s_1 = a$ and $s_2 = b$ with probabilities $p_a = \frac{2}{3}$ and $p_b = \frac{1}{3}$, then using arithmetic coding, we can map all length 3 messages ('aaa', 'aab', 'aba', ...) to intervals in $[0, 1)$. We can represent each interval with a single number in that interval. For this, we choose a number which can be represented compactly as a binary fraction – see Figure 3.4. From Figure 3.4, we can see that the first interval, corressponding to the message 'aaa' requires only $\frac{2}{3}$ bits/symbol, whereas the last interval, corresponding to message 'bbb' requires $\frac{5}{3}$ bits/symbol.*

Once the symbol probabilities are known, each symbol can be assigned a range within $[0.0, 1.0)$, which corresponds to its probability of occurrence in the cumulative density function. For example, symbol b with probability $\frac{1}{3}$ in Figure 3.4 is assigned the range $[\frac{2}{3}, 1.0)$. Thus, each symbol has a lower bound

and upper bound associated with its range. The first symbol encountered narrows the interval $[0.0, 1.0)$ to that symbol's range. For example, if the first symbol encoded in Figure 3.4 is 'a', then we know that our final output number will be restricted to the range $[0.0, \frac{2}{3})$, irrespective of what symbols are subsequently encoded. Subsequent symbols further restrict the current interval based on their lower and upper bounds. For example, if the second symbol encoded in Figure 3.4 is 'b' then the interval $[0.0, \frac{2}{3})$ is restricted to $[\frac{4}{9}, \frac{6}{9})$ (i.e. the 'b' portion of the current interval).

Decoding proceeds by reversing this process. The decoder receives a number between $[0.0, 1.0)$ and checks which symbol's range contains this number. This symbol is then decoded. Since, the lower and upper bounds of this symbol are known, their effects on the encoded number can be reversed. This gives, a new number to which we can apply the same process, and so on until there are no more symbols to decode. In fact, the decoder needs to be told explicitly when to stop decoding. As such, the input symbol alphabet typically contains a special symbol which is used as an "end of message" marker. Once the decoder decodes this symbol, it knows that the message is finished.

Arithmetic Coding and Decoding Algorithms

In the following algorithm, *low* and *high* store the bounds of the current interval and $range_H(symbol)$ is the upper bound of a symbol's interval on $[0.0, 1.0)$, whereas $range_L(symbol)$ is the lower bound.

The Arithmetic Encoding Algorithm

$low = 0.0$
$high = 1.0$
while there is still input symbols
{
    get input symbol
    $range = high - low$
    $high \leftarrow low + range \times range_H(symbol)$
    $low \leftarrow low + range \times range_L(symbol)$
}
output *low*

The Arithmetic Decoding Algorithm

get *encodednumber*
do
{
    find symbol whose range contains encoded number
    output symbol
    $range = range_H(symbol) - range_L(symbol)$
    $encodednumber \leftarrow encodednumber - range_L(symbol)$
    $encodednumber \leftarrow encodednumber \div range$
} while more symbols to decode

Example 3.7 *Arithmetic Encoding*
Assume we have two symbols, a and b with probabilities $\frac{3}{4}$ *and* $\frac{1}{4}$ *respectively.*

Symbol	*Prob.*	*Interval*
a	$\frac{3}{4}$	$[0.0, \frac{3}{4})$
b	$\frac{1}{4}$	$[\frac{3}{4}, 1.0)$

The arithmetic encoding steps for the message 'aaaaab' are illustrated below. Note that here we assume that the decoder "magically" knows when to stop.

		$low = 0$ $high = 1$
Symbol 1	a	$range = 1 - 0 = 1$ $high = 0 + 1 \times \frac{3}{4} = \frac{3}{4}$ $low = 0 + 1 \times 0 = 0$
Symbol 2	a	$range = \frac{3}{4} - 0 = \frac{3}{4}$ $high = 0 + \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$ $low = 0 + \frac{3}{4} \times 0 = 0$
$\vdots$	$\vdots$	
Symbol 5	a	$range = \frac{81}{256} - 0 = \frac{81}{256}$ $high = 0 + \frac{81}{256} \times \frac{3}{4} = \frac{243}{1024}$ $low = 0 + \frac{3}{4} \times 0 = 0$
Symbol 6	b	$range = \frac{243}{1024} - 0 = \frac{243}{1024}$ $high = 0 + \frac{243}{1024} \times 1 = \frac{972}{4096}$ $low = 0 + \frac{243}{1024} \times \frac{3}{4} = \frac{729}{4096}$

The final value for low is $\frac{729}{4096} = .001011011001$*, whilst the final value for high is* $\frac{972}{4096} = .001111001100$*. We can represent this interval with a single value – we choose the value .00110 which is in this range. Thus, our arithmetic code consists of 5 bits for six symbols! Compare this to the result of Example 3.5.*

Example 3.8 *Arithmetic Encoding and Decoding*
Given the following symbols and cumulative probability density, we trace the encoding and decoding processes (in decimal numbers) for the message "BILL GATES". Note again that we assume that the decoder knows when to stop.

Symbol	*Prob.*	*Interval*
$SPACE$	$\frac{1}{10}$	$[0.0, 0.1)$
A	$\frac{1}{10}$	$[0.1, 0.2)$
B	$\frac{1}{10}$	$[0.2, 0.3)$
E	$\frac{1}{10}$	$[0.3, 0.4)$
G	$\frac{1}{10}$	$[0.4, 0.5)$
I	$\frac{1}{10}$	$[0.5, 0.6)$
L	$\frac{2}{10}$	$[0.6, 0.8)$
S	$\frac{1}{10}$	$[0.8, 0.9)$
T	$\frac{1}{10}$	$[0.9, 0.1)$

Arithmetic encoding:

Symbol	*low*	*high*
	0.0	*1.0*
B	*0.2*	*0.3*
I	*0.25*	*0.26*
L	*0.256*	*0.258*
L	*0.2572*	*0.2576*
$SPACE$	*0.25720*	*0.25724*
G	*0.257216*	*0.257220*
A	*0.2572164*	*0.2572168*
T	*0.25721676*	*0.2572168*
E	*0.257216772*	*0.257216776*
S	*0.2572167752*	*0.2572167756*

The final low value of 0.2572167752 will uniquely encode the message "BILL GATES"
Arithmetic Decoding:

Encoded Number	*Symbol*	*low*	*high*	*range*
0.2572167752	*B*	*0.2*	*0.3*	*0.1*
0.572167752	*I*	*0.5*	*0.6*	*0.1*
0.72167752	*L*	*0.6*	*0.8*	*0.2*
0.6083876	*L*	*0.6*	*0.8*	*0.2*
0.041938	*SPACE*	*0.0*	*0.1*	*0.1*
0.41938	*G*	*0.4*	*0.5*	*0.1*
0.1938	*A*	*0.2*	*0.3*	*0.1*
0.938	*T*	*0.9*	*1.0*	*0.1*
0.38	*E*	*0.3*	*0.4*	*0.1*
0.8	*S*	*0.8*	*0.9*	*0.1*
0.0				

Practical Considerations

Due to the relatively complex nature of the process, initially arithmetic encoding may appear to be completely impractical. However, it transpires that arithmetic encoding can be performed very efficiently using 16/32 bit integer mathematics (i.e. with no floating point operations). The approach adopted is an incremental transmission scheme whereby bits are transmitted as they become available, thereby also neatly avoiding the drawback of having to wait until an entire message is encoded.

The first simplication is to use the value $0.999\ldots$ for the initial value for high rather than 1.0. In binary arithmetic encoding, this corresponds to using the value $0.111\ldots$. In fact, we can deal only with the fractional part and therefore need only use integer numbers. Thus, in binary arithmetic encoding, *high* initially stores 0xFFFF, whilst *low* stores 0x0000. With each symbol encoded, the most significant bit of both *high* and *low* is examined. If these bits are the same, then since we know that the interval can only ever be narrowed, we know they will never change form here on. Thus, we can output a '1' or '0' (depending on the value of these bits). Since these bits in *high* and *low* will never change, we can shift them out and shift in a '1' to *high* and a '0' to *low*, thereby avoiding a loss of precision. In the decimal example of Example 3.2, this corresponds to outputting a '2' after encoding 'I', a '5' after 'L', a '7' after 'L', ..., a '5' after 'S' and flushing the *low* buffer to cumulatively produce the output *low* value.

Of course, even with these simplifications, it should be clear that arithmetic coding is more computationally intensive and requires more memory than Huffman coding. Usually, Huffman coding is more attractive when simplicity in an encoder and decoder is a concern [6].

3.3.4 Run-length Coding

Run-length coding is not a source coding technique on its own. Rather, it can be considered to be a *pre-processing* step on the original information source prior to source coding. The preprocessing is carried out in order to transform the symbols in the original information source into new *coding events*, the entropy of which is lower than that of the original source. The new coding events consist of a set of *(run,symbol)* values, where 'run' indicates the number of contiguous occurences of 'symbol'.

Example 3.9 *Consider the block of pixels illustrated in Figure 3.5. Typically, horizontal or vertical runs on each line or column of pixels are used to form run-length coding events. The horizontal runs for Figure 3.5 would produce the following coding events:*

(8,4)
(1,4) *(5,5)* *(2,4)*
(1,4) *(5,5)* *(2,4)*
(1,4) *(5,5)* *(2,4)*
(1,4) *(5,5)* *(2,4)*
(1,4) *(5,5)* *(2,4)*
(8,4)
(8,4)

[6] In fact, the more recent image and video compression standards typically provide for either arithmetic or Huffman coding and leave the selection of which is used as a design choice in the encoder.

4	4	4	4	4	4	4	4
4	5	5	5	5	5	4	4
4	5	5	5	5	5	4	4
4	5	5	5	5	5	4	4
4	5	5	5	5	5	4	4
4	5	5	5	5	5	4	4
4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4

Figure 3.5: A sample 8×8 block of pixels to illustrate run length coding

which corresponds to 36 numbers which must undergo source coding, as oppose to 64 without run-length encoding. Taking the vertical runs in this case also gives 36 numbers.

Example 3.10 *Consider the block of pixels illustrated in Figure 3.2. Taking the horizontal runs would produce the following coding events:*

(7,4)	*(1,0)*						
(1,4)	*(5,5)*	*(1,4)*	*(1,0)*				
(1,4)	*(1,5)*	*(3,6)*	*(1,5)*	*(1,4)*	*(1,0)*		
(1,4)	*(1,5)*	*(1,6)*	*(1,7)*	*(1,6)*	*(1,5)*	*(1,4)*	*(1,0)*
(1,4)	*(1,5)*	*(3,6)*	*(1,5)*	*(1,4)*	*(1,0)*		
(1,4)	*(5,5)*	*(1,4)*	*(1,0)*				
(7,4)	*(1,0)*						
(7,4)	*(1,0)*						

which consist of 68 numbers to encode! However taking the vertical runs produces only 52 numbers to encode:

(8,4)						
(1,4)	*(5,5)*	*(2,4)*				
(1,4)	*(1,5)*	*(3,6)*	*(1,5)*	*(2,4)*		
(1,4)	*(1,5)*	*(1,6)*	*(1,7)*	*(1,6)*	*(1,5)*	*(2,4)*
(1,4)	*(1,5)*	*(3,6)*	*(1,5)*	*(2,4)*		
(1,4)	*(5,5)*	*(2,4)*				
(8,4)						
(8,0)						

Run-length coding is useful when we know that certain runs of symbols or more likely than others. A good example is bi-level image coding. In the case of a facsimile image (which is perhaps the best known type of bi-level image), then we might expect long runs of '0' corresponding to the paper, interspersed with smaller runs of '1', corresponding to text. Note, however, that whilst run-length encoding is suitable in this context, it is very susceptible to transmission errors since a change in the run length or symbol (e.g. due to bit errors) may change the entire image.

Example 3.11 *Consider the block of pixels illustrated in Figure 3.6(a). Taking the horizontal runs across this block forms the following 16 coding events, correspondint to 32 numbers to code:*

(2,1)	*(6,0)*
(2,1)	*(6,0)*
(2,1)	*(6,0)*
(2,1)	*(6,0)*
(2,1)	*(6,0)*
(2,1)	*(6,0)*
(6,1)	*(2,0)*
(6,1)	*(2,0)*

Exercise 3.3 *Calculate the horizontal and vertical run-length pairs for the block of pixels shown in Figure 3.6(b).*

1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0

(a)

1	1	0	0	0	0	1	1
1	1	1	0	0	1	1	1
0	1	1	0	1	1	1	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	1	1	0	1	1	0
1	1	1	0	0	1	1	1
1	1	0	0	0	0	1	1

(b)

Figure 3.6: Two 8×8 blocks of pixels from a bi-level image

3.3.5 Lempel-Ziv Coding

3.4 Lossless Image Compression Standards

In this section, we briefly review the major international standards for *lossless* image compression. There are two main bodies responsible for the development of international standards in this area. The International Telecommunications Union (ITU-T) develop standards dealing with the coding and transmission of signals over public telecommunications networks. This group has standardised compression approaches for bi-level images which are suitable for facsimile applications (see section 3.4.1). The International Standards Organisation (ISO) has collaborated with ITU-T in order to develop lossless image compression standards targeted at both bi-level and grey-level images (see section 3.4.2 and section 3.4.3). All these standards are based on the techniques described in previous sections of this chapter.

3.4.1 ITU-T Facsimile Compression Standards

There are two ITU-T standards for image compression for facsimile applications. The first is ITU-T Rec. T4, also referred to as Group 3. This approach uses a combination of run-length encoding and Huffman encoding in two modes:

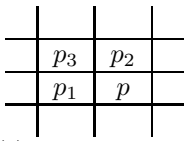
- Modified Huffman (MH): In this coding mode, run-length encoding is used to form runs of ones and zeroes for each line in the image to be compressed. Huffman coding is then applied to these (run,symbol) pairs. Different Huffman codes are used for runs of ones and zeros since these have different characteristics. After each line is encoded, a special end-of-line (EOL) symbol is encoded for error detection purposes.
- Modified Read (MR): In this coding mode, pixel values from the previous line are used as predictors for the current pixels to be encoded. The prediction residual is then encoded using Huffman coding. Again, the EOL symbol is used to signal the end of a line. In order to prevent error propagation, MR mode is periodically interspersed with MH mode.

The second ITU-T standard for facsimile applications is referred to as ITU-T Rec. T6, also referred to as Group 4. This uses a form of coding known as Modified Modified Read (MMR), which as the name[7] suggests, is a simplification of the MR coding approach. In this case, error protection is sacraficed in favour of improved compression efficiency.

3.4.2 The JBIG Standard

The Joint Binary Image Experts Group (JBIG) compression standard, was developed jointly by both ITU-T and ISO. The standard targets efficient lossless compression of bilevel images which may be either business document type images or grey-scale images of natural scenes which are rendered as bi-level images. The ITU-T facsimile compression standards are not suitable in this context since these different types of bi-level images have very different characteristics. For example, whilst we expect long

[7] A modification of this approach was proposed as a shape coding tool in response to the MPEG-4 Call for Proposals (CfP). Predictably enough, the new approach was given the name Modified Modified Modified Read (MMMR).



(a) The local context

$P = 0$	$P = \frac{p_1+p_2}{2}$
$P = p_1$	$P = p_1 + p_2 - p_3$
$P = p_2$	$P = p_1 + \frac{p_2-p_3}{2}$
$P = p_3$	$P = p_2 + \frac{p_1-p_3}{2}$

(b) Prediction residual: $R = P - p$

Figure 3.7: Lossless JPEG encoding

runs of ones and zeroes in a scanned image of a business document, we expect these runs to be much shorter in a bi-level natural image.

The approach employed uses adaptive arithmetic encoding. The *modelling* step estimates the probability of the next symbol to encoded based on a *context* consisting of local pixels. This probability is then used to drive the arithmetic encoder as explained in section 3.3.3. By using casual modelling (i.e. only using previously encoded pixels) the decoder can mimic the modelling operation of the encoder without any aditional information. Note that JBIG can be applied to grey-scale images by treating each grey-level image plane as a bi-level image.

3.4.3 The Lossless JPEG Standard

The Joint Photographic Experts Group (JPEG) image compression standard is described in detail in Chapter 6 in the context of lossy image compression. However, there is also a flavour of JPEG which allows lossy compression of images. In this approach, JPEG eschews the transform-based approach to compresion as described in Chapter 5 and uses only the techniques described in this chapter.

For each pixel to be encoded, a prediction is formed based on a context of previously encoded pixels. The context used is illustrated in Figure 3.7(a), where p indicates the current pixel to be encoded and p_1, p_2, p_3 are neighbouring pixels. There are a number of different ways specified for forming the prediction based on the context, and these are illustrated in Figure 3.7(b). The particular prediction method used is encoded as side-information for each scan line.

To encode the prediction residual, a $(length, magnitude)$ pair is formed[8]. The *length* indicates the number of bits used to encode the magnitude of the residual. A static Huffman code is used for this value. The *magnitude* is the actual residual value directly encoded. Negative residuals are encoded as the ones complement value of the absolute residual so that negative residuals always start with a zero bit.

Example 3.12 *Assume* $p = 190, p_1 = 184, p_2 = 176$ *and* $P = \frac{p_1+p_2}{2} = 180$. *In this case,* $R = -10$, *which is encoded as the event (4,0101). If the Huffman code for 4 is 001, then this give the final codeword 0010101 which is seven bits long. To directly encode the value of p would require sixteen bits. When the decoder receives this codeword, it first calculates the prediction value (180), then parses the Huffman code, which allows decoding of the magnitude (0101). When it detects a leading zero in the magnitude, it knows the value must be negative, so the next four bits are decoded as -10. Finally, the decoded value is added to the prediction, which can be formed by the decoder, to give the reconstructed value for p.*

[8]Using a single static Huffman code for the residual would require a very large set of codewords, making a practical implementation difficult.

Chapter 4

Fourier analysis of Signals

4.1 Introduction

In this chapter, in preparation for a treatment of transform-based coding of signals, start by taking a thoughtful look at what we mean by a signal and how there are different mathematical representations available to us to capture the physical variations that define a signal. In particular, in our explanations we are interested in crossing over and back between "discrete" and "continuous" representations of signals, between "digitized" signals and "analogue" signals.

We then present the two steps needed to bring the mathematical methods invented by Fourier for dealing with periodic functions to bear on the problem of representing and analysing general non-periodic signals. The first step is the conversion of non-periodic data to periodic data. The second step is to develop a suitable way of comparing different periodic functions. With this background we proceed to look at comparing sine and cosine functions and show the development of ideas involved in the Fourier series expansion. Examining the Fourier series expansions for a number of related functions illustrates some general properties of the Fourier series of phase- and amplitude-shifted functions, of functions which exhibit odd, even and translational symmetry and of functions with and without discontinuities. This is our entry point to the world of sine series and cosine series expansions.

At this stage, most of our development will have been in terms of functions defined on a continuous finite domain interval, or equivalently, in terms of infinitely wide periodic functions whose domain is the set of real numbers. We then examine the particular case of functions which are just defined on a discrete lattice (or set of sample points) on a finite domain interval, or equivalently, sampled periodic functions. Having discovered what is meant by *orthogonal* functions in the context of sine and cosine functions which have integer number of periods over a finite domain interval (the basis functions of the Fourier series expansion) we proceed to examine an idea that is complementary to this — the idea of *completeness*. This is the notion of having sufficiently many orthogonal functions to exactly represent any given function. In the Fourier series expansion, we would need an infinite number of orthogonal sine and cosine basis functions to exactly represent any function defined on a continuous finite domain interval. With sampled periodic functions, it transpires that we only need as many orthogonal functions as there are samples in one period.

4.2 Background to Fourier Expansions

4.2.1 Digital and analogue signals

Most signals that we deal with are measurements of variations of some physical quantity over time (e.g. pressure variations for sound) or over space (e.g. intensity variations for monochrome images) as illustrated in Figure 4.1. When we digitize these signals, we usually do so using evenly-spaced sample points in the signal domain (time or space). We then know from the Nyquist sampling criterion, that as long as our sampling *rate* is greater than twice the highest frequency component in the signal, we will be able to capture and reproduce that signal faithfully. Otherwise we will get an *aliasing* effect where

samples of the highest frequencies present appear to come from very low frequency variations.[1]

Remember that we normally sample the physical quantities which constitute our signals at discrete instants in time or at discrete locations in space because that is what is currently convenient for our (usually electronic) measuring systems to do. Even at present, there are cases where it is more convenient to make measurements at discrete frequencies or of a range of moments[2] and reproduce the spatial or temporal signal from these. Future technologies may make such measurements the norm rather than the exception — the regular discrete spatial or temporal sampling is more of a historical accident than a necessary decision.[3] Nevertheless, if only for ease of visualisation, we usually take as our starting point a regularly sampled signal in the space or time domain, and consider other domains or other representations of this same signal to be transformed versions of this domain.

For the sake of definiteness, from now on we will consider our "signal" as a variation which extends over space and which is represented for us by samples of this signal taken at a regular array of discrete positions in space. With a slight change in terminology, everything we say carries over to signals that involve variation over time.

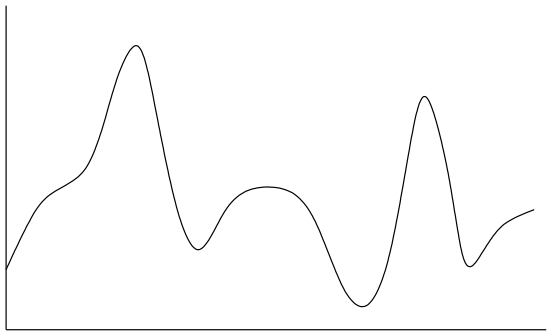


Figure 4.1: The graph of an analogue variation over space.

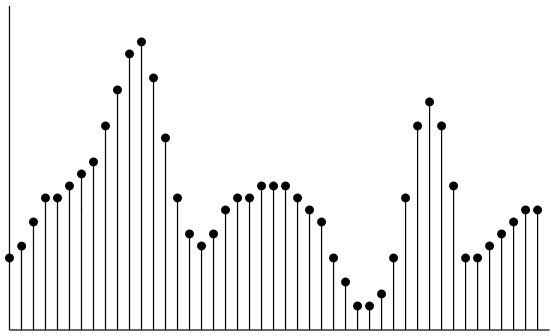


Figure 4.2: Illustration of a sampled and quantized version of the previous signal.

[1] Do not take the "requirements" of Nyquist sampling and evenly-spaced samples at face value. For example, the peripheral part of the human retina apparently has a much lower spatial sampling rate of cone receptors than the upper limit of spatial frequency of light variation than can be imaged on this part of the retina. Some researchers have argued that the cones are deliberately arranged in a pseudo-random spatial "array" to convert aliased frequencies to background "noise". See, for example, David R. Williams, "Seeing through the photoreceptor mosaic", *Trends in Neuroscience*, pp193-198, May 1986.

[2] "moments" is meant here in the related senses of mechanics (centre-of-gravity, moments of inertia) and statistics (average, variance, skewness, kurtosis) rather than in the sense of moments of time

[3] An example of non-spatial non-temporal measurements is the measurement of the spatio-temporal response of the human visual system. This is the sensitivity to various spatial patterns moving at various velocities across the visual field and might be implemented by a moving spatial pattern of neural connectivity called a spatio-temporal receptive field. It is not possible to take spatial or temporal samples of such receptive fields to find their shape. Nevertheless, their spatio-temporal shape has been determined by spatial-frequency and temporal-frequency measurements which are then transformed back to the spatio-temporal domain

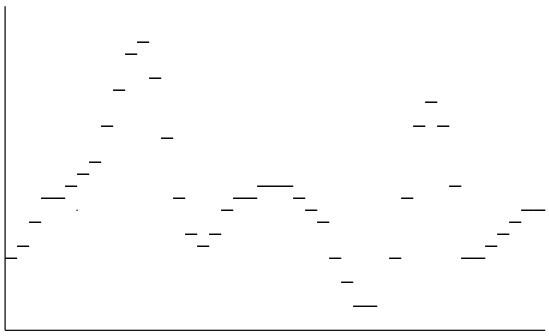


Figure 4.3: Another representation of a sampled and quantized signal.

4.2.2 Constructing Periodicity

For most of the purposes for which we might want to transform sampled signal data[4] to another domain, it makes sense to consider just a small collection of the samples at a time, say eight, sixteen, thirty-two or sixty-four contiguous samples. Such a collection of samples is shown in Figure 4.4. At this point many

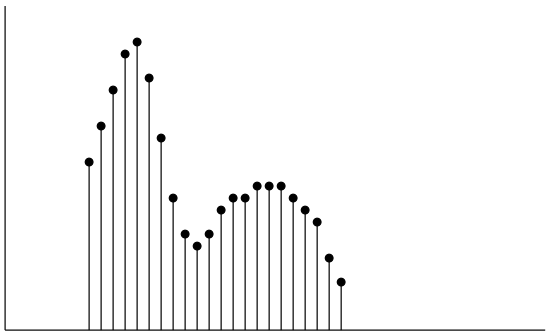


Figure 4.4: A contiguous set (or block) of samples from a signal.

authors will start to discuss orthogonal matrices that transform this collection of data values to other domains. However, in many ways, such an approach glosses over the power of the ideas that underlie such transforms, and is not easily extended to functions of a continuous variable.

A much more powerful approach is to realise that we can consider this collection of samples as comprising one period of a periodic function[5] that extends off infinitely in both directions, as shown in Figure 4.5.

This is important because a theorem by the French engineer, *Fourier*, tells us that any *periodic* function can be represented as a *linear combination* of sine and cosine functions whose frequencies are integer multiples of the basic or fundamental frequency of the periodic function. All we need to know are the values of the weights (called Fourier coefficients) corresponding to each of the sine and cosine functions in the linear combination and we can reproduce the shape of the original periodic function. This is true for a periodic function which is continuous over each period, and in fact even holds for a function which has a finite number of finite discontinuities (or jumps) in each period.

[4] A single measurement, represented by a single number, is referred to as a *datum* (plural *data*). It may seem that when we are making a measurement we are getting information, in the form of a single datum, but what we are actually doing is throwing away the answers to all sorts of other questions about a signal or measurements of a signal that we could make and reducing all of these possibilities to one single number: we are *throwing away* or discarding information, when we make a measurement. In effect, a measurement is making a *decision*, that of all the possibilities represented in the original physical quantity under consideration, the particular situation represented by our single datum *is* the case, whether or not this is exactly true, or even the whole story.

[5] A function is said to be *periodic* if its values repeat at exactly regular intervals of the independent variable. In other words $f(t+T) = f(t)$ for all values of the independent variable t and for one particular value of the period T

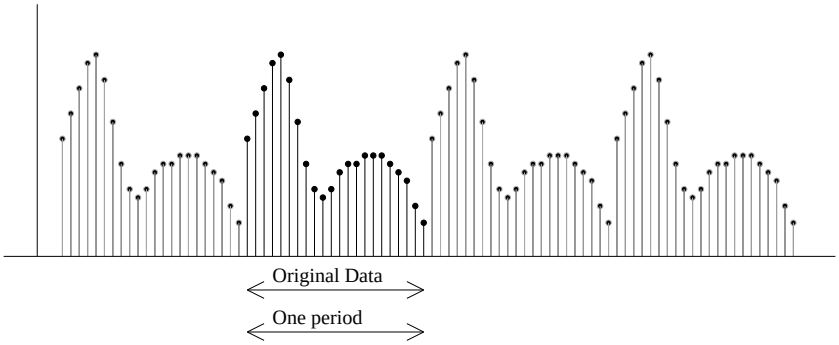


Figure 4.5: A block of samples from a non-periodic signal considered as one period of a periodic signal.

Consider a periodic function whose domain is the real number line and which is defined at every point on this real number line. In other words, for every real value x, the function has a corresponding value $f(x)$ defined. We say that such a mathematical function depends on, or is a function of the *continuous variable* x. There are clearly an infinite number of domain values in any interval of the real number line representing one period of such a function. If we wanted to specify an arbitrary periodic function, we would need to describe how our function maps each of these infinite number of domain values to their corresponding range values. In principle, then, when we are specifying a particular periodic function by specifying the map from domain to range values over one period, we are implicitly specifying an infinite number of ordered pairs, or an infinite number of maps of individual domain elements to individual range elements.[6]

Because in mathematics we rarely get something for nothing, when we change (or transform) to a description of a function of a continuous variable in terms of linear combinations of sine and cosine functions of different frequencies, we will still need an infinite number of Fourier coefficients to *exactly* describe the original function. The advantage of course is that a relatively small collection of the Fourier coefficients will nevertheless give us a function of a continuous variable that relatively closely approximates the original function. This is in contrast to the case where we have a description of the function in terms of its values at an infinite number of points in its domain. If we were to try to represent our function by its values at a small collection of domain positions, we might only poorly represent the original function, and would never be able to approximate it as a function of a continuous variable.

Suppose now we consider a function which is defined at only a finite number of domain values in any period. This could have been the original form for this function, or it might be the result of a sampling process on a function of a continuous variable. (We will assume that our samples are regularly spaced unless we say otherwise). We find that Fourier's theorem can also be extended to this case, except that now the sine and cosine functions are also sampled at the same sample points. We also find a nice symmetry between the representation in terms of particular domain values (hereafter referred to as the spatial representation) and the representation in terms of the values of Fourier coefficients. If, say, there are sixteen spatial samples of our function in each period of the function, then precisely sixteen Fourier coefficients will be required to exactly represent this sampled periodic function. These will consist of the weights for each of eight sampled sine functions whose frequencies are integer multiples of the fundamental frequency and ditto for eight cosine functions.

We mention above that the powerful insight in analysing or transforming a small part of the set of samples of an arbitrary (and generally non-periodic) sampled function is to consider the small number of samples on hand as being samples from one period of a periodic function. The periodic function constructed in this way bears no relationship of the remaining values in the original sampled function, outside the small number of samples under consideration, (compare for example Figure 4.2 with Figure 4.5) but it does help us to easily analyse and transform the particular samples on hand. Having found a way to analyse this particular subset of samples from the original set of samples, we can now go on to

[6] This in fact is the power of the concept of a function of a continuous variable: when we specify a function mathematically in the form $f : \Re \rightarrow \Re, x \mapsto x^2 + 2x + 1$ we are actually specifying an infinite number of individual element relationships, without needing to enumerate them individually.

apply the techniques described to the next subset of samples, and the next, and so on.

Now that we have made the jump from a small finite contiguous subset of samples (also known as a "block" of data) to a notional periodic function consisting of an infinite repetition of these samples, there are other ways that we could construct a periodic sampled function from the particular set of samples of the original function on hand. Suppose, for definiteness, we select a block of sixteen samples from our original non-periodic function. We could, for example, construct a periodic function with 32 samples, by duplicating the original samples in reverse and placing them immediately after the original samples in a single period of the constructed notional periodic function. This gives us a periodic function which is an *even* function over any single period (considering the origin to be at the centre of a period). Alternatively, we could construct a periodic function with 32 samples by changing the sign of the reversed duplicated samples. This gives us a periodic function which is an *odd* function over a single period. These two cases are illustrated in Figure 4.6 and Figure 4.7 below.

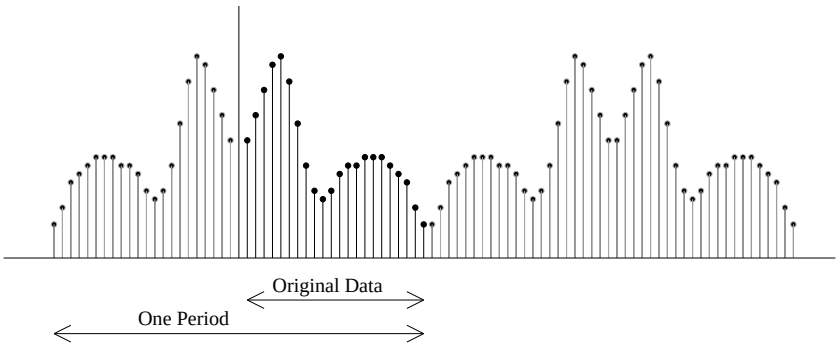


Figure 4.6: An *even* periodic function constructed from a finite block of samples from a non-periodic signal.

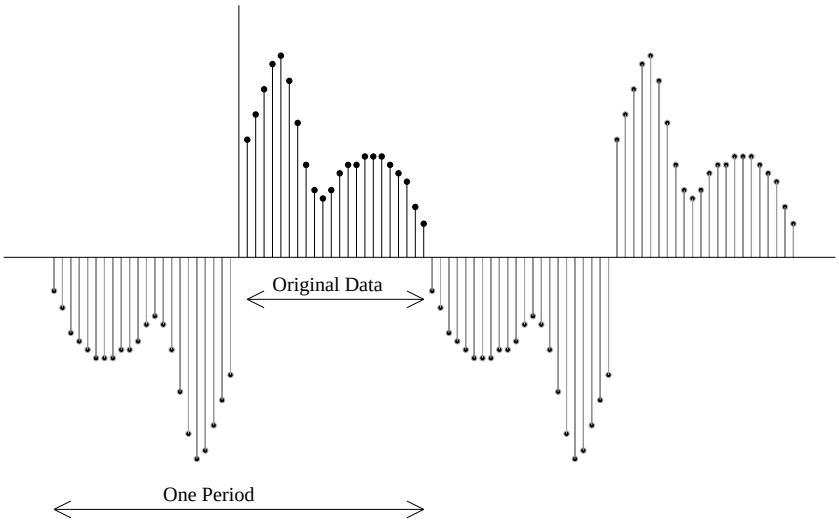


Figure 4.7: An *odd* periodic function constructed from a finite block of samples from a non-periodic signal.

Both of these constructions seem to be somewhat arbitrary, but by examining the properties of Fourier series we will see that there is intrinsic value in constructing odd or even periodic functions from the original data.

Before exploring further the different ways that we can transform the data, we need to get some understanding of how to compare functions, and the results that we get when we subsequently compare

different sine and cosine functions. This will in turn lead us to discover other collections of functions (called orthogonal functions) that have properties similar to the sine and cosines.

4.2.3 Comparing functions

How do we compare functions, in a formal and consistent way so that we can derive something useful from the comparison? Before answering, it is instructive to ask how we compare vectors. Consider, for example, a vector in three dimensions represented by coefficients along three perpendicular axes. Figure 4.8 shows such a vector. In addition it shows an alternative representation of the three components of the vector in a way which is reminiscent of the values of a sampled function.

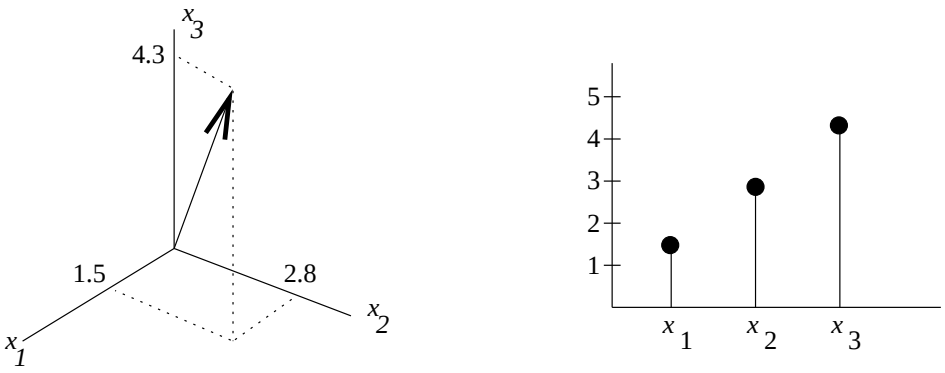


Figure 4.8: A Cartesian geometric representation and an array representation of the same 3-D vector.

We can tell if a vector is in the same direction as another vector, or is perpendicular to it, by evaluating the *dot product* of the two vectors. Even if the two vectors are neither exactly parallel nor exactly perpendicular, we can still use the dot product to cleave a vector into the part which is perpendicular and the part which is parallel to the other.[7]

The process involved in the dot product that gives us a useful way of comparing vectors is one of multiplying corresponding values and summing the result. Consider two vectors $\mathbf{a} = (a_1, a_2, a_3)$ and $\mathbf{b} = (b_1, b_2, b_3)$ Their dot product is written as

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 = \sum a_i b_i$$

If corresponding values are always positive or always negative, we see that when multiplied, they always make a positive contribution to the total. Thus we see that the more alike the components of two vectors being compared are, (at least in the sense of both being positive or both being negative), the greater they contribute to the overall dot-product total. In this way a larger total (or dot product value) is an indication of similarity. Equally, if we consider components of two vectors which are such that if a component of one is positive, the corresponding component of the other is negative, and vice versa, then these pairs always contribute negatively to the overall total and reduce it. We can see that this lowering of the overall total or dot product value is an indication of the corresponding components not being "similar" in some sense. A zero value for a dot product (assuming that not all the components of the vectors being compared are zero) is an indication that there are equal amounts of similarity and dissimilarity in the comparison of the two vectors and we say that they are *orthogonal*. (If there was nothing but dissimilarity, we would say that the vectors were opposites, or pointed in opposite directions). The geometric ideas and the corresponding components for three unit vectors in 2-D are shown in Figure 4.9.

Now we make the jump back to functions. Consider the components of our three-dimensional vectors *as if* they were the values of a function sampled at just three particular points as shown on the right-hand side of Figure 4.8. We still have the dot-product concepts of similarity, orthogonality and oppositions to apply to these three sampled values. However, now, we can extend these ideas to the underlying function that is represented by these three samples.

[7]For convenience, this argument implicitly assumes that we are dealing with unit vectors.

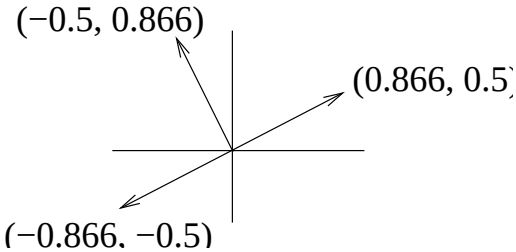


Figure 4.9: A vector unit vector with components (0.866,0.5), together with a unit vector in the opposite direction and a perpendicular unit vector. Note the relationships between the components of each.

We could consider our function defined on a continuous variable to be sampled at, say, 100 points. We can still do a comparison of two functions by multiplying all 100 corresponding sampled values from each function and summing. In spite of the seeming growth of our problem to 100-dimensions (which is impossible to visualize) we can still apply the original dot product ideas of similarity, orthogonality and oppositions. Now take this process all the way to every point on the continuous domain on which each of our functions being compared is defined. (Mathematicians would say that we are taking a *limit* as our number of samples becomes infinite). If we multiply corresponding values of the functions being compared and add the result, just as we did in the dot product, the total we get tells us about the similarity or otherwise of the corresponding functions.

Now, whether or not you have realised it, the process just described for functions defined on a continuous domain is one of integrating the product of two functions over an interval. So now we have our tool for comparing functions.

Another more general mathematical term for the dot product is the *inner* product. The inner product of two functions $f(x)$ and $g(x)$ defined over the interval $[p, q]$ in the Reals is

$$< f, g >= \int_p^q f(x)g(x)dx$$

4.3 Fourier Analysis of Periodic Functions

4.3.1 Comparing sines and cosines

Consider a collection of sine and cosine functions defined over an interval which all have the property that they have an integer number of periods over that interval. This is only a small subset of all the sine and cosine functions that we could define over such an interval, most of which do not have an exactly integer number of periods over the defined interval, so it is not representative of all sine and cosine functions. It is, nevertheless, a very interesting collection of functions with interesting properties, including the fact that there is an infinite number of functions in the set. Some of these sine and cosine functions with an integer number of periods in the given interval are shown in the first six lines of Figure 4.10. Examples of some perfectly valid sine and cosine functions which do not satisfy the integer-period requirement are shown in the last two lines of Figure 4.10.

It is fairly straightforward to show that every one of the integer-period functions is *orthogonal* to every other one, in the sense that like the dot product of two perpendicular vectors, their inner products are zero: [8]

$$< f, g >= 0$$

In the case of periodic functions, with period L, we don't lose generality by considering the interval over which to compare our functions, and over which the inner product is defined, as the interval $[0, L]$. The period L and fundamental angular frequency ω are related by $L = 2\pi/\omega$. Then the above statement about orthogonality is embodied in the following relationships:

[8] In 2-D, given two perpendicular vectors, it is not possible to find a third which is perpendicular to both. Similarly in 3-D, given three perpendicular vectors, it is not possible to find a fourth which is perpendicular to *all* of the three. In a direct extrapolation of these ideas, if we have an infinite number of sine and cosine functions that are all "perpendicular" to each other, then these functions must be elements of an infinite-dimensional vector space.

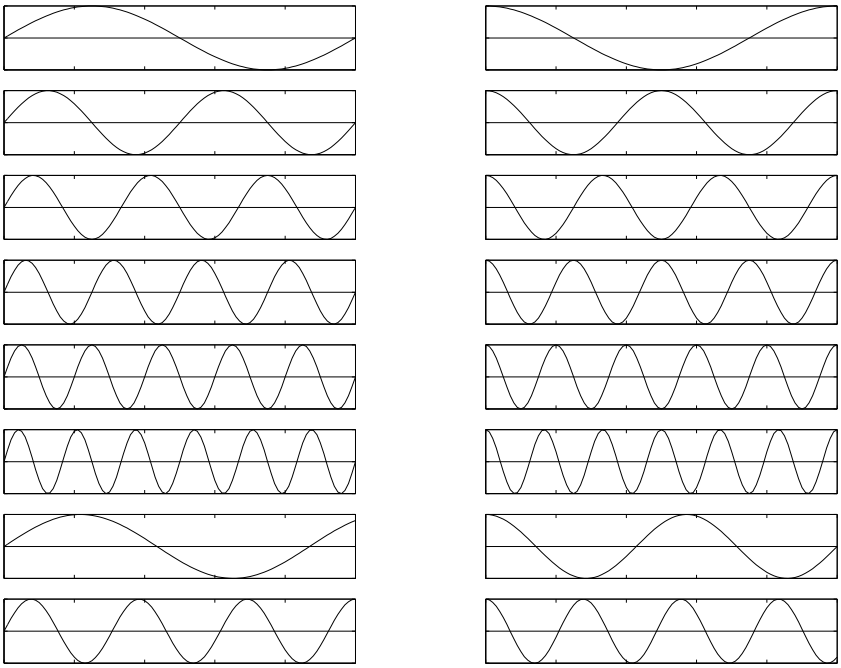


Figure 4.10: The first six rows show an illustration of some sine functions (LHS) and some cosine functions (RHS) which have an integer number of periods over the domain interval on which they are defined. The last two rows show some sine and cosine functions defined over the same interval which do not satisfy the integer-period requirement.

$$\int_0^L \sin n\omega x \cos m\omega x dx = 0,$$

for all integers n and m

$$\int_0^L \sin n\omega x \sin m\omega x dx = 0,$$

for all integers $n \neq m$

$$\int_0^L \cos n\omega x \cos m\omega x dx = 0,$$

for all integers $n \neq m$

$$\int_0^L \cos n\omega x \cos n\omega x dx = \frac{L}{2},$$

for all integers $n \neq 0$

$$\int_0^L \sin n\omega x \sin n\omega x dx = \frac{L}{2},$$

for all integers $n \neq 0$. The orthogonality of sine and cosine functions becomes important when we try to calculate the coefficients of the linear Fourier expansion.

4.3.2 Fourier's Theorem

Consider the one dimensional wave equation

$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 f}{\partial t^2}$$

defined over the finite 1-D interval $[0, L]$. We know that this has solutions of the form $f(x \pm ct)$. Yet it is also possible using the technique of separation of variables to show that the most general solutions of this wave equation are of the form

$$\sum_{n=0}^{\infty} A_n \sin\left(\frac{n\pi x}{L}\right)\left\{C_n \sin\frac{n\pi ct}{L} + D_n \cos\frac{n\pi ct}{L}\right\}$$

When Fourier noticed this dual form of the solution of the PDE, he made the deduction that the relatively arbitrary function $f(x)$ of the first solution type could be expressed as a (possibly infinite) linear combination of the sines and cosines of the separation-of-variables solution type. If fact, the precise statement of Fourier's deduction is that a periodic function $f(x)$ (or a function defined over a finite interval, say $[0, L]$) which has a finite number of finite-size jump discontinuities over the interval of definition) can be expanded in the form:

$$\begin{aligned} f(x) &= A_0 + \sum_{n=1}^{\infty} A_n \sin(n\omega x + \phi_n) \\ &= \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega x + \sum_{n=1}^{\infty} b_n \sin n\omega x \end{aligned}$$

where we have used standard trig relationships and the identifications

$$a_n = A_n \sin\phi_n$$

$$b_n = A_n \cos\phi_n$$

$$\phi_n = \tan^{-1}\left(\frac{a_n}{b_n}\right)$$

$$A_n = \sqrt{a_n^2 + b_n^2}$$

to convert between these two versions of the expansion, and where the a_n and b_n terms are called *Fourier Coefficients*.

Our next task is to establish the values of the Fourier Coefficients corresponding to a particular function. Suppose we integrate both sides of the relationship

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega x + \sum_{n=1}^{\infty} b_n \sin n\omega x$$

over the interval $[0, L]$. Each term on the RHS will involve evaluating the area under the graph of a sine or cosine function with a whole number of cycles and will thus evaluate to zero. The single exception is the a_0 term. We thus have

$$\int_0^L f(x)dx = \int_0^L \frac{1}{2}a_0 dx$$

or

$$a_0 = \frac{2}{L}\int_0^L f(x)dx$$

The term $a_0/2$ is thus the mean value of the function over one period and is often referred to as the *dc level* or *dc coefficient*.

Multiplying both sides of the original Fourier expansion by $\cos\omega x$ and integrating over the interval $[0, L]$ we find that the only non-zero term on the RHS is one involving $\cos^2\omega x$. That is,

$$\int_0^L f(x)\cos\omega x dx = \int_0^L a_1 \cos^2\omega x dx = \frac{1}{2}a_1 L$$

and we get

$$a_1 = \frac{2}{L}\int_0^L f(x)\cos\omega x dx$$

Continuing in this vein, multiplying both sides of the Fourier expansion by $\cos n\omega x$ (for different values of n) and integrating, we can derive the relationships

$$a_n = \frac{2}{L}\int_0^L f(x)\cos n\omega x dx$$

Similarly, multiplying by $\sin n\omega x$ and integrating allows us to derive the relationships

$$b_n = \frac{2}{L}\int_0^L f(x)\sin n\omega x dx$$

There is also another form of the Fourier series expansion that we need later. This is the form written in terms of complex exponentials. Using Euler's formula

$$e^{jn\omega x} = \cos(n\omega x) + j\sin(n\omega x)$$

and defining the complex coefficients

$$c_0 = \frac{a_0}{2}$$

$$c_n = \frac{a_n - jb_n}{2}$$

$$c_{-n} = \frac{a_n + jb_n}{2}$$

we get

$$f(x) = c_0 + \sum_{n=1}^{\infty}(c_n e^{jn\omega x} + c_{-n}e^{-jn\omega x})$$

or equivalently

$$f(x) = \sum_{n=-\infty}^{+\infty}(c_n e^{jn\omega x})$$

The complex form of the Fourier coefficients can be derived from the real a_n and b_n coefficients, again using Euler's formula as

$$c_n = \frac{a_n - jb_n}{2} = \frac{1}{L}\int_0^L f(x)\, e^{-jn\omega x}\, dx$$

4.3.3 Fourier series expansions for particular functions

There are some important properties of the Fourier series expansion that we need below and they are best illustrated by examining the Fourier series expansions for some particular functions. We list these results for the functions concerned without explicitly deriving the results here. We need nothing more than standard integration methods (particularly integration by parts) to derive the results, but the integrals are often quite involved for all except the simplest functions.

(a) A positively-biased rectangular waveform, period T

$$y = \begin{cases} A & \text{for } 0 \le x < L/2 \\ 0 & \text{for } L/2 \le x < L \end{cases}$$

Fourier Expansion:

$$a_0 = A, a_n = 0 \text{ for } n \neq 0$$

$$b_n = 0 \text{ for } n \text{ even}, b_n = \frac{2A}{n\pi} \text{ for } n \text{ odd}$$

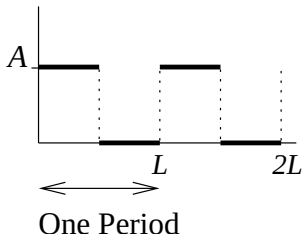


Figure 4.11: The spatial representation of a positively-biased rectangular waveform.

$$y(x) = \frac{A}{2} + \frac{2}{\pi}\left(\sin\omega x + \frac{1}{3}\sin h3\omega x + \frac{1}{5}\sin 5\omega x + \cdots\right)$$

(b) A zero-average rectangular waveform, period L

$$y = \begin{cases} A & \text{for } 0 \le x < L/2 \\ -A & \text{for } L/2 \le x < L \end{cases}$$

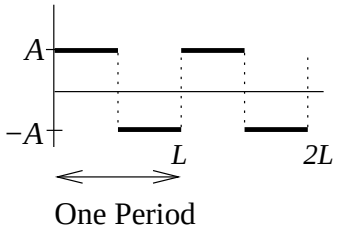


Figure 4.12: The spatial representation of a zero average rectangular waveform.

Fourier Expansion:

$$a_0 = 0, a_n = 0 \text{ for } n \neq 0$$

$$b_n = 0 \text{ for } n \text{ even}, b_n = \frac{4A}{n\pi} \text{ for } n \text{ odd}$$

$$y(x) = \frac{4A}{\pi}\left(\sin\omega x + \frac{1}{3}\sin 3\omega x + \frac{1}{5}\sin 5\omega x + \cdots\right)$$

(c) A zero average rectangular waveform, period L, with 1/4 cycle phase shift

$$y = \begin{cases} A & \text{for } 0 \le x < L/4 \\ -A & \text{for } L/4 \le x < 3L/4 \\ A & \text{for } 3L/4 \le x < L \end{cases}$$

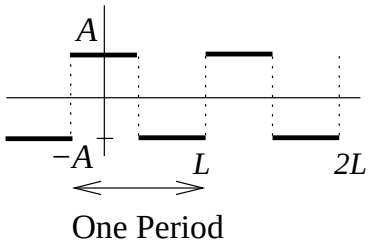


Figure 4.13: The spatial representation of a 1/4-cycle-phase-shifted, zero-average rectangular waveform.

Fourier Expansion:

$$a_0 = 0, a_n = \frac{4A}{\pi}[\ 1 \quad 0 \quad -\tfrac{1}{3} \quad 0 \quad \tfrac{1}{5} \quad \cdots\ ]$$

$$b_n = 0 \text{ for all } n$$

$$y(x) = \frac{4A}{\pi}\left(\cos\omega x - \frac{1}{3}\cos 3\omega x + \frac{1}{5}\cos 5\omega x + \cdots\right)$$

(d) A sawtooth waveform, period L

$$y = A\frac{x}{L} \text{ for } 0 \le x < L.$$

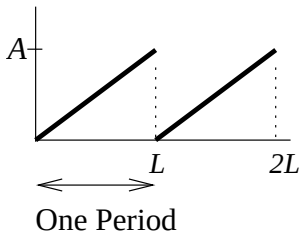


Figure 4.14: The spatial representation of a sawtooth waveform.

Fourier Expansion:

$$a_0 = A, a_n = 0 \text{ for } n \neq 0$$

$$b_n = -\frac{A}{n\pi}$$

$$y(x) = \frac{A}{2} + \frac{A}{\pi}\left(-\sin\omega x - \frac{1}{2}\sin 2\omega x - \frac{1}{3}\sin 3\omega x - \cdots\right)$$

(e) A triangular waveform, period L, positive bias

$$y = \begin{cases} \frac{2Ax}{L} & \text{for } 0 \le x < L/2 \\ 2A(1 - \frac{x}{L}) & \text{for } L/2 \le x < L \end{cases}$$

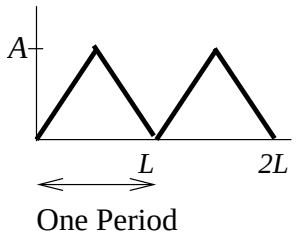


Figure 4.15: The spatial representation of a triangular waveform with positive bias.

(f) A triangular waveform, period L, with zero bias and odd symmetry

$$y = \begin{cases} 4A\frac{x}{L} & \text{for } 0 \le x < L/4 \\ 2A(1 - \frac{2x}{L}) & \text{for } L/4 \le x < 3L/4 \\ 4A(\frac{x}{L} - 1) & \text{for } 3L/4 \le x < L \end{cases}$$

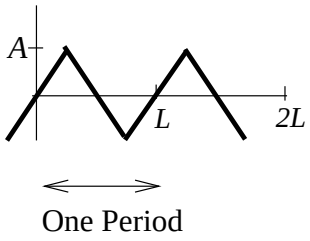


Figure 4.16: The spatial representation of a triangular waveform with zero bias and odd symmetry.

4.3.4 General properties of Fourier series

It is possible to draw some general conclusions based on the Fourier series of the given functions.

(a) Spatial Shift of Waveform

A phase (or spatial) shift of the original function does not directly affect the Fourier coefficients, just the phase of the sine or cosine basis functions. The Fourier series of the waveform in Figure 4.23 is

$$y(x) = \frac{4A}{\pi}\left(\sin\omega x + \frac{1}{3}\sin 3\omega x + \frac{1}{5}\sin 5\omega x + \cdots\right)$$

The Fourier series of the waveform in Figure 4.24 is

$$y(x) = \frac{4A}{\pi}\left(\cos\omega x - \frac{1}{3}\cos 3\omega x + \frac{1}{5}\cos 5\omega x + \cdots\right)$$

which is equivalent to

$$y(x) = \frac{4A}{\pi}\left(\sin\omega(x+\frac{\pi}{2}) + \frac{1}{3}\sin 3\omega(x+\frac{\pi}{2}) + \frac{1}{5}\sin 5\omega(x+\frac{\pi}{2}) + \cdots\right)$$

Clearly a phase shift can affect the Fourier coefficients indirectly, as in this case it has converted them from sine coefficients (b_n's) to cosine coefficients (a_n's).

(b) Amplitude Shift of Waveform

An amplitude shift (or bias) of a waveform has no effect on any of the Fourier coefficients, except the average value a_0, to which the amplitude shift is simply added. As before, the Fourier series of the waveform in Figure 4.25 is

$$y(x) = \frac{4A}{\pi}\left(\sin\omega x + \frac{1}{3}\sin 3\omega x + \frac{1}{5}\sin 5\omega x + \cdots\right)$$

while that of the waveform in Figure 4.26 is

$$y(x) = A + \frac{4A}{\pi}\left(\sin\omega x + \frac{1}{3}\sin 3\omega x + \frac{1}{5}\sin 5\omega x + \cdots\right)$$

(c) Waveform Even/Odd Symmetry

- Even Symmetry: $f(-x) = f(x)$ (a mirror image through the y-axis)
- Odd Symmetry: $f(-x) = -f(x)$ (symmetric about the origin)

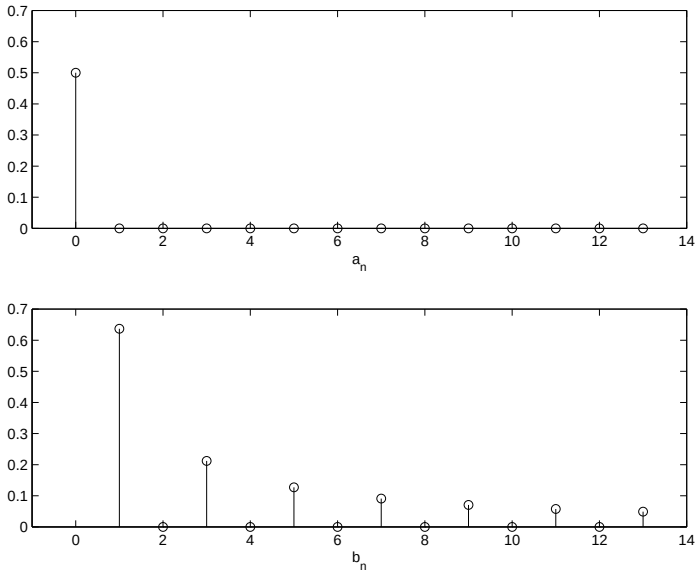


Figure 4.17: Values for the first few coefficients for the above positively-biased rectangular waveform with $A = 1$.

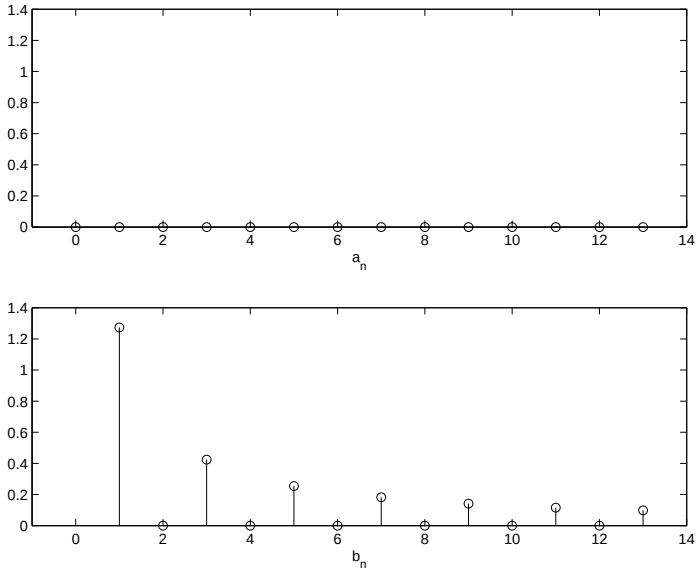


Figure 4.18: Values for the first few coefficients for the above zero-average rectangular waveform with $A = 1$.

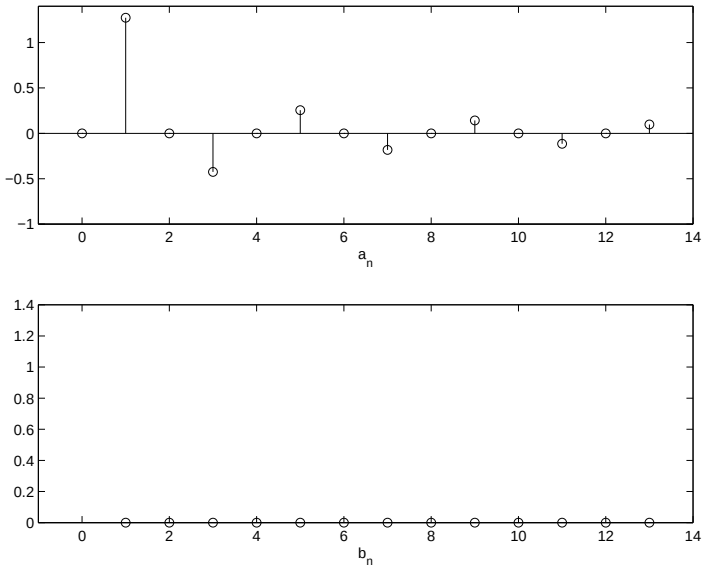


Figure 4.19: Values for the first few coefficients for the zero-average 1/4-cycle-phase-shifted rectangular waveform with $A = 1$.

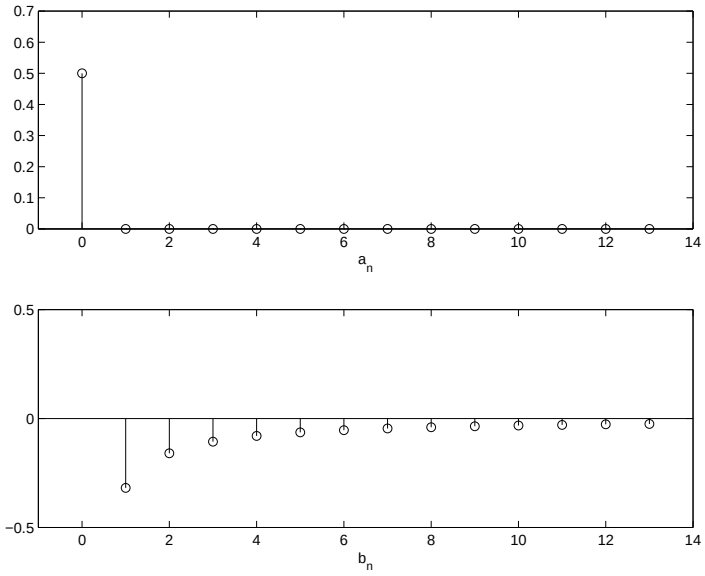


Figure 4.20: Values for the first few coefficients for the sawtooth waveform with $A = 1$.

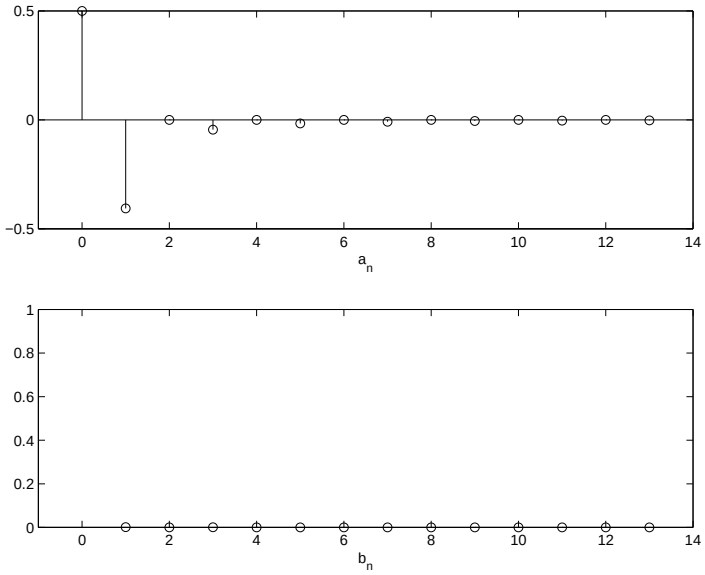


Figure 4.21: Values for the first few coefficients for the triangle waveform with $A = 1$ and positive bias.

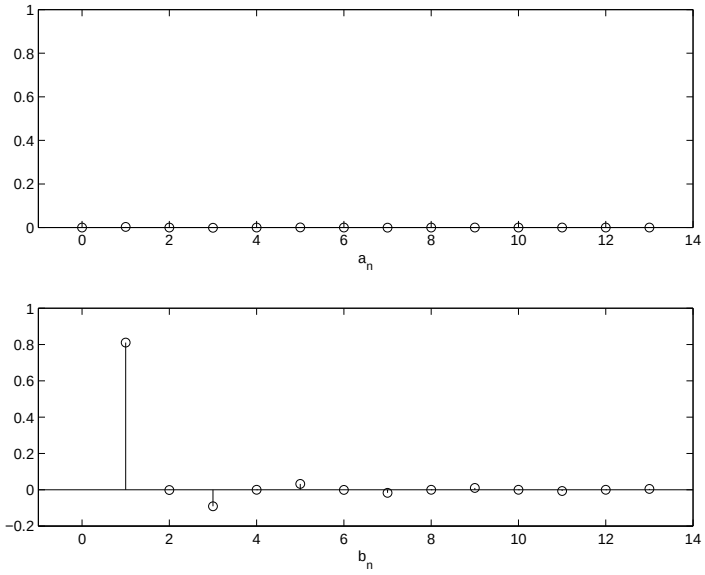


Figure 4.22: Values for the first few coefficients for the triangle waveform with $A = 1$, zero bias and odd symmetry.

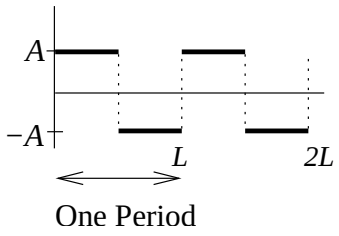


Figure 4.23:

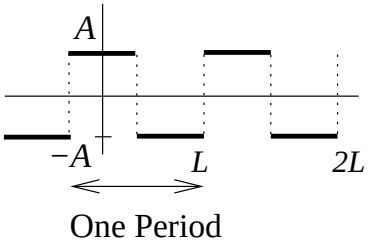


Figure 4.24:

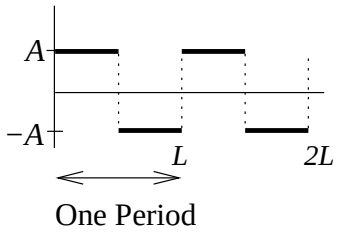


Figure 4.25: A zero-mean waveform.

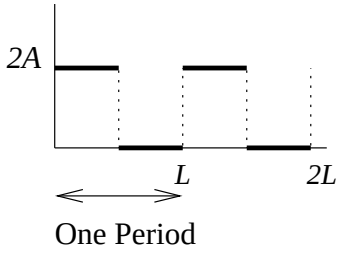


Figure 4.26: A positively-biased waveform.

It can be shown that any function either is an even function or an odd function or can be written as the sum of two functions, one even, one odd. The generality of this fact makes it quite important, so any properties attaching to even symmetry or odd symmetry are likely to be significant. We see above that the Fourier series expansion is based on the idea of comparing a function defined on a finite domain interval (or equivalently a periodic function) with integer-period sine and cosine functions defined over the same interval. Remember, the comparison involves multiplying the values of the functions over the domain interval and then integrating over this interval. In addition, it is straightforward to show that multiplication of odd/even functions has some fairly predictable properties:

1. The product of two even functions is an even function
2. The product of two odd functions is an even function
3. The product of an odd and an even function is an odd function

We can, however, go a step further to point out that symmetric integral about the origin of an odd function is always zero, i.e.

$$\int_{-L/2}^{L/2} f(x)dx = 0, \text{ for } f(x) \text{ odd.}$$

Now, sine functions quite definitely possess odd symmetry about the origin, while cosine functions are just as definitely even functions. Consequently, we can use this property of symmetry about the origin as a way of predicting properties of the Fourier series of a function. For convenience, consider the interval of definition of our functions to be $[-L/2, L/2]$:

1. $a_0/2$ is the *mean* value of a function and it is evaluated by a symmetric integral about the origin. For an odd function this integral is zero, so the mean value is zero, so consequently a_0 is zero. For an even function we get

$$a_0 = \frac{2}{L}\int_{-L/2}^{L/2} f(x)\,dx = \frac{4}{L}\int_{0}^{L/2} f(x)\,dx$$

   (Note the limits of integration!)

2. The a_n coefficients involve integrals of the form

$$a_n = \frac{2}{L}\int_{0}^{L} f(x)\cos n\omega x\,dx = \frac{2}{L}\int_{-L/2}^{L/2} f(x)\cos n\omega x\,dx$$

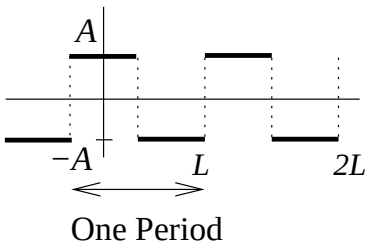


Figure 4.27: An *even* waveform.

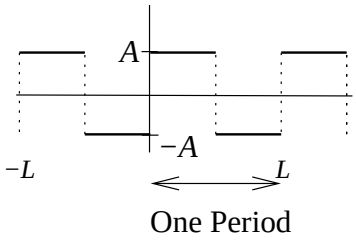


Figure 4.28: An *odd* waveform.

If $f(x)$ is odd, as $\cos n\omega x$ is even, the product is odd and $a_n = 0$ for all n. If $f(x)$ is even, as $\cos n\omega x$ is even, the product is even and a_n is given by

$$a_n = \frac{4}{L}\int_0^{L/2} f(x)\cos n\omega x\, dx$$

(Note the limits of integration!)

3. The b_n coefficients involve integrals of the form

$$b_n = \frac{2}{L}\int_0^{L} f(x)\sin n\omega x\, dx = \frac{2}{L}\int_{-L/2}^{L/2} f(x)\sin n\omega x\, dx$$

If $f(x)$ is even, as $\sin n\omega x$ is odd, the product is odd and $b_n = 0$ for all n. If $f(x)$ is odd, as $\sin n\omega x$ is odd, the product is even and b_n is given by

$$b_n = \frac{4}{L}\int_0^{L/2} f(x)\sin n\omega x\, dx$$

(Note the limits of integration!)

In summary, if $f(x)$ is an even function, then a_0 is non-zero and the corresponding Fourier series has only cosine terms. On the other hand, if $f(x)$ is odd, $a_0 = 0$ and the corresponding Fourier series only has sine terms.

Note that this actually gives us a method, albeit it rather involved, of fractioning a function into the sum of two parts, one even (the sum of a_0 and the other a_n terms of its Fourier series) and one odd (the sum of the b_n terms of its Fourier series).

Now, recall our construction of periodic functions earlier by a repetition of a block of data samples to give a period which was twice as long as the block of data, but where the periodic function was even or odd. Now we can see that the effect of this is to construct Fourier series expansions of these "doubled-up" functions which consist of cosine terms only or alternatively, of sine terms only. We return to this topic in more detail below.

(d) Waveform Translational Symmetry

We can regroup the terms of the Fourier expansion so that terms with the same spatial frequency or period are grouped together, as follows.

$$\begin{aligned} f(x) &= \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega x + \sum_{n=1}^{\infty} b_n \sin n\omega x \\ &= \frac{1}{2}a_0 + \sum_{n=1}^{\infty} (a_n \cos n\omega x + b_n \sin n\omega x) \\ &= \frac{1}{2}a_0 + (a_1 \cos \omega x + b_1 \sin \omega x) + (a_2 \cos 2\omega x + b_2 \sin 2\omega x) + \cdots \end{aligned}$$

The term $(a_1 \cos \omega x + b_1 \sin \omega x)$ is known as the *fundamental*.
The term $(a_2 \cos 2\omega x + b_2 \sin 2\omega x)$ is known as the *second harmonic*.
The term $(a_3 \cos 3\omega x + b_3 \sin 3\omega x)$ is known as the *third harmonic*, and so on.

You may have noted that the Fourier expansions for all of the periodic functions considered above contained only *odd* harmonics, with the exception of that for the sawtooth waveform. By considering the integrals involved in computing the fundamental and each of the harmonics, it should be possible to show that for a periodic function $f(x)$ which has period L, the *even* harmonics are all zero if it has *translational anti-symmetry* over a distance of $L/2$, i.e. if

$$f(x + \frac{L}{2}) = -f(x).$$

On the other hand, it should be possible to show that for a periodic function $f(x)$ which has period L, the *odd* harmonics are all zero if it has *translational symmetry* over a distance of $L/2$, i.e. if

$$f(x + \frac{L}{2}) = f(x)$$

or in other words, if it actually has period $L/2$. We discuss below why we might want to pretend that a periodic function has a period L, when it actually has a period $L/2$.

Note that (even ignoring the positive bias which pushes the sawtooth upwards to have a mean value of $1/2$) the sawtooth waveform does not exhibit either of the above types of translational symmetry when translated by half of one period. Hence we should not be surprised to discover that it has both even and odd harmonics.

(e) Waveform discontinuities

One other feature that can be noted from the Fourier expansions for the periodic functions considered above, is that for increasing value of n, the coefficients get smaller at very different rates. In some cases the coefficients go to zero with increasing n as $1/n$ and in other cases as $1/n^2$. Observe that the faster convergence is associated with the periodic functions not having jump discontinuities within or at the ends of a period interval. While it is possible to show that in the limit as $n \to \infty$ the Fourier expansion converges to be an exact representation of the original function, if we stop a Fourier expansion at any finite value of n, a type of "ringing" error in the approximation is exhibited around any jump discontinuity of the original function. This type of error in the approximation is called a *Gibb's phenomenon* and is an indication that while convergence does eventually occur, it is invariably going to be slow.

Continuing this theme, one might be tempted to ask, if convergence is at least as fast as $1/n^2$ when there are no jump discontinuities in the original function, under what conditions would convergence be faster than $1/n^2$? The answer is simple and intuitive. Convergence will be faster than $1/n^2$ when there are no jump discontinuities in the *first derivative* of the original function — in other words, when there are no abrupt changes in slope of the original function. For the triangle waveform above we see that there *are* two abrupt changes in slope, one at the centre and one at the end of the period interval (as the period interval is defined above). Consequently, we do not expect the coefficients of the Fourier expansion of this function to converge more rapidly than $1/n^2$. However, the following function

$$y = \begin{cases} 2x^2 & \text{for } 0 \le x < 0.25 \\ -2x^2 + 2x - 0.25 & \text{for } 0.25 \le x < 0.75 \\ 2x^2 - 4x + 2 & \text{for } 0.75 \le x < 1 \end{cases}$$

has the following first derivative

$$y = \begin{cases} 4x & \text{for } 0 \le x < 0.25 \\ -4x + 2 & \text{for } 0.25 \le x < 0.75 \\ 4x - 4 & \text{for } 0.75 \le x < 1 \end{cases}$$

This is continuous, but has changes of slope at the changeover points of the branches. The original function has a second derivative given by

$$y = \begin{cases} 4 & \text{for } 0 \le x < 0.25 \\ -4 & \text{for } 0.25 \le x < 0.75 \\ 4 & \text{for } 0.75 \le x < 1 \end{cases}$$

which is discontinuous, with jump discontinuities at the changeover points. Consequently, we expect this function to have a Fourier expansion with coefficients which converge as $1/n^3$, but no faster.

We examine the usefulness of this and the earlier observations on the coefficients of Fourier expansions in the next section. Firstly we briefly return to wrap up one general aspect of the Fourier representation of periodic functions

4.3.5 Complete sets of functions

Returning to our earlier discussion on comparing functions, we see that the Fourier Coefficients are simply the results of *comparing* our target function $f(x)$ with sine and cosine functions defined over the interval $[0, L]$ that have an integer number of cycles in this interval. We have already seen that the sines and cosines with an integer number of cycles over our given interval form an orthogonal set. In a sense this means that when we compare a target function $f(x)$ with our collection of integer-period sines and cosines and determine the corresponding Fourier coefficient, each comparison captures something *unique* about the way that the function $f(x)$ changes that is not captured by any of the others. ¿From Fourier's theorem, we now have another fact, which is that the set of integer-cycle sines and cosines is in some sense *complete*. That is, between them they capture *every* way in which a function $f(x)$ can change as a function of x over the interval $[0, L]$.[9] What Fourier's theorem hints at, but doesn't tell us directly, is that *any* complete set of orthogonal functions can be used in this way.

It may not be clear how Fourier's theorem tells us that our set of integer-cycle sines and cosines capture *every* way in which a function $f(x)$ can change. The point is that Fourier established the fact that an *arbitrary* function $f(x)$ can be represented as a linear combination of the complete set of orthogonal integer-cycle sines and cosines. Speaking of "*every* way in which a function $f(x)$ can change as a function of x over the interval $[0, L]$" is just a paraphrase of this result.

4.4 Fourier analysis of non-periodic functions

There are basically two ways in which we can extend Fourier analysis to non-periodic functions. One is to consider what a periodic function would become as the length of a period becomes infinite. In this case we get a formulation which is suitable for dealing with non-periodic functions defined over the entire real line — this is the Fourier transform. The second is to consider a part of a non-periodic function defined over a finite interval as if it were one cycle of a periodic function and then to bring the machinery of the Fourier series expansion to bear. Developing this second case brings us to the discrete cosine transform

4.4.1 The Fourier Transform

Before pursuing our main objective of the discrete cosine transform and other orthogonal transforms, we present a brief outline of the construction of the Fourier transform.

It is possible to express the Fourier series expansion in a complex exponential form:

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{j\frac{2\pi n}{L}x}$$

where

$$c_n = \frac{1}{L}\int_{-L/2}^{L/2} f(x) e^{-j\frac{2\pi n}{L}x}\, dx$$

As $L \to \infty$ the second of these expressions tells us that $c_n \to 0$, which is not much use. However, the product Lc_n does not tend to zero as $L \to \infty$, so it is worth examining what we can understand about the integral in this expression. In the diagrams of Fourier series coefficients above it was convenient to use the Fourier coefficient index n to label the horizontal, or "frequency" axis. The spatial frequency of the n^{th} sine or cosine basis function could be calculated as $2\pi n/L$. It was also convenient at that time to use the symbol ω to represent the constant fundamental frequency of the target periodic function:

[9] Strictly speaking we should say every *reasonable* way in which a function can change as the Fourier result is only established for functions having a finite number of finite discontinuities over our interval. Nevertheless the theorem certainly covers any function likely to arise in an engineering context.

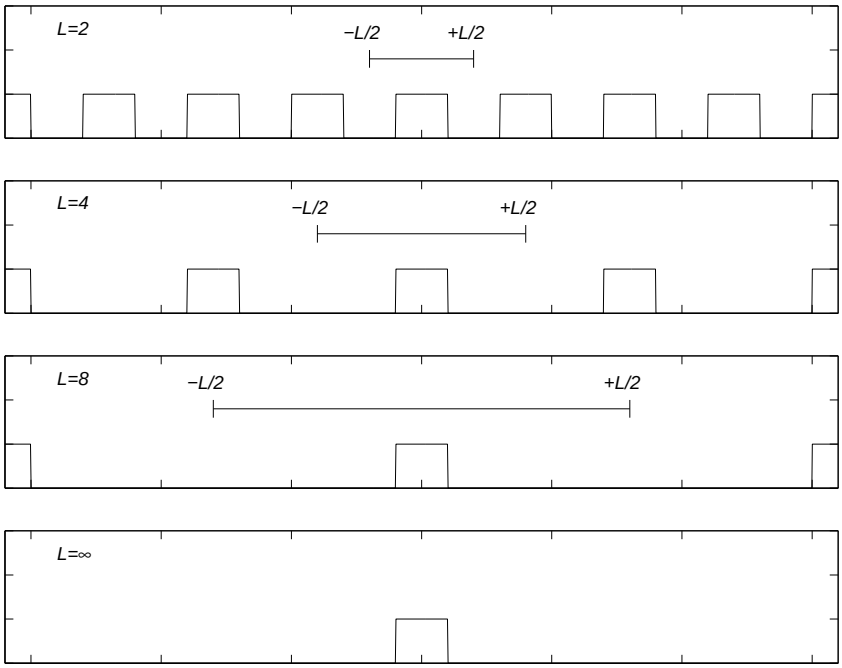


Figure 4.29: Illustration of the period L of a periodic function becoming increasingly large, while maintaining the basic pulse shape being represented.

$\omega = 2\pi/L$. Now we are considering the situation where the period L is becoming infinite, so the concept of a fundamental frequency is becoming redundant as $2\pi/L \to 0$. Consequently we introduce a proper variable to represent position on the horizontal or frequency axis when we are plotting Fourier coefficients:

$$\omega_n = \frac{2\pi n}{L}$$

Suppose, for example, this variable has the value 3, i.e. $\omega_n = 3$ radians m^{-1}. Then as L gets larger, we will have to go to a larger value of n to find the coefficient which corresponds to the spatial frequency with the value 3 radians m^{-1}. Our concept of spatial frequency has not changed, just the fact that a particular coefficient n corresponds to smaller and smaller spatial frequencies as the period L gets longer. Now we no longer have to consider the expression

$$Lc_n = \int_{-L/2}^{L/2} f(x) e^{-j\frac{2\pi n}{L}x}\, dx$$

as dependent on the Fourier coefficient number n, but can instead consider it as a function of frequency ω_n [10]:

$$F(\omega_n) = \int_{-L/2}^{L/2} f(x) e^{-j\omega_n x}\, dx = Lc_n$$

Now, c_n appears in the Fourier series expansion expression above:

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{j\frac{2\pi n}{L}x}$$

[10] ω_n is a proper frequency variable. The subscript n is just to remind us that for a given period value L there will be a certain n coefficient that will give us a desired spatial frequency. This need to separately describe Fourier coefficient numbers n will be redundant as soon as $L \to \infty$

so we will have to replace this by Lc_n. This will leave us with a factor $1/L$ in this expression, a factor which is proportional to the spacing between the spatial frequencies corresponding to two consecutive Fourier series coefficients. Let us represent this frequency increment explicitly:

$$\Delta\omega_n = \omega_{n+1} - \omega_n = \frac{2\pi(n+1)}{L} - \frac{2\pi n}{L} = \frac{2\pi}{L}$$

So the expansion expression becomes

$$f(x) = \sum_{n=-\infty}^{+\infty} Lc_n \frac{\Delta\omega_n}{2\pi} e^{j\omega_n x} = \frac{1}{2\pi} \sum_{n=-\infty}^{+\infty} F(\omega_n)\, \Delta\omega_n\, e^{j\omega_n x}$$

Now we are in a position to derive the forms as $L \to \infty$. The transform of the spatial function $f(x)$ to a function of frequency $F(\omega)$ becomes

$$F(\omega) = \int_{-\infty}^{+\infty} f(x) e^{-j\omega x}\, dx$$

while in the limit as $L \to \infty$ the summation in the expansion expression becomes an integral, giving the inverse transform from a function of frequency $F(\omega)$ to a spatial function $f(x)$

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{j\omega x}\, d\omega$$

where we have dropped the subscript n in the frequency variable ω.

We do not pursue the Fourier transform further here as we are more interested in the analysis of parts of non-period functions or blocks of data in a signal, and thus in returning to the Fourier series-based analysis.

4.4.2 Half-range and Quarter-range series

The principal objective of this chapter has been to consider the analysis of part of a function of a continuous variable or a contiguous block of data from a sampled function by considering these as if they were one cycle of a periodic function and then to bring the machinery of the Fourier series expansion to bear. We now have all the tools in place to use this approach to design transforms with properties which suit a given set of requirements. For example, we can choose to construct transforms that have only cosine terms or only sine terms, that have only even harmonics or only odd harmonics or that satisfy particular convergence properties.

The most obvious way of using the methods of Fourier series expansion to analyse a finite interval from a function is to consider that finite interval as one period of a constructed periodic function and to derive the Fourier coefficients. This is referred to as a *full-range series* and we really have no control over the properties of the derived series. All our integrals are over one full period of the constructed periodic function and depending on our implementation, we may have to use complex arithmetic. The basic idea is shown schematically in Figure 4.30(c).

By considering our finite interval from a function as one half of the period of a periodic function, we can put a copy of this function part in the other half so that the period centred on the origin is even, see Figure 4.30(d). The corresponding Fourier series then has only the dc coefficient and cosine terms. We see above that the integrals used to calculate the series coefficients are only evaluated over one half of the period as this is sufficient to encompass all of the function part that is being analysed. A major advantage of the series derived, which is called a cosine series, is that real arithmetic is sufficient. The coefficient integrals are repeated here for convenience.

Half-range cosine series:

$$a_0 = \frac{2}{L} \int_0^L f(x)\, dx = \frac{4}{L} \int_0^{L/2} f(x)\, dx,$$

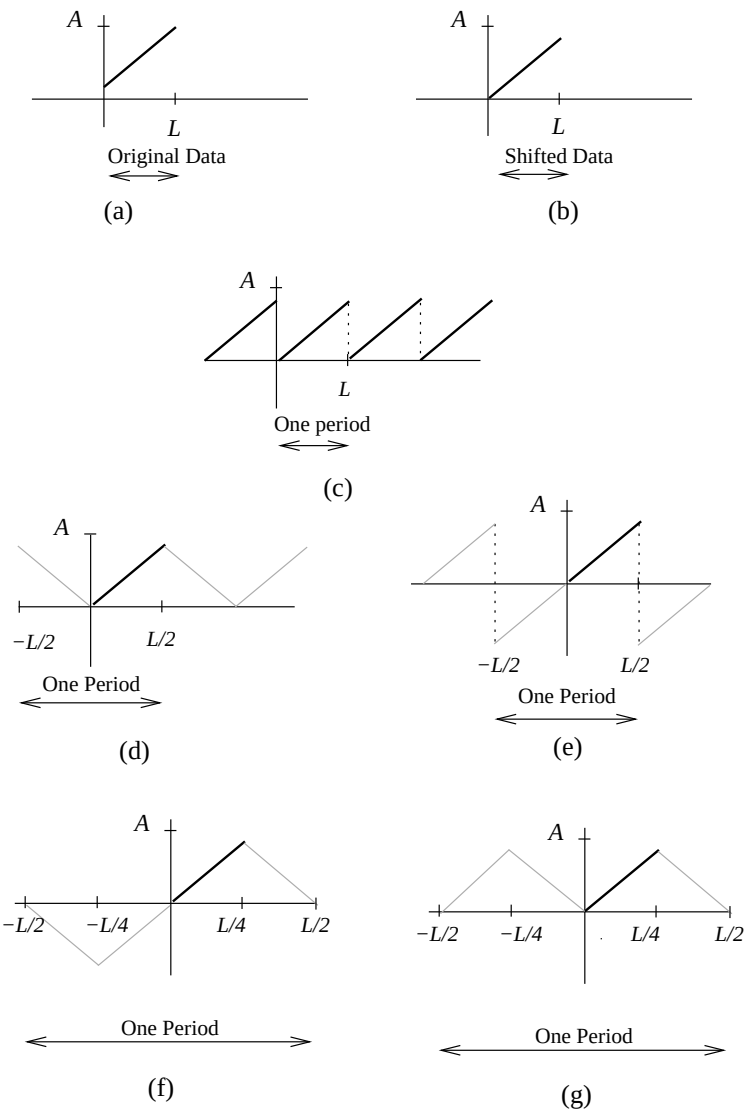


Figure 4.30: Illustration of how a function defined over a finite interval (a) can be used as one period of a periodic function (c), as one half of a period (c), (d), and as one quarter of a period (e), (f). In order to reduce discontinuities in the half-range and quarter-range series, it may be necessary to shift the function upwards or downwards as shown in (b)

$$a_n = \frac{2}{L}\int_0^L f(x)\cos n\omega x\,dx = \frac{4}{L}\int_0^{L/2} f(x)\cos n\omega x\,dx$$

$$b_n = 0$$

By using our function part to construct an odd periodic function, (Figure 4.30(e)) we get sine terms only in the corresponding Fourier series and consequently this series is usually referred to as a sine series.

Half-range sine series:

$$a_0 = 0, a_n = 0 \text{ for } n \neq 0$$

$$b_n = \frac{2}{L}\int_0^L f(x)\sin n\omega x\,dx = \frac{4}{L}\int_0^{L/2} f(x)\sin n\omega x\,dx$$

Both the sine and cosine series are examples of *half-range series*. A disadvantage of the sine series is that it usually introduces a large jump discontinuity at the join between two cycles, which means that the sine series coefficients do not converge as quickly as the cosine coefficients. Another way of saying this is that the cosine expansion tends to concentrate more signal energy in the lower frequency coefficients, which is better for the subsequent quantization and run-level steps involved in lossy coding.

In Figure 4.30(f) and 4.30(g), the construction of periodic functions in which the original function part forms one quarter of a cycle is illustrated. Naturally this results in series that are called *quarter-range series*, though these tend not to be used very often. The advantage of the periodic construction in Figure 4.30(f) is that we get a sine series expansion which has no jump discontinuities and thus has better convergence properties than the half-range sine series. The periodic construction shown in Figure 4.30(g) gives a cosine series with only even harmonics. However, this is unlikely to be a major advantage as we are still going to need as many of these even harmonics as the total of even and odd harmonics in each of the previous cases.

The cosine series expansion that derives from the periodic construction illustrated in Figure 4.30(d) turns out to have by far the greatest utility in image and video coding standards because it allows real arithmetic and has good convergence (or energy compaction) properties. We look at this issue in the next chapter, where we start with a brief look at the issue of sampled data.

Chapter 5

Transform-based Coding

5.1 Introduction

Having put all the necessary ideas from Fourier Analysis in place in the previous chapter, we now begin our discussion of transforms based on these ideas with a brief treatment of sampled data. After developing the various one- and two-dimensional transforms that are normally used in signal processing, we proceed to discuss the issues of quantization and bit-allocation that allow us to turn these transforms into data compression methods.

5.1.1 Sampled Data

The discussion in the previous chapter on Fourier series expansions and the derived sine and cosine expansions is couched mostly in terms of functions defined on finite intervals of a continuous variable. This is the most general form of these series expansions and the inner products involved in calculating the coefficients are all integrals. As stated above, when we consider a function that is defined at only a finite number of regularly-spaced domain values, we find that Fourier's theorem can also be extended to this case. Because the target data is only defined at these discrete points, we can only compare these with sampled versions of the sine and cosine functions also only defined at the same sample points. Now our inner products (the basis of the functional or sampled data comparisons) are all dot products. The changeover from functions defined on a continuous interval to sampled functions is illustrated in Figure 5.1. We can now legitimately speak of a *discrete* cosine transform, say.

There is no fixed number of samples that should be in the block of data from which we construct the half-range series. The only restriction is that our (sine or cosine) basis functions should be sampled at the same number of points and the same point locations. However, for various memory and algorithmic reasons, the number of samples is usually taken to be a power of two.

The discrete transforms discussed below all have in common with the cosine expansion the fact that their basic functions can be considered as an orthogonal basis set for a vector space. A given block of data or function part (whichever is applicable) can then be considered as a "point" or "vector" in this vector space and it can be resolved into its components (the series coefficients) "in the direction" of the basic "vectors". The only substantive difference between the sampled case and the case of a function part defined on an interval of a continuous variable is that the latter exists in an infinite-dimensional vector space, while the sampled data is a vector in a space whose dimension equals the number of samples. As we cannot have more orthogonal basis vectors than there are vector space dimensions, this means that we will have just as many sampled basis functions as there are samples in the original data.

5.1.2 A Simple Discrete Linear Transform

The detail of Fourier series expansions can hide some of the fundamental properties of discrete linear transforms, so following Jayant & Knoll, p.513, it is instructive to consider a toy discrete transform with just two samples.

Consider the data sequence $\{x_n, n = 1, 2\}$ given by $x_1 = 2$ and $x_2 = 3$. These could correspond to a block of two contiguous samples taken from a much longer sequence. (We do not bother to consider

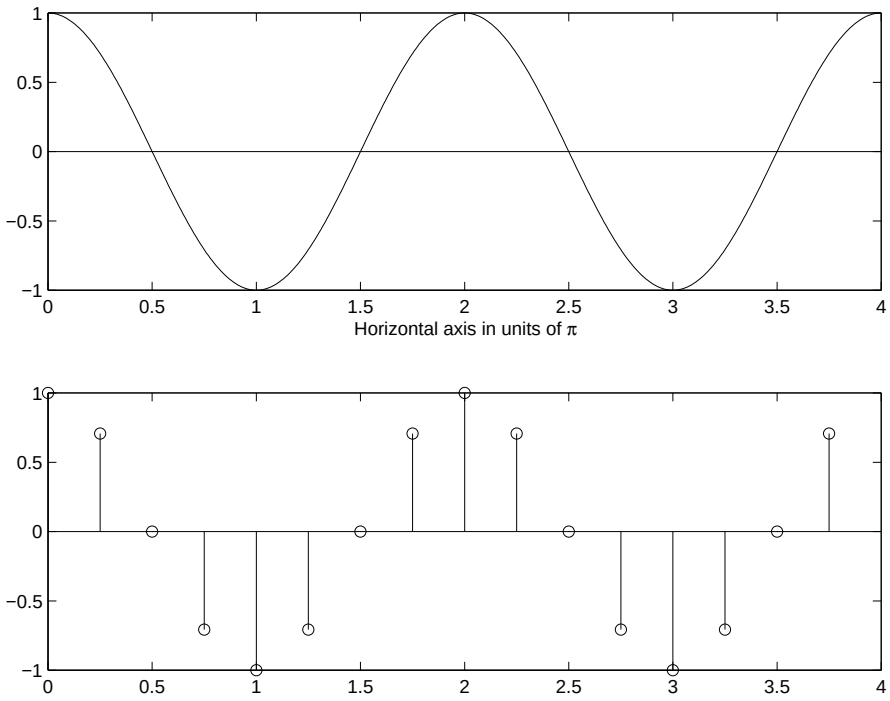


Figure 5.1: Two cycles of a cosine function (top) along with a sampled version of the same two cycles with eight samples per cycle (bottom).

these two samples as a part of a periodic function: that is only necessary above to invoke the results of Fourier's theorem). We want to introduce an orthogonal linear transform which will transform this, and any other similar pair of data, to a representation where aspects of this data other than its spatial values are made explicit. We may subsequently be able to exploit this new form for coding or other purposes.

We have two samples and because this transform is intended to be invertable we will thus have two coefficients in the transformed domain. Consequently, we will need two non-parallel basis vectors, each with two components, against which to compare our given data. It is also usually desirable to make the basis vectors orthogonal so that they represent completely independent aspects of the original data. Finally, to avoid having to deal with problems of scale, we assume that the basis vectors are unit vectors. (Technically speaking we are specifying a complete orthonormal basis for a 2-D real vector space). A standard pair of vectors which fit the given requirements are

$$\mathbf{u}_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and

$$\mathbf{u}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Relative to this pair of orthonormal basis vectors, the data set $\{2, 3\}$ considered as a vector

$$\mathbf{x} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

has components given by the dot products:

$$X_1 = \mathbf{u}_1 \cdot \mathbf{x} = \mathbf{u}_1^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{5}{\sqrt{2}}$$

$$X_2 = \mathbf{u}_2 \cdot \mathbf{x} = \mathbf{u}_2^T \mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \frac{-1}{\sqrt{2}}$$

Note that for unspecified values of the sequence $\{x_1, x_2\}$ the transform coefficients represent the sum and difference of the pair respectively:

$$X_1 = \frac{1}{\sqrt{2}}[x_1 + x_2]$$

$$X_2 = \frac{1}{\sqrt{2}}[x_1 - x_2]$$

It is possible to reconstruct the original sequence from the transform coefficients as the weighted sum of the transform basis vectors with the transform coefficients as the weights:

$$\mathbf{x} = X_1\mathbf{u}_1 + X_2\mathbf{u}_2 = \left(\frac{5}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \left(\frac{-1}{\sqrt{2}}\right)\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Even in this simple 2-D vector-space representation of the data, it is possible to make a frequency interpretation of the transform coefficients. As X_1 is the average of the input data values, it is the low-frequency or *dc* coefficient. On the other hand, as X_2 is the difference of the two data values, it represents the extent to which there is "high" frequency or change in the data.

In this simple 2-D case where it is possible to visualize the original and transformed bases directly, it is instructive to examine what the various elements of this transform coding scheme correspond to geometrically. This is shown in Figure 5.2. Notice that, notwithstanding the interpretation of the components in the direction of the transform basis vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ as respectively low frequency and high-frequency components, the transform basis vectors are simply a rotated (and reflected) version of the standard basis vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, the vectors whose components are

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

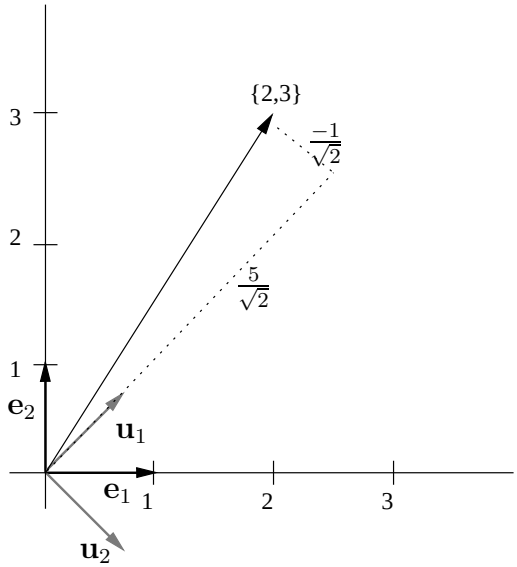


Figure 5.2: Illustration of the original data $\{2, 3\}$ expressed as a 2-D vector relative to the default coordinate vectors $\mathbf{e}_1$ and $\mathbf{e}_2$, and relative to the transform basis vectors $\mathbf{u}_1$ and $\mathbf{u}_2$.

Imagine now that our two data samples came from a low-pass filtered data set where large magnitude changes between samples were eliminated and there there was a strong correlation between consecutive samples. In any two element block of data from such a data set, the second component will tend to have a similar value to the first. If we were to plot a large population of such two-element data blocks, we would find them distributed in the manner shown in Figure 5.3. The strong correlation between the data pair elements is reflected in the way they are distributed approximately along the diagonal in the

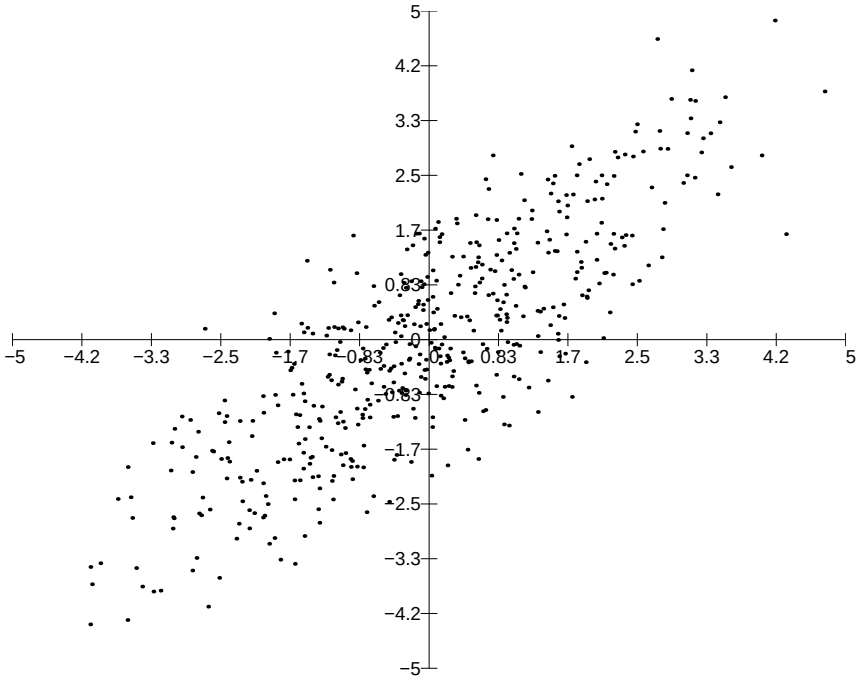


Figure 5.3: A population of consecutive data pairs from a data set which was low-pass filtered.

$\mathbf{e}_1, \mathbf{e}_2$ coordinate system. That is, in each case, $x_1 \approx x_2$. In contrast to this, the components (X_1, X_2) with respect of the transform basis $\mathbf{u}_1, \mathbf{u}_2$ is decorrelated and in particular

$$E[X_1 X_2] = E[X_1^2] - E[X_2^2] = 0$$

We can also observe that the variance along the $\mathbf{e}_1$ and $\mathbf{e}_2$ directions is approximately equal, while the variance along the $\mathbf{u}_1$ direction is greater than that along the $\mathbf{u}_2$ direction. In general this means that we would have a smaller mean-square error if we only used the transform component X_1 to represent the original data, than we would if we were to use only the transform component X_2. For example, in the case of the original (strongly correlated) data, spatial-domain values $\{y_1, y_2\}$ as approximate reconstructions of $\{x_1, x_2\}$ on the basis of X_1 only would be $\{y_1, y_2\} = \{2.5, 2.5\}$ while $\{y_1, y_2\}$ as approximate reconstructions of $\{x_1, x_2\}$ on the basis of X_2 only would be $\{y_1, y_2\} = \{-0.5, 0.5\}$.

It is usually more convenient to represent a linear transform such as the one just described in terms of a matrix operation, than constantly enumerating the individual transform vectors. With the transform basis vectors arranged as the *rows* of a matrix, as in

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and $\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}$, we can write

$$\mathbf{X} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \mathbf{A}\mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

and

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \mathbf{B}\mathbf{X} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$$

In *transform coding* we transmit quantized versions of the transform coefficients. This quantization step is a separate one independent of the lossless transformation step. Following the quantization step we are only able to reconstruct an approximate version of the original data given by

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \mathbf{B}\, Q[\mathbf{X}] = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} Q[X_1] \\ Q[X_2] \end{bmatrix} = Q[X_1]\mathbf{u}_1 + Q[X_2]\mathbf{u}_2$$

where $Q[.]$ represents the quantization scheme we apply to the transform coefficients.

5.1.3 General Transform Properties

In general, invertable linear transforms with the property of producing uncorrelated coefficients in an N-dimensional space are defined by *unitary* $N \times N$ matrices, i.e. matrices which satisfy

$$\mathbf{A}^{-1} = \mathbf{A}^{*T}$$

where * is the complex conjugate operation, or by *orthogonal* $N \times N$ matrices, i.e. matrices which satisfy

$$\mathbf{A}^{-1} = \mathbf{A}^{T}$$

Orthonormality is the property that all the transform basis vectors (the columns of $\mathbf{A}$) are unit vectors. It means that the sum of the variance of the transform coefficients $\sigma^2_{X_i}$ equals the sum of the variance of the original data components $\sigma^2_{x_i}$

As we see in the $N = 2$ example above, unitary transforms preserve the signal energy, or equivalently, the length of the signal vector in the N-dimensional vector space. In mathematical terms, if $\mathbf{X} = \mathbf{Ax}$ then

$$\|\mathbf{X}\|^2 = \|\mathbf{x}\|^2$$

where the Euclidian vector *norm* operation $\|.\|$ is defined by

$$\|\mathbf{x}\|^2 = \sum_{i=1}^{N} |x_i|^2 = \mathbf{x}^{*T}\mathbf{x}$$

This is because a unitary transform can be considered as simply a rotation (and possibly reflection) of the original coordinate vectors.

In the simple $N = 2$ case above, the transform is deliberately chosen to decorrelate the spatial-domain data, given that these low-pass data are strongly correlated. The strong correlation between the data in turn means that the signal energy is roughly equally distributed in the spatial coefficients. The transform is thus being chosen to concentrate the energy in one coefficient at the expense of the other. In fact in general unitary transforms are chosen for their ability to concentrate a large fraction of the average energy in a few transform coefficients. As the total energy is preserved, this means that some coefficients carry very little signal energy. We see in the discussion on half-range series expansions above that in general, cosine series of functions defined on a continuous interval have better *energy compaction* than sine series as they do not have to contend with built in discontinuities. This property also carries to discrete cosine transforms. However, we see below that it is the Karhunen-Loeve transform which by design has the best energy compaction properties of all linear transforms. Again, a consequence of the tendency to to compact the energy is the tendency to decorrelate transform coefficients, so the KLT is also the optimal linear transform for decorrelating coefficients.

In general, to get significant coding gains using 1-D transforms on speech signals, large values of N are needed, e.g. $N = 128$ or $N = 256$. With transform methods applied to 2-D problems, such as image coding, the coding gain saturates at transform sizes of the order of 8×8.

The following diagram is intended as an aide memoir for the relationship between the original data, the transform coefficients, the transform basis vectors and the forward and inverse transform matrices ($\mathbf{X} = \mathbf{Ax}$, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{X}$):

$$\begin{bmatrix} X_0 \\ \vdots \\ X_{n-1} \end{bmatrix} = \begin{bmatrix} \longleftarrow \mathbf{u}_0^T \longrightarrow \\ \vdots \\ \longleftarrow \mathbf{u}_{n-1}^T \longrightarrow \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

$$\begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} \uparrow & & \uparrow \\ \mathbf{u}_0 & \cdots & \mathbf{u}_{n-1} \\ \downarrow & & \downarrow \end{bmatrix} \begin{bmatrix} X_0 \\ \vdots \\ X_{n-1} \end{bmatrix}$$

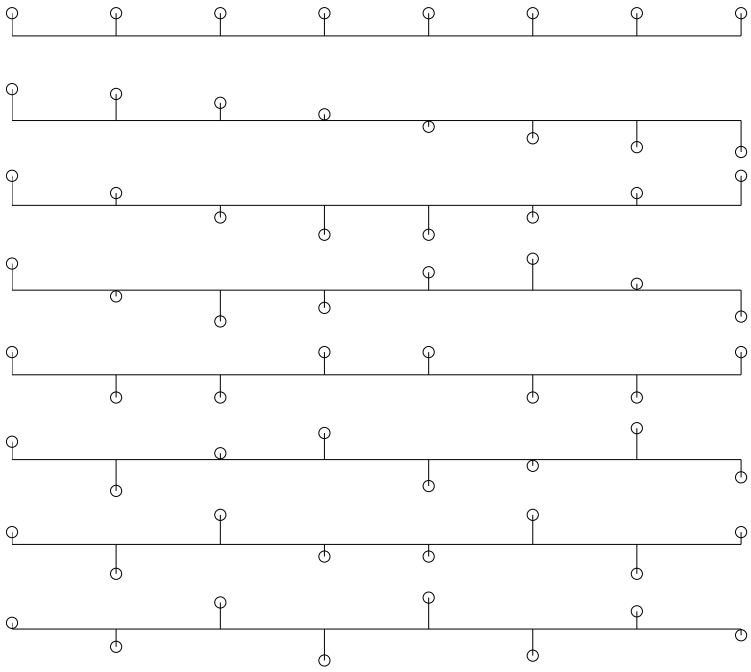


Figure 5.4: Plot of the basis functions for an eight sample 1-D DCT.

5.2 One-dimensional Discrete Orthogonal Transforms

It is worth examining the characteristics and properties of some of the more common discrete orthogonal transforms in their 1-dimensional forms, before looking at the 2-D versions of the ones most often used in image processing.

5.2.1 The Discrete Cosine Transform

As discussed above, the key advantages of the cosine expansion are the fact that it requires only real arithmetic and the relatively faster convergence compared to the sine expansion. The discrete version of the cosine expansion, which is called the *discrete cosine transform* retains these properties.

Recall that the cosine expansion is a half-range series. That is, it consists of the set of non-zero coefficients derived from the application of a Fourier series expansion to an even-symmetry doubled up version of the original data block. Because of the doubling up involved, the fact the the original Fourier expansion functions all had an integer number of cycles in the finite interval, and the fact that we only need to do the inner products to calculate the expansion coefficients over half of the interval we see that the cosine expansion functions will have either an integer or half-integer number of cycles. Effectively these functions with a half-integer number of cycles capture the changes that would have been captured by the sine functions, if we were doing a normal Fourier expansion.

The first row of the $N \times N$ DCT matrix is defined by

$$\mathbf{c}_0 = \frac{1}{\sqrt{N}}$$

and the remaining rows for $k = 1$ to $N - 1$ are given by

$$\mathbf{c}_k = \sqrt{\frac{2}{N}} \cos \frac{(2n+1)k\pi}{2N}$$

where n ranges from 0 to $N-1$ down each column. The contents of each column is plotted as a horizontal row in Figure 5.4.

For $N = 2$ the DCT matrix is

$$\mathbf{C}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos\frac{\pi}{4} & \cos\frac{3\pi}{4} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

For $N = 4$ the DCT matrix is

$$\mathbf{C}_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ a & b & -b & -a \\ 1 & -1 & -1 & 1 \\ b & -a & a & -b \end{bmatrix} \qquad \begin{aligned} a &= \sqrt{2}\cos\tfrac{\pi}{8} = 1.306 \\ b &= \sqrt{2}\cos\tfrac{3\pi}{8} = 0.541 \end{aligned}$$

Properties of the DCT

1. The DCT matrix is real and orthogonal: $\mathbf{C} = \mathbf{C}^*, \quad \mathbf{C}^{-1} = \mathbf{C}^T$
2. The forward and inverse transform have identical matrices (up to a scaling factor).
3. The DCT has a "Fast" version based on the FFT ($2N \log_2 N$ multiply-accumulates). The procedure consists of extending the N-sample input data block to a $2N$ block with even symmetry, taking a size $2N$ DFT and retaining just N terms from the output. This is of course possible because only the cosine terms are non-zero. [1]
4. The DCT has very good energy compaction properties.

5.2.2 The Discrete Sine Transform

The columns of the $N \times N$ DST matrix are given for $k = 0$ to $N - 1$ by

$$\mathbf{s}_k = \sqrt{\frac{2}{N+1}} \sin\frac{(n+1)(k+1)\pi}{N+1}$$

where n ranges from 0 to $N-1$ down each column. The contents of each column is plotted as a horizontal row in Figure 5.5.

Properties of the DST

1. The DST matrix is real, symmetric and orthogonal: $\mathbf{S} = \mathbf{S}^* = \mathbf{S}^{-1} = \mathbf{S}^T$
2. The forward and inverse transform are identical.
3. The DST has a "Fast" version based on the FFT ($2N \log_2 N$ multiply-accumulates).

5.2.3 The Hadamard and Walsh-Hadamard Transforms

The elements of the orthonormal basis vectors of the Hadamard transform only take the values ± 1. The smallest Hadamard matrix is

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

One construction for higher order Hadamard matrices is

$$\mathbf{H}_{2N} = \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{bmatrix}$$

which means that all such matrices have dimensions which are integer powers of 2. For example,

$$\mathbf{H}_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

[1] Be very careful about this point. An N-point DCT is *not* the same as the cosine terms of an N-point DFT. This discussion here is about the cosine terms of a $2N$-point symmetrically extended DFT.

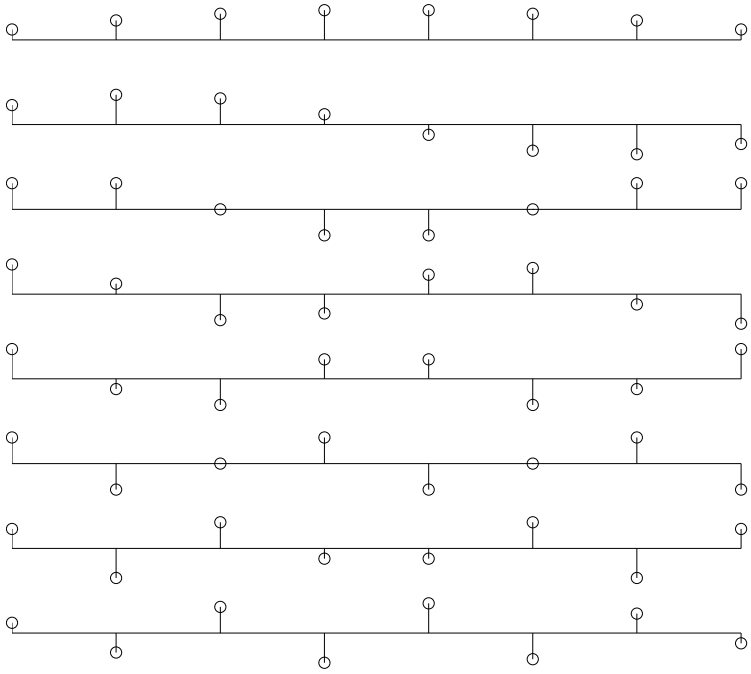


Figure 5.5: Plot of the basis functions for an eight sample 1-D DST.

It is also possible to have orthogonal ± 1 element matrices whose dimensions are multiples of 4, but these will need to be generated by same other method.

Frequency Concepts

In the discussion on the Fourier Transform above, we use the concept of a frequency variable ω and the term *spatial frequency* can also be applied to this concept. There is a direct relationship between this spatial frequency and the number of *zero-crossings* of the corresponding sine or cosine functions of the Fourier transform. However, in cases where we are dealing with orthogonal functions which are not sinusoidal, it is possible to use the concept of zero-crossings to capture what is termed *transform frequency* and give a relative ordering of the basic vectors. In the case of the Hadamard transform, this transform frequency is given the special name *sequency*, where it is defined as half the number of sign changes in one period of the sequence. Note that because of the "square-wave" character of the Hadamard basis vectors, a vector with just one zero-crossing may still capture high spatial-frequency harmonics of the signal being analysed. Consequently, one should only use transform frequency or sequency as a means of ordering vectors within a given transform. The rows of the Hadamard matrix defined above are not in sequency order, but the order they are in is referred to as the *Hadamard order*. Sequency ordered Hadamard matrices are sometimes referred to as Walsh-Hadamard matrices. The functions of a continuous variable referred to as Walsh functions also only take the values ± 1 and like the infinite set of sine and cosine functions in the Fourier expansion form a complete orthonormal basis for the set of square-integrable functions defined on the same finite interval of a continuous variable. In most cases the Hadamard basis vectors can be generated as samples of the Walsh functions.

The sequency-ordered 4×4 Walsh-Hadamard matrix is

$$\mathbf{H}'_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

and its sequency order is $0, 0.5, 1, 1.5$. By way of contrast, the sequency order of the Hadamard matrix $\mathbf{H}_4$ is $0, 1.5, 0.5, 1$.

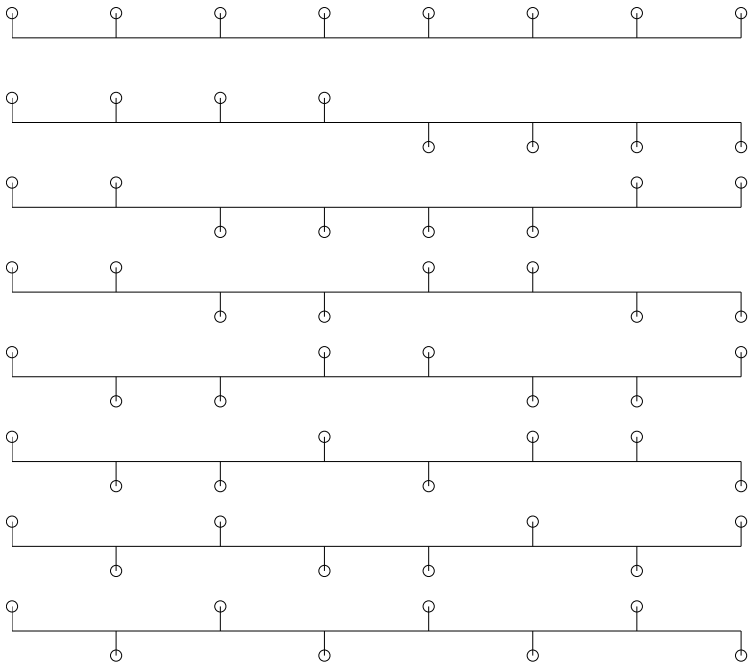


Figure 5.6: Plot of the basis functions for an eight sample 1-D discrete Walsh-Hadamard transform arranged in sequency order.

The elements of the $N \times N$ Hadamard matrix $\mathbf{H}_N$, $N = 2^n$, can be calculated as

$$h(k,m) = \frac{1}{\sqrt{N}}(-1)^{b(k,m)}, \quad 0 \le k, m \le N-1$$

where

$$b(k,m) = \sum_{i=0}^{n-1} k_i \, m_i$$

and where $\{k_i\}$ and $\{m_i\}$ are the bit values of the binary representations of the values k and m:

$$k = 2^{n-1}k_{n-1} + \cdots + 2k_1 + k_0$$
$$m = 2^{n-1}m_{n-1} + \cdots + 2m_1 + m_0$$

For later reference, the elements of the 2-D version of the Hadamard transform are given by the expression

$$h(k,m,p,q) = \frac{1}{N}(-1)^{b(k,m,p,q)}, \quad 0 \le k, m, p, q \le N-1$$

where

$$b(k,m,p,q) = \sum_{i=0}^{n-1} (p_i \, k_i + q_i \, m_i)$$

and as before $\{k_i\}$, $\{m_i\}$, $\{p_i\}$ and $\{q_i\}$ are the bit values of the binary representations of the values k, m, p and q. Similarly, the elements of the 2-D Walsh-Hadamard sequency-ordered matrix are given by

$$h'(k,m,p,q) = \frac{1}{N}(-1)^{b'(k,m,p,q)}, \quad 0 \le k, m, p, q \le N-1$$

where

$$b'(k,m,p,q) = \sum_{i=0}^{n-1} [g_i(p) \, k_i + g_i(q) \, m_i]$$

and

$$\begin{aligned} g_0(p) &= p_{n-1} \\ g_1(p) &= p_{n-1} + p_{n-2} \\ g_2(p) &= p_{n-2} + p_{n-3} \\ &\vdots \\ g_{n-1}(p) &= p_1 + p_0 \end{aligned}$$

Properties of the Hadamard transform

1. The Hadamard and Walsh-Hadamard matrices are real, symmetric and orthogonal: $\mathbf{H} = \mathbf{H}^* = \mathbf{H}^{-1} = \mathbf{H}^T$

2. The Hadamard transform has a "Fast" version ($2N \log_2 N$ accumulates). In addition, because the basis vectors only have ± 1 components, no multiplications are required in transform calculations.

3. The Hadamard order for the sequency mentioned above turns out to be the bit-reversed Gray code representation of its sequency.

4. The Hadamard transform has reasonable energy compaction properties for strongly correlated signals. So for example, given a low-pass signal and a sequency-ordered Walsh-Hadamard matrix such as $\mathbf{H}'_4$, the coefficient variances should decrease monotonically with increasing coefficient index or sequency. Both the DHT and the DWHT are not near as good for transform coding of speech signals, but they tend to have an advantage in representing image data, which tends to have lots of intensity discontinuities.

5. Hadamard transforms do not diagonalise Toeplitz matrices, but they do diagonalise dyadic matrices.

5.2.4 The Haar Transform

The Haar transform is based on the orthogonal Haar matrix which consists of ± 1 and zero elements only. Examples for $N = 4$ and $N = 8$ are shown below.

$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

$$\mathbf{H}_8 = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

Notice that the Haar basis vectors effectively extract information about the signal at resolutions increasing as powers of two, and at more and more localised locations.

The Haar basis vectors for $N = 2^n$ are defined as follows. For $k = 0, \ldots, N-1$, the integer k can be uniquely decomposed as

$$k = 2^p + q - 1$$

Then for the fractional values $x = \frac{r}{N}$, $r = 0, \ldots, N-1$

$$h_0(x) = \frac{1}{\sqrt{N}}$$

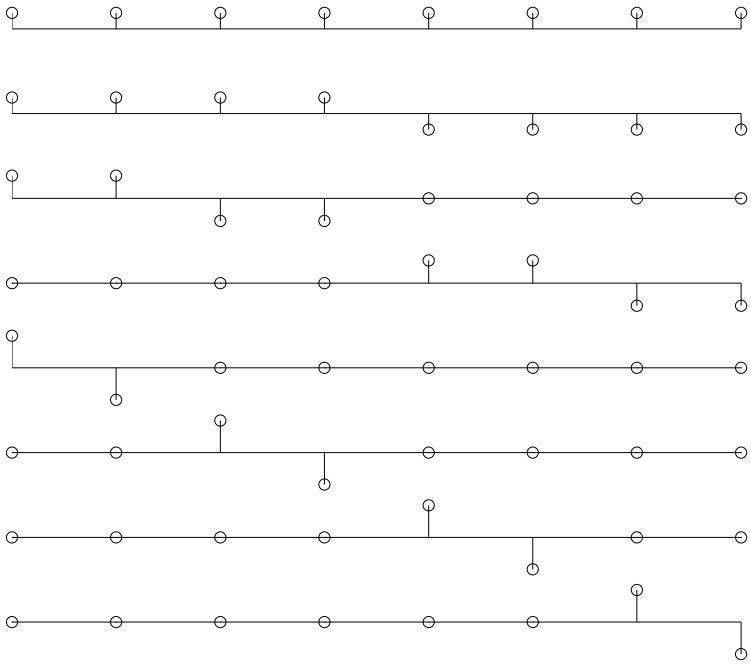


Figure 5.7: Plot of the basis functions for an eight sample 1-D discrete Haar transform.

and

$$h_k(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2}, & \frac{q-1}{2^p} \leq x < \frac{q-\frac{1}{2}}{2^p} \\ -2^{p/2}, & \frac{q-\frac{1}{2}}{2^p} \leq x < \frac{q}{2^p} \\ 0, & otherwise \end{cases}$$

Properties of the Haar transform

1. The Haar matrix is real and orthogonal ($\mathbf{H} = \mathbf{H}^*$, $\mathbf{H}^{-1} = \mathbf{H}^T$).
2. The Haar transform is very fast. For N data points it can be implemented in $O(N)$ operations.
3. The basis vectors (rows of the Haar matrix) are sequency ordered.
4. The Haar transform has poor energy compaction for image-type data.

5.2.5 The Slant Transform

The slant transform is an orthogonal transform designed to have the following properties:

1. It should have a constant basis vector.
2. It should have a slant (or ramp) basis vector.
3. It should be sequency ordered.
4. It should be computationally fast.
5. It should have high energy compaction.

For $N = 2$ the slant matrix is given by

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

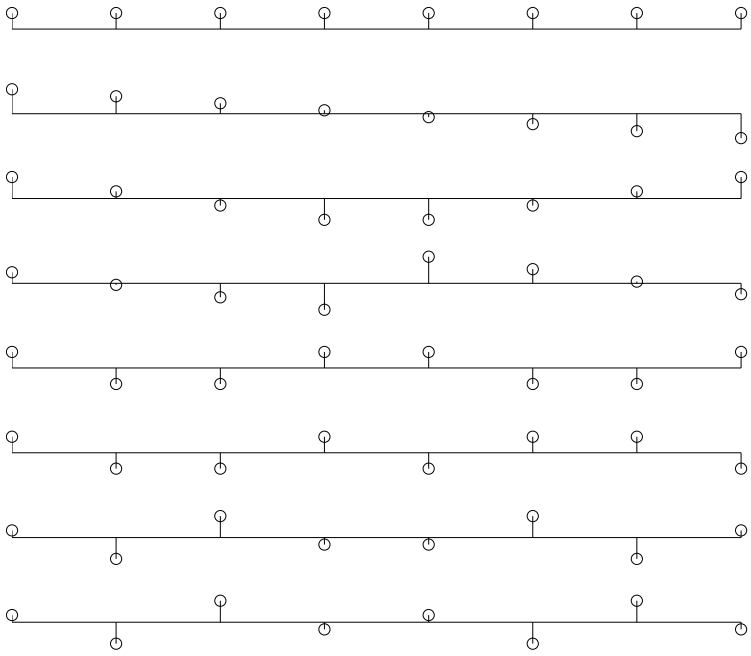


Figure 5.8: Plot of the basis functions for an eight sample 1-D discrete slant transform.

The slant transform matrix for $N = 4$ is

$$\mathbf{H}'_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{bmatrix}$$

which is orthonormal and sequency-ordered.

5.2.6 The Discrete Fourier Transform

The unitary form of the N-element discrete Fourier transform (DFT) is defined by the $N \times N$ DFT matrix, [2] where

$$\mathbf{F} = \tfrac{1}{\sqrt{N}}(W_N)^{kn}, \quad \text{and} \quad W_N = exp\big(-j\tfrac{2\pi}{N}\big) \quad \text{for} \quad 0 \le k, n \le N-1$$

So

$$\mathbf{X} = \mathbf{F}\mathbf{x}, \quad \mathbf{x} = \mathbf{F}^*\mathbf{X}$$

where

$$\mathbf{F} = \left\{ \tfrac{1}{\sqrt{N}}\, e^{-j2\pi kn/N} \right\}_{k,n=0,\cdots,N-1} \qquad \mathbf{F}^* = \left\{ \tfrac{1}{\sqrt{N}}\, e^{j2\pi kn/N} \right\}_{k,n=0,\cdots,N-1}$$

For $N = 2$ and $N = 4$ the DFT matrices are

$$\mathbf{F}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} e^0 & e^0 \\ e^0 & e^{-j\pi} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$\mathbf{F}_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

[2] Note that in the non-unitary form of the DFT, the two normalization factors $\frac{1}{\sqrt{N}}$ are usually gathered together and applied as a factor $\frac{1}{N}$ to the inverse transform.

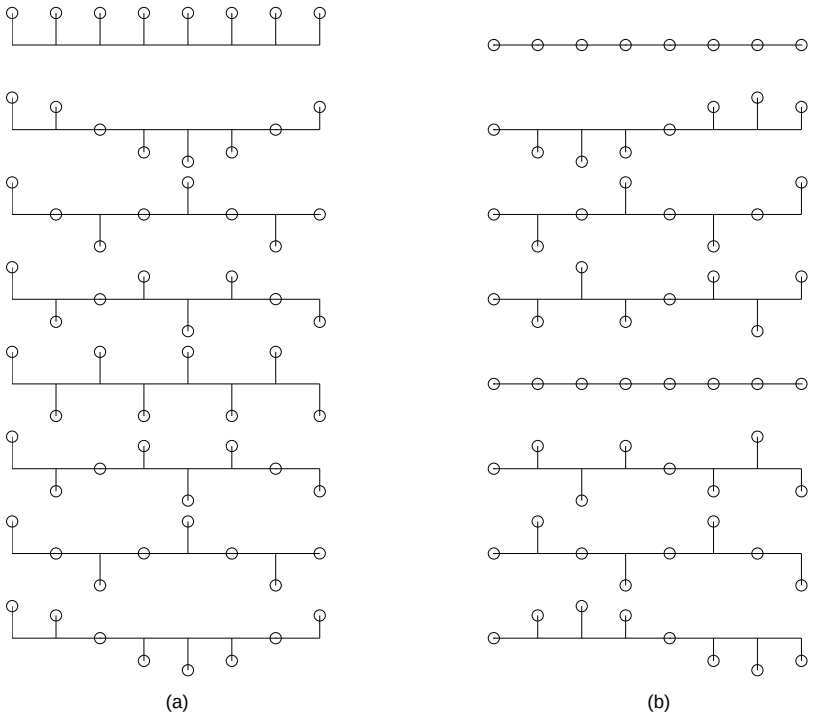


Figure 5.9: Plot of the real parts (a) and imaginary parts (b) of the basis functions for an eight sample 1-D discrete Fourier transform.

Now, $\mathbf{F}$ and $\mathbf{F}^*$ have complex elements, so it would seem that we would end up having to store both real and complex parts for all N coefficients, in other words, $2N$ numbers. However, for real-valued inputs, it is possible to show that the DFT transform coefficients exhibit a conjugate symmetry:

$$X_k = X^*_{N-k}, \;\; k = 1, \cdots, \frac{N}{2} - 1$$

so we actually only need to store N real numbers or $N/2$ complex numbers to capture the discrete Fourier representation.

Properties of the discrete Fourier transform

1. $\mathbf{F}$ is symmetric, and
$$\mathbf{F}^{-1} = \mathbf{F}^{*T} = \mathbf{F}^*$$
2. Extending the transform coefficient index k beyond the limits $[0, N-1]$ or the inverse transform coefficient index n beyond the limits $[0, N-1]$ results in periodic sequences with period N.
3. The DFT of a finite sequence x_n is the sampled spectrum of the finite sequence extended to infinity by zeros outside the interval $[0, N-1]$. Let the zero extended sequence $\tilde{x}(n)$ be defined by
$$\tilde{x}(n) = \begin{cases} x(n), & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases}$$
Then its (unitary) *Fourier transform* is given by
$$\tilde{X}(\omega) = \frac{1}{\sqrt{N}} \sum_{n=-\infty}^{n=+\infty} \tilde{x}(n)\, e^{-j\omega n} = \frac{1}{\sqrt{N}} \sum_{n=0}^{n=N-1} \tilde{x}(n)\, e^{-j\omega n}$$
and the (unitary) *discrete Fourier transform* is then a sampled version of this continuous transform at N points:
$$X(k) = \tilde{X}\left(\frac{2\pi k}{N}\right)$$

4. The N-element DFT can be implemented by the fast Fourier transform (FFT) algorithm in $O(N \log_2 N)$ operations if N is a power of 2, where one "operation" is a real multiply-accumulate.

5. The DFT of a real-valued sequence is conjugate symmetric about $N/2$. That is, considering the periodic extension of the transform coefficients $X(k)$ we have

$$X(-k) = X(N-k)$$

The DFT coefficients or frequencies for $k = N/2, \cdots, N-1$ are exactly the negative frequencies

$$\omega = \left(\tfrac{2\pi}{N}\right) k, \quad \text{for} \quad k = -\tfrac{N}{2}, \cdots, -1$$

6. The (unitary) DFT basis vectors are the orthogonal eigenvectors of all circulant matrices. The *eigenvalues* of a circulant matrix are given by the DFT of its first column. Thus the DFT/IDFT matrices will diagonalise any circulant matrix

7. The DFT of the circular convolution of two sequences is equal to the product of their respective DCTs.

8. A linear convolution of two sequences can be calculated with the FFT version of the DFT by embedding it in a circular convolution. This is done by padding each sequence out with zeros to a power of 2 length which is greater than the combined length of the original sequences.

5.3 Two-dimensional Discrete Transforms

Given two-dimensional data, such as an 8×8 data block extracted from an image, there are two different ways that we can proceed to apply transform methods. The first is to re-arrange the 8×8 data into a 64×1 data vector in some systematic way. Depending on the transform and for what the transform coefficients are going to be used, it may not matter how this is done, but common ways are column stacking and row stacking. We are now back in the 1-D domain and all the methods and results of 1-D transforms will apply. When we need to go back to the 2-D domain, we simply reverse the stacking procedure.

The second approach is to extend the 1-D transforms to proper 2-D kernels, either by (i) developing kernels specifically (and uniquely) for 2-D, or more commonly by (ii) constructing a separable 2-D kernel as the outer product of two copies of a given 1-D kernel. Only the separable kernel approach is discussed here.

Given an $N \times N$ image data block $\mathbf{x}$, it can be expressed as a linear combination $N \times N$ basis images $\mathbf{u}_{kl}$ (the equivalent of the earlier basis vectors) as

$$\mathbf{x} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} X_{kl} \mathbf{u}_{kl}$$

where $\mathbf{u}_{kl}$ is constructed by the outer product $\mathbf{u}_{kl} = \mathbf{u}_k \mathbf{u}_l^T$. The 2-D set of transform coefficients X_{kl} are computed in two steps for separable 2-D transform kernels. The first step is a 1-D transform applied to each of the rows of the $N \times N$ image data block $\mathbf{x}$. This is represented in matrix notation by the matrix product $\mathbf{x}\mathbf{A}^T$, i.e. right-multiplication by $\mathbf{A}^T$. We then apply the 1-D transform to each of the columns of the previous result, i.e., left-multiplication by $\mathbf{A}$, giving the combined result

$$\mathbf{X} = \mathbf{A}\mathbf{x}\mathbf{A}^T$$

The following exercise is a 2-D extension of the toy $N = 2$ example in 1-D described above. (See Jayant & Noll, p.522). We want to represent the 2×2 block of pixels

$$\mathbf{x} = \begin{bmatrix} 35 & 30 \\ 25 & 15 \end{bmatrix}$$

as a linear combination of the four basis images

$$\mathbf{u}_{00} = \tfrac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{u}_{01} = \tfrac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

$$\mathbf{u}_{10} = \tfrac{1}{2}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \quad \mathbf{u}_{11} = \tfrac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

where $\mathbf{u}_{kl} = \mathbf{u}_k \mathbf{u}_l^T$ for $k, l = 0, 1$. Show that these basis images are in fact the outer product of the basis vectors:

$$\mathbf{u}_1 = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \mathbf{u}_2 = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Work out the values of the four transform coefficients $\{X_{kl}\}$ for $k, l = 0, 1$

5.3.1 The 8x8 DCT

An image representation of the basis images of the 2-D DCT is shown in Figure 5.10.

5.3.2 The 8x8 DWHT

An image representation of the basis images of the 2-D DWHT is shown in Figure 5.11

5.4 Quantization

5.4.1 Scalar Quantization

In a scalar quantization process, the amplitude of a scalar quantity x is compared with a set of decision levels. Depending on which band between decision levels the amplitude value occupies, the quantity is assigned a code corresponding to a reference value lying somewhere between the decision levels of that particular band. The scalar quantity can be an analogue signal sample as in the case of the digitization of an input signal, or it could be a digital quantity with a relatively fine spacing between values, as in the case of an accurately represented DCT coefficient against which we are trying to allocate the absolute minimum number of bits. Even though in this latter case, the scalar quantity is already quantized, it is still possible to quantize it further. If $d_j \leq x < d_{j+1}$, then after quantization, the scalar quantity $\hat{x}$ will be considered to have the value of the corresponding reconstruction level r_j. The design of a quantizer involves the selection of decision and reconstruction levels which minimize some measure of the distortion from the quantization process, usually the mean square error between x and $\hat{x}$. The error incurred by replacing an input value x by a quantized value $\hat{x}$ that is the corresponding reconstruction level is represented by the shaded region in Figure 5.13 below.

Lloyd-Max or Optimum Mean Square Quantizers

The probability distribution of the values of the scalar variable x is now important for the following reason. If a particular range of x-values occurs very frequently, it would be more important to minimize the quantization errors introduced for these values, than it would be for other less common values. This means that the decision levels should be closely spaced in the vicinity of values that occur frequently. The overall quantization error e_q introduced by N decision levels is given by

$$e_q = E[(x - \hat{x})^2] = \int_{d_0}^{d_N} (x - \hat{x})^2 \, P(x) \, dx = \sum_{i=0}^{N-1} \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, P(x) \, dx$$

which is just the sum of the mean square errors introduced in the bands between each consecutive pair of decision levels.

In some cases (e.g. N large) it may be reasonable to take $P(x)$ to be a constant value over each quantization band, i.e. $P(x) = P(r_i)$. This is illustrated in Figure 5.14. In this case the quantization

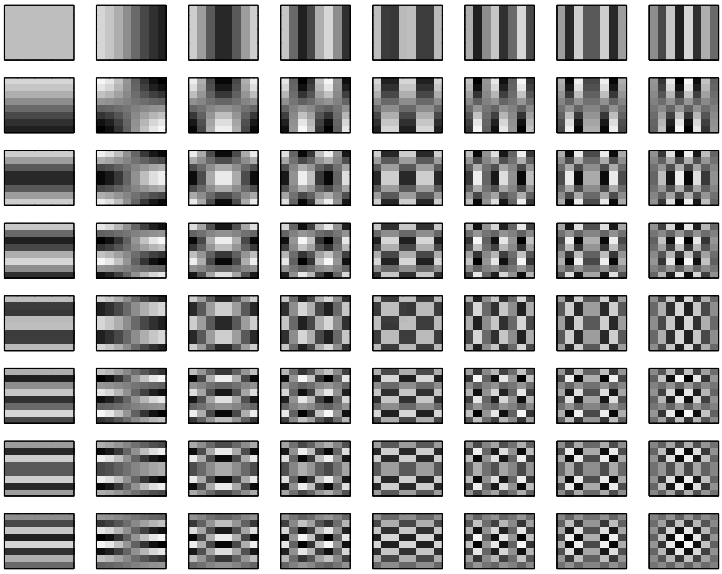


Figure 5.10: Plot of the basis images for an 8×8 2-D DCT.

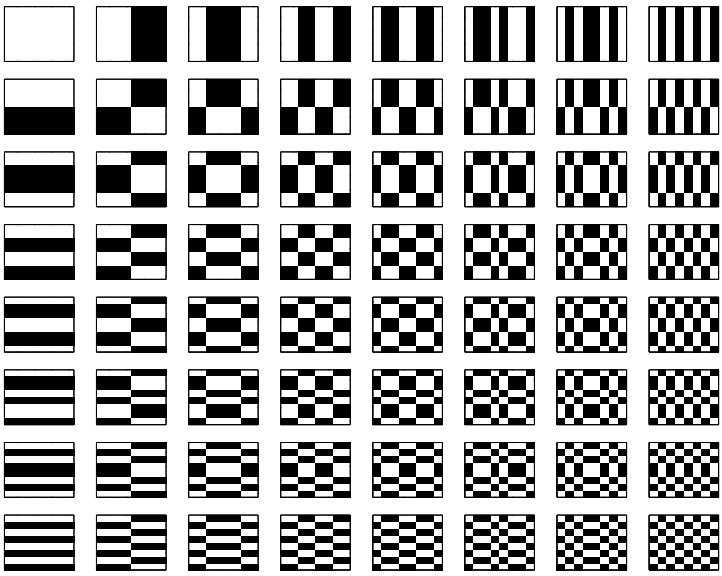


Figure 5.11: Plot of the basis images for an 8×8 2-D DWHT.

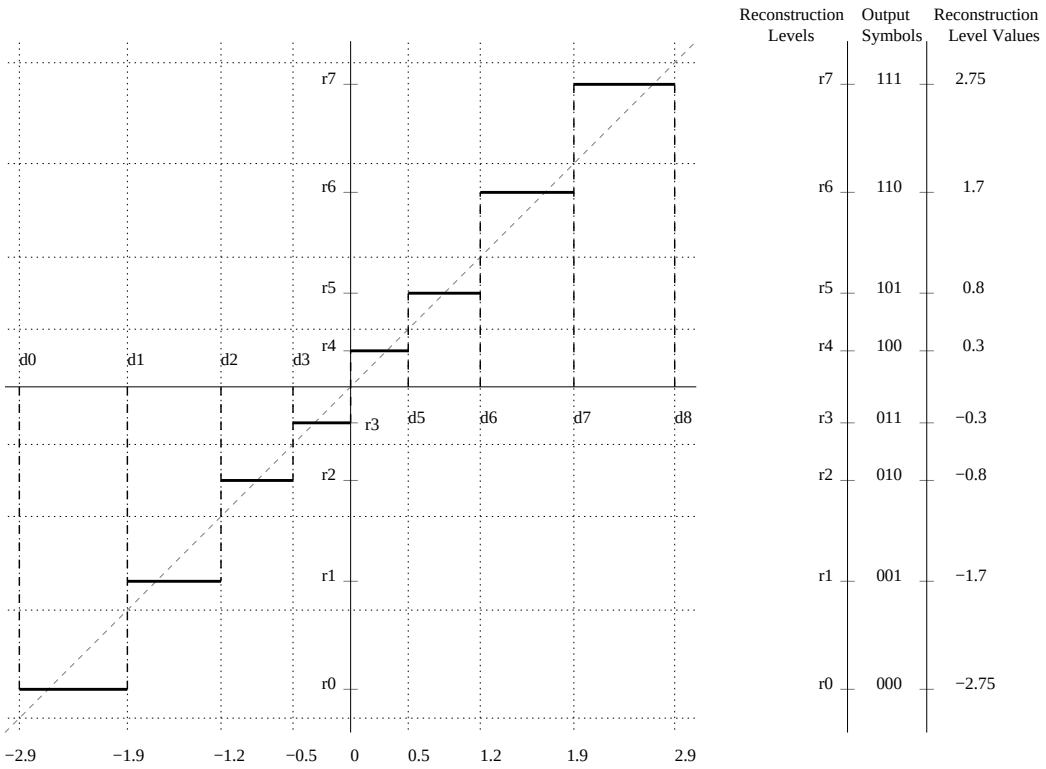


Figure 5.12: Schematic illustration of a quantization process. The input decision levels (labeled d0-d8) are shown along the horizontal (input) axis. The reconstruction levels (labeled r0-r7) corresponding to each band between a consecutive pair of decision levels are shown along the vertical (output) axis. Also shown on the right are the binary symbols corresponding to each decision band and the real-number value corresponding to each reconstruction level.

error evaluates to

$$e_q = \sum_{i=0}^{N-1} P(r_i) \int_{d_i}^{d_{i+1}} (x - r_i)^2 \, dx = \frac{1}{3} \sum_{i=0}^{N-1} P(r_i) \left[(d_{i+1} - r_i)^3 - (d_i - r_i)^3 \right]$$

By finding the value of r_i which gives $\frac{de_q}{dr_i} = 0$, we are finding the location of r_i within its quantization band that minimizes the quantization error. The answer is, not surprisingly,

$$r_i = \frac{d_{i+1} + d_i}{2}$$

The optimum location of the decision levels is a more difficult problem solvable by Lagrange multipliers, giving a set of integral equations for the values of d_i that usually need to be computed numerically.

Lloyd-Max Quantizers for Uniform Distributions

In the particular case of a uniform probability distribution over the signal range, the decision bands will all be the same width and the reconstruction levels will be equally spaced. Let Δ be the uniform width between decision levels:

$$\Delta = d_{i+1} - d_i$$

then the reconstruction levels will be at $r_i = d_i + \Delta/2$. The quantization error magnitude will thus be no larger than $\Delta/2$ and its values will be uniformly distributed over the interval $(-\frac{\Delta}{2}, \frac{\Delta}{2})$ as all input values in the interval (d_i, d_{i+1}) are equally likely. The variance of a uniform random variable with range Δ is $\frac{\Delta^2}{12}$ so this is the quantization error variance here. If the original signal is uniformly distributed over

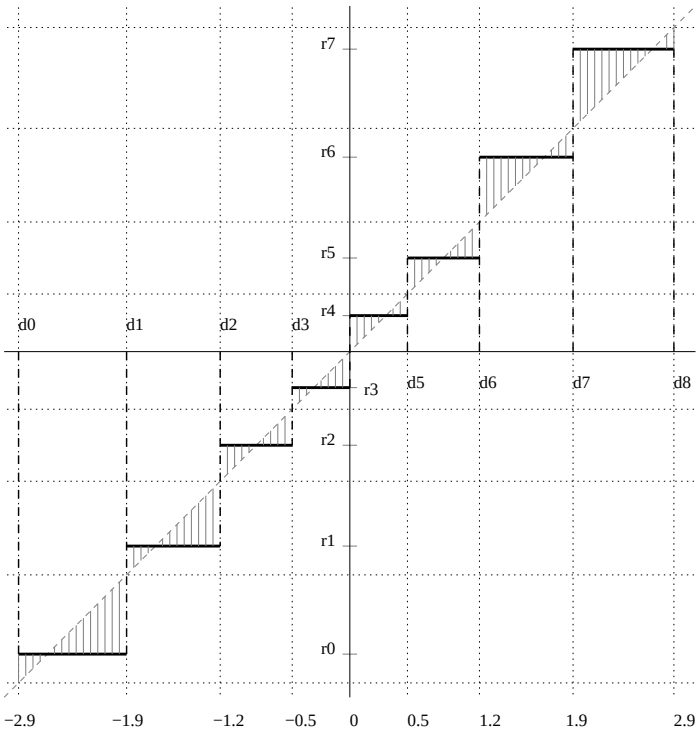


Figure 5.13: Illustration of the error introduced by a quantization process. The vertical shading shown the difference between what an unchanged output would be and the actual output at the corresponding reconstruction level.

the interval $(-\frac{A}{2}, \frac{A}{2})$ then it will have a variance of $\frac{A^2}{12}$. If we allocate n bits to a uniform quantizer, it will have 2^n decision levels covering the interval $(-\frac{A}{2}, \frac{A}{2})$ so

$$\Delta = \frac{A}{2^n}$$

and the signal to noise ratio will be

$$\frac{A^2/12}{\Delta^2/12} = \frac{1}{(1/2^n)^2} = 2^{2n}$$

In dBs this is $10 \log_{10} 2^{2n} = 6n$ dB or 6 dB per bit (see Jain, p.103). In terms of a distortion measure this result would be written as $\sigma_X^2 \, 2^{-2n}$, where $\sigma_X^2 = \frac{A^2}{12}$.

Lloyd-Max Quantizers for Non-uniform Distributions

Where the approximation of constant probability in a quantization band is not sufficiently accurate, numerical methods must be applied to finding both the optimum location of the reconstruction levels and the decision levels. In the case of standard probability distributions such as Gaussian, Laplacian and Rayleigh, however, tables of optimum values are available for popular values of N. The resulting solution sets of r_i and d_i are called *Lloyd-Max quantizers*. Some useful properties of the optimum mean square quantizer are

1. The output of the quantizer $\hat{x}$ is an unbiased estimate of the input variable x. That is, $E[\hat{x}] = E[x]$.

2. The quantization error $x - \hat{x}$ is orthogonal to the output $\hat{x}$. That is $E[(x - \hat{x}) \cdot \hat{x}] = 0$.

Companding Quantizers

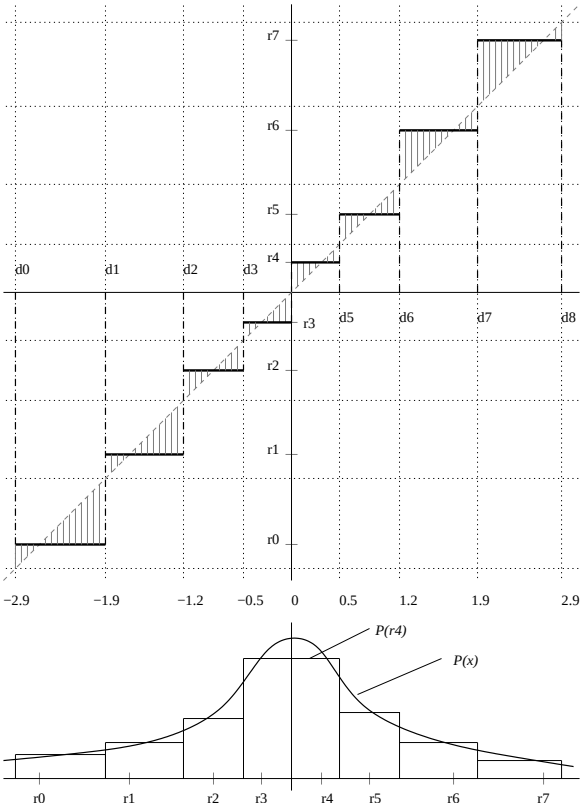


Figure 5.14: On the bottom of this version of the quantization diagram the probability distribution for the values of x is shown, along with an approximation to this distribution which is constant across each quantization band.

An alternative approach to constructing a reduced-error quantizer is the *companding quantizer*. In this case, the scalar values x are first transformed by a non-linear point transformation so that their probability distribution is uniform. They are then quantized with a linear quantizer (equal-width quantization bands) before the quantized values $\hat{x}$ are transformed by the inverse of the original non-linear point transformation. It is not surprising that the non-linear transformation function is the cumulative probability distribution of x:

$$T(x) = \int_{-\infty}^{x} P(t)\,dt - \frac{1}{2}$$

The performance of compandors is comparable to that of optimal mean-square error quantizers.

Optimum Uniform Quantizers

It is sometimes the case that the values x have a non-uniform probability distribution $P(x)$, but it is required that they be quantized by a uniform quantizer with an even number N of decision levels. We do not lose generality by assuming that the probability distribution $P(x)$ has zero mean and unity variance, see Jain, p.112. For convenience we also assume here that $P(x)$ is even. Hence the decision levels are symmetrically distributed in the interval $[-\Delta N/2, \Delta N/2]$, where $\Delta = d_{i+1} - d_i$ is the width of each quantization band. Then the quantization error is given by

$$e_q = \sum_{i=1}^{N-2} \int_{d_i}^{d_{i+1}} (x - r_i)^2\, P(x)\,dx + 2\int_{d_{N-2}}^{\infty} (x - r_{N-1})^2\, P(x)\,dx$$

The first term in this expression involves counting from the second to the second-last quantization band. The second term represents the quantization error arising from the first and the last band, and in addition

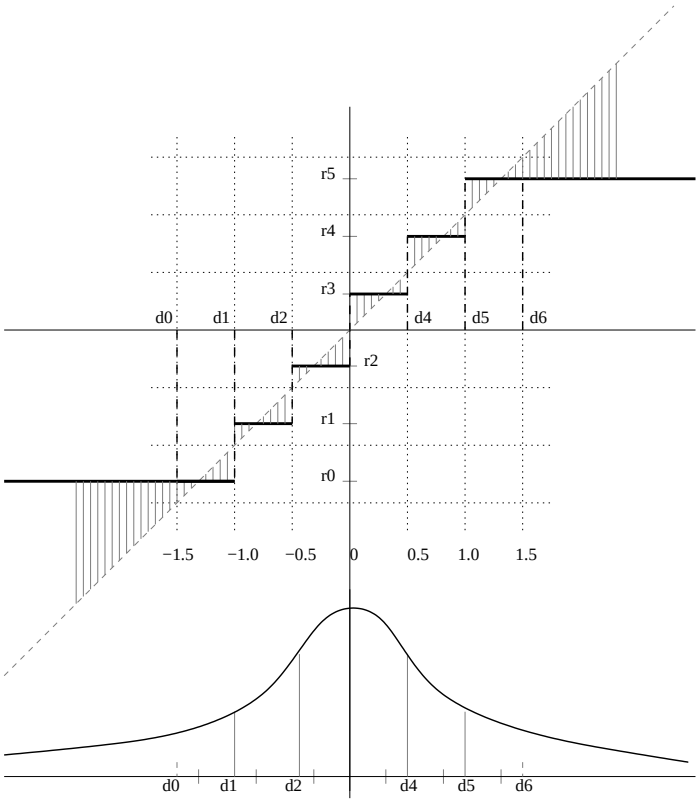


Figure 5.15: Illustration of a uniform quantizer used to quantize a non-uniformly distributed variable x defined on the interval $(-\infty, \infty)$. The only variable in the system design is the quantization band width $\Delta = d_{i+1} - d_i$ or equivalently the location of the first decision level d_0 (assuming the probability distribution $P(x)$ is even).

the two tails of the distribution which are not covered by any quantization bands (the overload regions). The expression breaks down like this because all x values greater than d_{N-1} are quantized to the last reconstruction level r_{N-1}, and similarly for $x \leq d_1$ and r_0. The situation is illustrated in Figure 5.15. Simplifying this expression and minimizing it with respect to q we get the decision levels for the *optimum uniform quantizer*.

For Gaussian probability distributions and 6-bit word-length, the optimum mean square quantizers are about 2 dB better than optimum uniform quantizers. The figure worsens to over 4 dB for Laplacian distributions. On the other hand, an optimum uniform quantizer followed by entropy coding gives lower distortion for a given bit rate than the optimum mean square quantizer without entropy coding (see Jain, p.118). Hence it is important not to consider a quantizer in isolation, but as a part of an overall rate-distortion package. Also, it is often the case that the probability distribution of the input values is not stationary and compromises may need to be made between resolution and signal range in achieving a particular dynamic range.

The distortion-rate function measures the quantizer mean square error as a function of the number of bits B. Distortion-rate models are usually of the for $f(B) = a\,2^{-bB}$ which are monotonically decreasing functions of B with the rate of decrease slowing with increasing B values. Values for the parameters a and b for optimum mean square quantizers and optimum uniform quantizers applied to Gaussian distributed data are shown in the following table (see Jain, p.118).

	$0 \leq 2^B < 5$		$5 \leq 2^B < 36$		$36 \leq 2^B < 512$	
Quantizer	a	b	a	b	a	b
Mean square Gaussian	1	1.5047	1.5253	1.8274	2.2573	1.9626
Optimum uniform Gaussian	1	1.5012	1.2477	1.6883	1.5414	1.7562

Visual Quantizers

In the context of the digitization of image samples, there are a couple of other issues worth mentioning. The principal lower limit on the number of bits per pixel is the visibility of *contouring* when the steps between reconstruction levels become noticeable. For uniform quantization, this effect begins at around 6 bits per pixel. By using a mean square quantizer matched to the image histogram (as an approximation to the underlying probability distribution) it is possible to reduce the contouring threshold by about 1 bit per pixel. In practice, however, 8-bit uniform quantization is used for the initial digitization of image samples.

Another approach that is sometimes used is to invoke the fact the the human visual system is equally sensitive to equal changes in contrast, rather than to equal changes in luminance. This fact suggest that a uniform quantizer applied to a contrast variable would spread contouring effects equally over the range of the variable and therefore allow a more equitable human-quantified bit distribution. The usual relationships proposed between a contrast variable c and the luminance variable x we have been using until now are logarithmic

$$c = a\ \log(1 + bx), \quad \text{for } 0 \leq x \leq 1$$

or power laws[3]

$$c = a\, x^b$$

Finally, it is important to be aware of the use of pseudo-random noise, or dithering, to render contouring effects resulting from smaller analogue-digital converter word-lengths as well as the principles behind binary half-tone processes used to represent grey-scale images. These are not directly related to quantization for transform coding as presently used but can provide a useful insight to the quantization process.

5.4.2 Vector Quantization

Instead of treating each scalar value or sample separately, and quantizing them one at a time, it is possible to reduce the overall quantization error by considering the scalar values N at a time. In this case the N values $\mathbf{x}$ considered as the elements of an $N \times 1$ vector correspond to a point in an N-dimensional vector space. This vector space is partitioned into volumes called *decision regions*, D_i, with one *reconstruction vector* $\mathbf{r}_i$ per decision region. An input vector $\mathbf{x}$ which lies in decision region D_i will be transformed into the quantized vector $\hat{\mathbf{x}} = \mathbf{r}_i$ with mean square quantization error

$$e_q = \sum_{i=0}^{N-1} \int_{D_i} tr[(\mathbf{x} - \mathbf{r}_i)(\mathbf{x} - \mathbf{r}_i)^T]\, P(\mathbf{x})\, d\mathbf{x}$$

For given decision regions D_i, determination of the optimum reconstruction levels $\mathbf{r}_i$ requires a knowledge of the joint probability distribution

$$P(\mathbf{x}) = P(x_1, \cdots, x_N)$$

Apart from the difficulty in evaluating the multi-dimensional integrals involved, sufficient information on the joint PDF may not be available. However, in certain circumstances where compression is at a premium, it may be worth the trouble trying to overcome these problems with various simplifying assumptions.

5.4.3 Processing Quantized Variables

In Figure 5.12 the distinction between the real values of the reconstruction levels used to calculate the quantization error and the corresponding binary code symbols used in most quantization processes to label these reconstruction levels is clearly drawn. For example, in the case of the standard method for quantizing the values of each sample of a grey-level image giving pixel values, the pixel values are usually represented as eight-bit integers in the range 0-255. In many cases we "get away" with considering these labels as arithmetic values, particularly where we are coding data simply for transmission and remote

[3]Note that some imaging devices allow the control of a parameter called *gamma* which controls a point transform of intensity similar to either of the above functional relationships.

re-constitution, but strictly speaking they should be replaced by the corresponding reconstruction levels before any arithmetic operations as only these reconstruction levels bear any arithmetic relationship to the physical quantity being sampled and quantized. For this reason, most image digitizing hardware offers the possibility of a look-up table (LUT) conversion before any computation takes place on the quantized values. In this case the binary or integer labels (such as the 0-255 above) are used as table addresses for memory locations containing the corresponding reconstruction levels, which could even be floating-point values.

5.5 Transform Coding

The basic principles of transform coding are that (i) data values are grouped into blocks which are processed as a unit, (ii) signal energy is concentrated in as few transform coefficients as possible, and (iii) bits are allocated to quantized versions of these transform coefficients so as to minimize the (usually mean square) distortion of the reconstructed data for a given bit quota. The overall scheme is illustrated below.

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \rightarrow [\mathbf{A}] \rightarrow \begin{bmatrix} X_1 \\ \vdots \\ X_N \end{bmatrix} \begin{matrix} \rightarrow \\ \\ \rightarrow \end{matrix} \begin{matrix} Q_0[\cdot] \\ \vdots \\ Q_N[\cdot] \end{matrix} \begin{matrix} \rightarrow \\ \\ \rightarrow \end{matrix} \begin{bmatrix} \tilde{X}_1 \\ \vdots \\ \tilde{X}_N \end{bmatrix} \rightarrow \left[\mathbf{A^{-1}}\right] \rightarrow \begin{bmatrix} x'_1 \\ \vdots \\ x'_N \end{bmatrix}$$

An optimal transform coder is assumed to mean one that minimizes the average mean square error between $\mathbf{x}$ and $\mathbf{x}'$, i.e the transform coding error

$$e_{TC} = \frac{1}{N} E[(\mathbf{x} - \mathbf{x})^T (\mathbf{x} - \mathbf{x}')]$$

where the "mean" in "mean square" is represented here by $E[\cdot]$ and is calculated over all the values that a single input coefficient (e.g. a single pixel position) might have, while the "average" is represented here by the summation of the dot product and by the$1/N$ factor, and is calculated over all the input coefficient positions (e.g. pixel positions) in the data block.

It can be shown that the optimal transform coder uses the KLT for transform and inverse transform and independent Lloyd-Max quantizers for data reduction. In practice, however, compromises may be made on either or both of these elements for speed or complexity reasons. On the transform end, there are well-defined circumstances in which the KL transform may be replaced by any of the discrete transforms described above. In terms of the quantization process, the design of a transform codec will require us to deal with different statistical distributions for various coefficients, differing coefficient variances and suitable algorithms for the allocation of bits among the coefficients.

5.5.1 Bit Allocation Algorithms

The input parameter for the design of a quantizer for each transform coefficient is the number of bits available to code that coefficient, given its variance relative to the variance of all the other coefficients and given the total number of bits available, assuming that we are trying to minimize distortion. With respect to the transform coefficients X_i and their quantized counterparts $\tilde{X}_i$, the average distortion, or error introduced by quantization, is given by

$$e_q = \frac{1}{N} \sum_{i=0}^{N-1} E[(X_i - \tilde{X}_i)^2] = \frac{1}{N} \sum_{i=0}^{N-1} \sigma_i^2 \, f(n_i)$$

where σ_i^2 is the variance of the transform coefficient X_i, where $f(\cdot)$ is the quantizer rate distortion function (e.g. $f(n) = a\, 2^{-b\,n}$, $b \leq 2$) and where n_i the number of bits allocated to coefficient X_i. The distortion introduced by quantization to each individual coefficient is the product of the variance and the rate-distortion function: $D_i(n_i) = \sigma_i^2 \, f(n_i)$. If n_{av} is the average number of bits allocated per sample, then we should also have

$$n_{av} = \frac{1}{N} \sum_{i=0}^{N-1} n_i$$

The task is to find the bit allocation values n_i that minimize the average distortion subject to the constraint of the average bit rate just given.

For simplicity, consider the case of just two transform coefficients X_0 and X_1 that are considered to be samples of stationary random processes with zero mean and variances σ_0^2 and σ_1^2 respectively. (See Jayant & Noll, p.526). Then the average distortion is

$$D_{av} = \frac{1}{2}\left[D_0(n_0) + D_1(n_1)\right] = \frac{1}{2}[\sigma_0^2\, f(n_0) + \sigma_1^2\, f(n_1)]$$

where the average bitrate n_{av} is

$$n_{av} = \frac{1}{2}[n_0 + n_1]$$

It is possible to find the optimal division of bits between the two coefficients by writing $n_1 = 2n_{av} - n_0$, substituting this in $D_{av} = \frac{1}{2}\left[D_0(n_0) + D_1(n_1)\right]$ and setting

$$\frac{d\, D_{av}}{d\, n_0} = 0.$$

The result is that

$$n_0 = n_{av} + \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}, \quad n_1 = n_{av} - \frac{1}{b}\log_2\frac{\sigma_0}{\sigma_1}$$

giving an minimum average distortion subject to n_{av} of

$$D_{av\,,min}(n_{av}) = \sigma_0\,\sigma_1\, f(n_{av})$$

The advantage of transform coding is that it concentrates signal energy in some coefficients (giving them larger variance) at the expense of others. As just outlined, we can then match the numbers of bits allocated per coefficient to the variance of the coefficient, hopefully producing a more efficient representation of the signal energy. We can see just how much more efficient by comparing the distortion thus introduced for a given bitrate with what we would have if we simply used the same number of bits for each transformed coefficient, (or equivalently for each un-transformed sample) as is the case in pulse-coded modulation (PCM). For coefficients produced by an orthogonal transform, the average variance of the coefficients $\frac{1}{2}[\sigma_0^2 + \sigma_1^2]$ is the same as the variance of the input σ_X^2. If the number of bits per sample n is set to the same value B_{av} as used in the transform coding case above, the average distortion introduced in the PCM case will be $D_{av\,,PCM}(n_{av}) = \sigma_x^2\, f(n_{av}) = \frac{1}{2}[\sigma_0^2 + \sigma_1^2]f(n_{av})$. The advantage or gain derived from transform coding over straightforward PCM is thus given by

$$\frac{D_{av\,,PCM}}{D_{av\,,min}} = \frac{\frac{1}{2}[\sigma_0^2 + \sigma_1^2]}{\sigma_0\,\sigma_1} = \frac{\frac{1}{2}[\sigma_0^2 + \sigma_1^2]}{[\sigma_0^2\,\sigma_1^2]^{\frac{1}{2}}}$$

This is concisely expressed as *the ratio of the arithmetic mean to the geometric mean of the coefficient variances* and is an important general result for transform coding theory. It says that, at worst, transform coding will be as good as PCM, (where the sample variances are assumed to be uniform) with the transform coding gain over PCM improving as the coefficient variances deviate further from uniformity.

In the general case of N transform coefficients we have to use the technique of Lagrange multipliers to find the minimum distortion subject to the constraint of a given average bitrate, but the result is just a generalisation of the simple $N = 2$ case considered. The optimal bit allocation is given by

$$n_i = n_{av} + \frac{1}{b}\log_2\frac{\sigma_i^2}{\left[\prod_{j=0}^{N-1}\sigma_j^2\right]^{1/N}}$$

Another result worth noticing for the optimal bit allocation in a transform coder is that all the quantizers give equal distortion. For example, in the case $N = 2$, the optimal values of n_0 and n_1 give

$$D_0(n_0) = D_1(n_1) = D_{av\,,min}(n_{av}) = \sigma_0\,\sigma_1\, f(n_{av})$$

and in the general case

$$D_i(n_i) = D_{av\,,min}(n_{av}) = \left[\prod_{j=0}^{N-1}\sigma_j^2\right]^{1/N} f(n_{av})$$

This has to be the case as otherwise it would be possible to "steal" bits from a lower distortion coefficient and give them to a higher distortion coefficient thus lowering the average distortion for the same average bitrate and contradicting the assumption of optimality.

We show in the discussion on quantizer design above that the signal to noise ratio for a uniform n-bit quantizer and a uniformly distributed input is given by 2^{2n}, which corresponds to a distortion of $\sigma_X^2\, 2^{-2n}$. In the more general case of uniform quantizers operating without overload on non-uniformly distributed coefficient values, the distortion introduced by quantizing each coefficient is $\sigma_i^2\, a\, 2^{-b\, n_i}$, $b \leq 2$. In the case of multiple coefficients, each with its own quantizer configured so that the bits are optimally distributed across the coefficients, we find that the above quantization distortion is actually the same for all the coefficients and thus independent of i, even though σ_i^2 and n_i both vary as i ranges across the coefficients. It is worthwhile understanding clearly how the quantizer design for a coefficient is related to the coefficient's variance.

Even though the probability distribution for a particular coefficient's values is allowed to be non-uniform, assume that it is approximately uniform over a single decision band of a uniform quantizer with step size $\Delta = \frac{A}{2^n}$. Than, as before, we can show that the quantization error variance is $\frac{\Delta^2}{12}$, which from the definition of Δ equals $\frac{A^2}{12}\, 2^{-2n}$. Now the quantization error variance is also the distortion produced by the quantization, which is given by $\sigma_i^2\, a\, 2^{-b\, n_i}$. We already know that in the optimal case, this is invariant across coefficients, hence in the optimal case, *the quantizers of each coefficient have the same step size* Δ. For $b \approx 2$, we can also see that if one coefficient has twice the range of another, its variance σ_i^2 is a factor of four larger and we need double the number of levels in the quantizer, which corresponds to one extra bit, all the while keeping the step size fixed. The fixed step size and the variation of the number of quantizer steps with coefficient range (or variance) are illustrated in Figure 5.16.

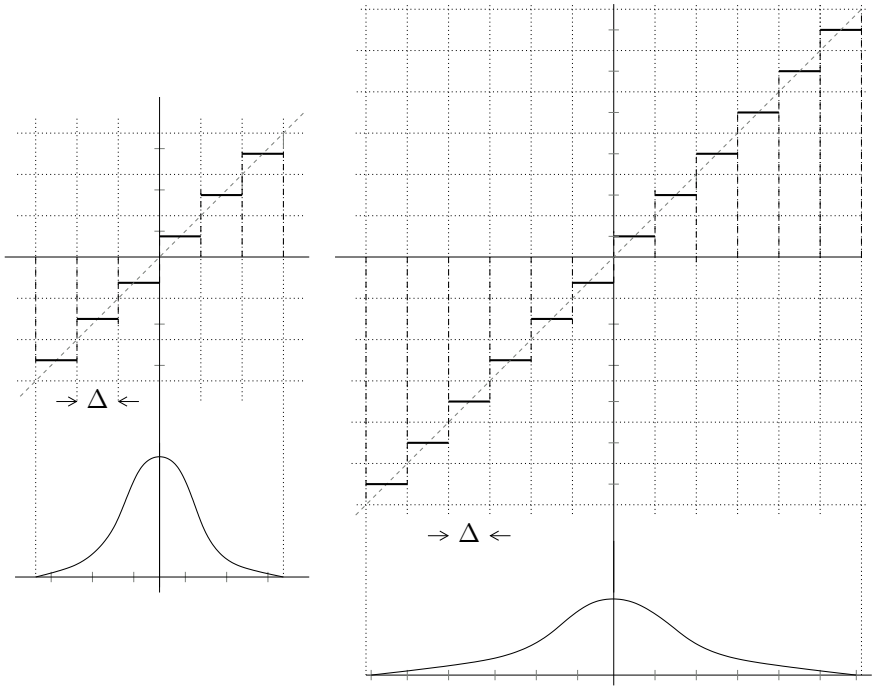


Figure 5.16: Illustration of two transform coefficient distributions along with their corresponding optimal bitrate quantizers. The step sizes Δ are equal to ensure identical quantization error variances, while extra quantization steps and thus extra bits are allocated to cope with the coefficient with the larger dynamic range.

Practical Bit Allocation

The scheme for optimal bit allocation discussed above may give values for the number of bits to be allocated to particular transform coefficients that are non-integers or even negative. Bitrates with fractional

parts can either be rounded up to an integer value, or achieved by using a quantizer with more than the required number of levels and entropy coding the output. In this latter case, only the average of a variable length code needs to equal the target bitrate.

When the variance of a coefficient is significantly smaller then the geometric mean of all the coefficient variances, the optimal solution may allocate a negative "number" of bits n_i. These negative values of n_i will need to be set to zero and the values of bitrate allocated to other coefficients adjusted accordingly (i.e. lowered) to maintain the target average bitrate n_{av}. In addition, the overhead of coding coefficients allocated small bitrates may mean that it is more efficient to set some of these to zero also and re-allocate the bits saved to other coefficients with larger bitrates. Finally, some coefficients with very large variance may be allocated bitrates which exceed the perceptual requirements of the coding system, so these values of n_i which exceed some maximum value n_{max} can be trimmed back and the excess reallocated elsewhere to decrease the distortion of other coefficients. These three different factors mean that a procedure is required to allocate actual bitrates to individual coefficients, beyond that prescribed by the optimal bitrate allocation process derived above.

If fractional bitrate assignments are to be allowed, the total excess bitrate from the combination of coefficients being set to zero and from coefficients which have greater than required accuracy, can simply be divided evenly across all coefficients where n_i is below n_{max}. If this brings some more coefficients above n_{max}, this new surplus can be reallocated in turn, and so on. The objective will be to maintain identical quantizer step sizes for all non-thresholded coefficients with rates below n_{max}. The minimum achievable distortion given by

$$D = \frac{1}{N}\left[\begin{array}{ll} \sum\limits_{\substack{i=0,\\ \sigma_i^2 \geq \theta}}^{N-1} & \sigma_i^2\, f(n_i) + \sum\limits_{\substack{i=0,\\ \sigma_i^2 < \theta}}^{N-1} \sigma_i^2 \end{array}\right]$$

In the case where the number of bits n_i to be allocated to the quantizer for each coefficient must be an integer, the following straightforward procedure will do the trick.

Firstly, set all the n_i to zero. Then determine for which of the coefficients, an additional bit will make the greatest difference. This means evaluating

$$\sigma_i^2\,[f(n_i) - f(n_i + 1)]$$

for each of the coefficients, given the current values of each n_i. The value of i that has the largest value of this distortion decrement has its n_i incremented by one and its distortion decrement recalculated. The distortion decrement of the other coefficients will not have changed, so we search for the next largest distortion decrement amongst these and the newly recalculated one. The process repeats until all available bits $N\,n_{av}$ have been allocated.

If there is a tie for the largest distortion decrement at a point in the allocation, we will need to try each of the possible progression paths that the algorithm might take if we were to allocate this bit to just one of the tie-ing coefficients.

When the distortion-rate function is of the form $f(n) = a\,2^{-b\,n}$, the distortion increment will be $(1 - 2^{-b})\,\sigma_i^2\,f(n_i)$, so the quantizer with the largest distortion decrement will be the one with the largest distortion.

5.5.2 Transform Coding of Images

All of the 1-D transform coding results carry over to 2-D directly if we consider an $M \times N$ image as a $MN \times 1$ data vector. However, it is more usual to divide an image into smaller 2-D blocks of size 4×4, 8×8 or 16×16 pixels and apply 1-D transforms to these as separable 2-D transforms as described above. These sizes represent various trade offs of hardware/algorithmic complexity and coding efficiency, with the 8×8 block size being the most common.

Quantizer Design

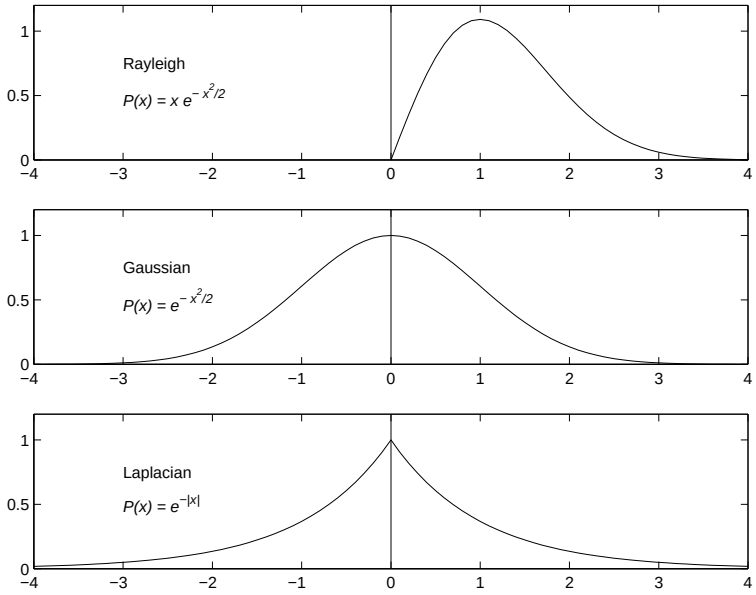


Figure 5.17: Plot of the Rayleigh, Gaussian and Laplacian functions which are common models for statistical distributions.

For images, the dc coefficient is non-negative and is often modeled by a Rayleigh distribution, or by one-sided Gaussian or Laplacian distributions. The remaining coefficients usually have zero mean and are often modeled by symmetric distributions such as Gaussian or Laplace. The shape and functional form of these distributions is shown in Figure 5.17. Typically, anything from 8 bits to zero bits may be allocated to each coefficient, so one quantizer will be needed for the dc coefficient and up to eight different quantizers for the remaining coefficients.

Zonal Coding

Because the transform coefficients for a particular unitary transform may often be divided into groups which share broadly similar characteristics, these groups or zones with similar quantization requirements may be permanently fixed as a feature of the transform coder. An example is shown in Figure 5.18. Sometimes the term zonal coding is used in the context of the definition of a *zonal mask*, which is a region of the block where the coefficients have sufficiently small *variance* that they are always set to zero and not processed for quantization.

Threshold Coding

Threshold coding is closely related to zonal coding. The difference is that decisions about setting particular coefficients to zero are made on the basis of the small *amplitude* for each instance of a transformed block, rather than on small variance which is calculated over the ensemble of blocks in the population. An example of the particular threshold decisions that might be made on the basis of coefficient amplitude for one particular instance of a transform coded block is shown in Figure 5.19.

Zig-zag Scanning

Even though the transform coefficients of an 8×8 DCT are relatively uncorrelated, they are still statistically related. For example, if horizontal frequencies are present in an image, then diagonal or vertical signal energy at the same frequency is also likely to be present. It thus makes sense to keep the coefficients which are statistically related to each other close together when they are being processed subsequently. In particular, run-level coding works much more efficiently if there are long runs of zeros to be coded. This is the case, for example, if a large number of high frequency coefficients have relatively low values and get thresholded to zero. Now, because the zig-zag sequence keeps these together, they are not going

8	8	6	6	6	4	4	4
8	6	6	6	4	4	4	2
6	6	6	4	4	4	2	2
6	6	4	4	4	2	2	2
6	4	4	4	2	2	2	0
4	4	4	2	2	2	0	0
4	4	2	2	2	0	0	0
4	2	2	2	0	0	0	0

Figure 5.18: Example of fixed zones in an 8×8 transform coder where the number of bits allocated to quantize each coefficient is shown.

1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	0
1	1	1	0	0	1	1	0
1	1	1	1	0	0	0	0
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0

Figure 5.19: Example of a threshold mask where a 1 indicates that the amplitude of the corresponding coefficient was sufficiently large in this instance to be processed further, while a 0 indicates that that coefficient was below threshold and will not be processed further.

to be interrupted by a large-amplitude low-frequency coefficient, as would be the case with a horizontal raster-scan. A zig-zag scan pattern typical of that found in most DCT-based transform coders is shown in Figure 5.20.

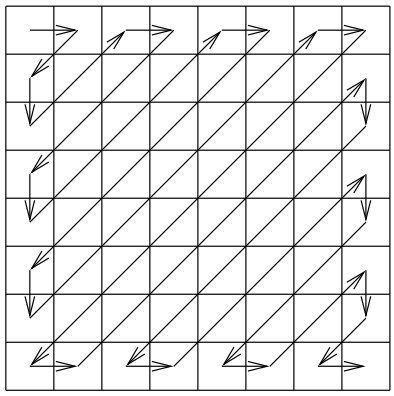


Figure 5.20: Illustration of the "zig-zag" sequence in which DCT coefficients are processed in order to keep coefficients with similar statistical properties together.

Chapter 6

The JPEG Image Compression Standard

6.1 Introduction

JPEG, standing for Joint Photographic Experts Group, is the short title for an image coding standard which was finalised in the early 1990s. It was intended to be a flexible generic compression standard for continuous-tone still images, both greyscale and colour, with state of the art compression but without excessive computational complexity. What resulted should be regarded as a toolkit that cab span a wide range of continuous-tone image applications.

6.2 Overview of JPEG

JPEG has four modes of operation:

Sequential encoding : each image component is encoded in a single left-to-right, top-to-bottom scan. Encoding can begin before the full image is available and encoded blocks can be transmitted before the end of the image has been encoded;

Progressive encoding : the image is encoded in multiple scans for applications where the transmission time is long and the viewer wants to watch the image build up in multiple coarse-to-fine passes;

Lossless encoding : the image is encoded to guarantee exact recovery of the original image pixel values;

Hierarchical encoding : the image is encoded in multiple resolutions so that lower resolution versions can be accessed without having to decompress the full image resolution.

6.2.1 DCT-based sequential-mode codecs

Schematic diagrams of the stages in the encoding and decoding stages of DCT-based sequential mode codecs are shown in Figures 6.1 and 6.2. The detail of the various steps in the baseline JPEG encoding process is described next. The main algorithmic steps are bulleted and these are interspersed with explanations. (Remember only the decoding process is actually defined by the JPEG standard).

- Source image pixels are grouped into 8×8 non-overlapping blocks. If the dimensions of the image are not multiples of eight, new rows or columns are added as duplicates of the last row or column respectively.

- Pixel values are converted from unsigned integers in the range $[0,\ 2^P - 1]$ to signed integers in the range $[-2^{P-1},\ 2^{P-1} - 1]$.

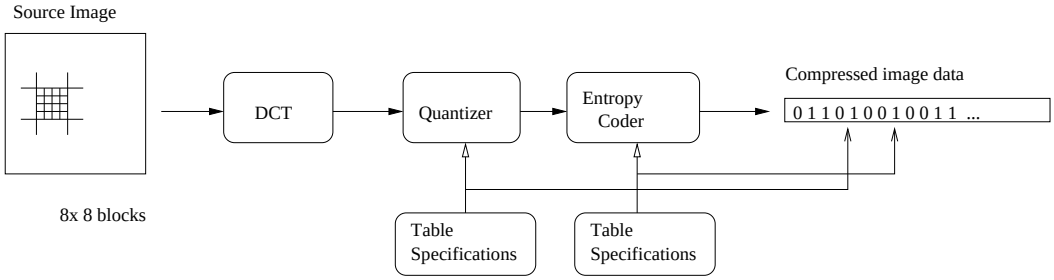


Figure 6.1: Diagram of the processing steps in the encoding stage of the single-component (greyscale) DCT-based JPEG standard sequential modes .

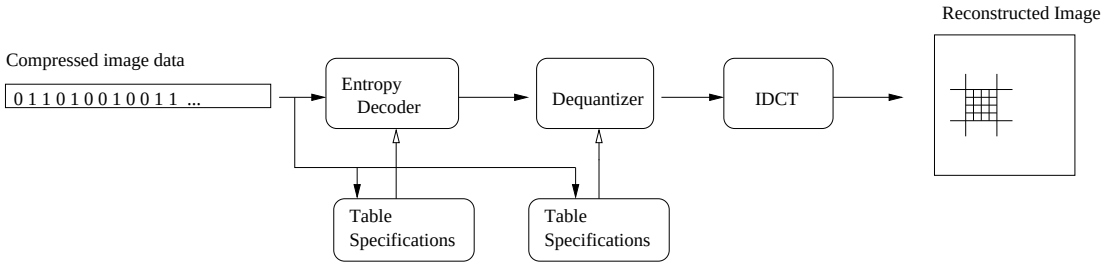


Figure 6.2: Diagram of the processing steps in the decoding stage of the single-component (greyscale) DCT-based JPEG standard sequential modes .

- The 8×8 DCT is carried out to give the effect of the following formula:

$$F(0,0) = \frac{1}{8} \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j)$$

$$F(m,n) = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos \frac{(2i+1)m\pi}{16} \cos \frac{(2j+1)n\pi}{16}$$

  for $0 \leq m, n \leq 7$ and one of $m, n \neq 0$.

The DCT coefficients represent the relative amounts of combinations of horizontal and vertical spatial frequencies found in the given 8×8 block. The coefficient corresponding to zero spatial frequency in both directions is called the "DC coefficient" and is effectively the average over all the 64 pixel values in a block. The remaining coefficients which all represent some type of spatial change over the block are called "AC coefficients". The JPEG standard does not specify a unique DCT *algorithm* for implementing the above transform, but rather an *accuracy test* as part of the compliance testing of the decoder. However, the JPEG standard does specify that different codecs be used for 8-bit versus 12-bit input pixel resolution, so that the extra computational resources required for the 12-bit applications do not reduce the efficiency when sample resolutions of only 8-bit are required.

- Each of the 64 DCT coefficients is *uniformly* quantized using a user-specified 64-element Quantization Table. The elements of the Quantization Table are integers in the range 1 to 255 which are used to specify the step size of the quantizer. The quantization operation is implemented as the *division* of each DCT coefficient by the corresponding element in the quantization table and rounding to the nearest integer.

The quantizer is the coding step where compression is achieved by a deliberately lossy means: information which is assumed to be less visually significant is thrown away, up to a level consistent with a specified image quality.

The quantization tables are not specified in the standard and so need to be based on the design criteria of each particular implementation. The two main methods for designing quantization tables are psychophysical and rate-distortion.

Psychophysical approach for generating quantization tables

The psychophysical approach is based on the fact that the human ability to notice changes depends on many factors, but in particular is strongly dependent on the frequency of those changes. It is more difficult to see identical magnitude changes in very high spatial frequencies and very low spatial frequencies than it is in mid spatial frequencies. Lohscheller applied this to DCT basis functions and showed that the human visual system has different thresholds for just noticing a low amplitude DCT basis function against a plain background, for different DCT basis functions. The following two quantization tables, one for luminance images and the second for colour images are, when used in a Q, Q^{-1} pair, intended to give changes in the corresponding components which are equivalently noticeable by the human visual system.

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

Lohscheller quantization table for luminance images.

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Lohscheller quantization table for chrominance component images.

By multiplying these quantization tables by a *quality factor* it is possible to generate quantization tables that uniformly increase or decrease the quality depending on whether better quality or better compression is required. Remember these quantization tables correspond to just noticeable changes. If a quality factor of less than one is used, just-noticeable changes will go below the threshold of noticability and will not be visible. These quantization tables are given in an annex to the JPEG standard, but their use is not required by the standard.

Rate-distortion approach for generating quantization tables

Let r be the desired average bit rate before the the entropy coding stage. If b_{ij} bits are allocated to each quantized DCT coefficient, then

$$\frac{1}{64}\sum_{i=1}^{8}\sum_{j=1}^{8} b_{ij} = r$$

If there are N blocks of 8×8 pixels in the image, let C_{ik}^{k} be the ij^{th} coefficient of the k^{th} block (k is an index, not a power), let μ_{ij} be the mean value of the coefficients C_{ik}^{k} over all N blocks and let σ_{ij}^{2} be

the variance of the coefficients C_{ik}^k over all N blocks. We want to allocate more bits to the coefficients with larger variances, so the optimum bit allocation is given by

$$b_{ij} = r + \frac{1}{2}\log_2 \frac{\sigma_{ij}^2}{\left[\prod_{ij} \sigma_{ij}^2\right]^{1/64}}$$

For 8bits/pixel images, the AC coefficient range is $[-1023, 1023]$. For uniform quantization with b_{ij} bits, this range will be divided into $2^{b_{ij}}$ levels, so the elements of the quantization matrix will be

$$Q_{ij} = \frac{2046}{2^{b_{ij}}}$$

6.3 Entropy Coding

The entropy coding block of JPEG (see Figure 6.1) has the task of source coding the quantized DCT coefficients. DC and AC coefficients are treated separately since they have very different characteristics. Either Huffman coding or arithmetic coding can be used for the source coding. In the case of Huffman coding, one or more (up to a maximum of eight) tables of codewords must be specified by the application. There are no official default tables but most applications use the tables recommended in an annex of the standard. There are no restrictions imposed on the tables actually used except that (i) no codeword may be longer than 16 bits, (ii) no codeword may be all 1's (i.e. $0xFF$).

In this section only the Huffman coding approach is described, since this is the only method of source coding supported in the baseline JPEG implementation. In the baseline implementation, only four different Huffman tables may be used – two for the DC coefficients (luminance and chrominance) and two for the AC coefficients (luminance and chrominance).

Arithmetic coding would give better compression that Huffman coding, but many JPEG implementations do not use it. This is partially because it is not required in the baseline implementation, but also because the gain of 2% to 10% is not very significant and arithmetic coding algorithms are covered by Patents in the US and Japan.

6.3.1 DC Coefficients

The DC coefficients of successive blocks are predictively encoded:

- The DC coefficient of the DCT has the value of the DC coefficient of the previous 8×8 block subtracted from it and only the difference is retained for entropy encoding.

Because the DC coefficient represents the average pixel value in each 8×8 block, there tends to be a strong correlation between the DC coefficients of consecutive blocks (i.e. 8×8 blocks in a particular part of an image will have average grey levels in and around the same value). This means that the value of the difference between two consecutive DC coefficients is likely to have much smaller entropy than the original DC coefficients themselves. Consequently a coding advantage will accrue from using the DC difference value instead of the original DC value[1]. This process is illustrated in Figure 6.3 where the prediction residual (i.e. the actual coding event) is calculated as $DC_{diff} = DC_i - DC_{i+1}$, where DC_i and DC_{i-1} are the DC coefficients of the current block ($Block_i$) and the previous block ($Block_{i-1}$) respectively.

- The differences DC_{diff} are entropy encoded, usually with a Huffmann procedure as described below.

A process very similar to the approach used in the lossless mode of the JPEG standard is used to encode DC_{diff} (see section 3.4.3). Assuming eight bit pixel values are used, DC_{diff} can take values in the range $[-2047, +2047]$. If each of these values was to be Huffman coded, a table of 4095 entries would be required! Consequently DC_{diff} is coded as a $(length, magnitude)$ pair. The $length$ indicates the

[1] There is also a statistical dependence between the AC coefficients of one block and the next, but it is usually not as significant as the DC correlation. The JPEG standard does not attempt to exploit this correlation of AC coefficients.

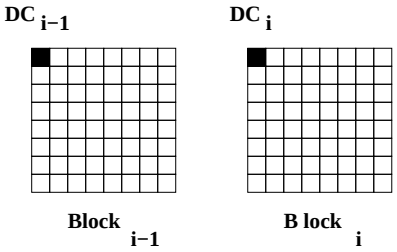


Figure 6.3: Predictive coding of DC DCT coefficients

number of bits used to encode the magnitude of DC_{diff}. A static Huffman code is used for this value[2]. The *magnitude* is the actual value of DC_{diff} directly encoded. Negative values are encoded as the ones complement value of $|DC_i - DC_{i-1}|$ so that negative residuals always start with a zero bit.

6.3.2 AC Coefficients

The AC coefficients in a block are transformed into a new set of encoding events using run-length encoding prior to Huffman coding. For 8bit/pixel images, AC coefficients may take any value in the range $[-1023, 1023]$. This range is divided into ten size categories where the i^{th} category includes all coefficients that can be represented by i bits. Each coefficient can thus be described by the $(length, magnitude)$ pair as before.

- The quantized coefficients are first read out as a one-dimensional sequence in the "zig-zag" fashion illustrated in Figure 6.4.

As explained in the transform chapter, coefficients representing similar spatial frequency tend to be statistically related. For example, if a particular 8×8 block of an image, contains no high horizontal spatial frequency it is quite likely that coefficients corresponding to the same spatial frequency arranged diagonally or vertically will also be small or zero. Coupled with strong quantization applied to the higher spatial frequency coefficients, the zig-zag scanning may result in long runs of zeros in its output stream. This is useful because the next step involves encoding runs of zeros as a single unit and the fewer of these there are, the more efficient the coding.

- Each non-zero AC coefficient is encoded as a $(run/length, magnitude)$ combination where *length* is the number of bits required to encode the *magnitude* of the coefficient, and *run* is the number of zero coefficients prior to the current non-zero one.

- The value of *run/length* is then Huffman coded, and the value of *magnitude* is appended directly to that code. Negative values are encoded in ones complement form.

For example, suppose an AC coefficient with the value -18 is preceded by six zeros. -18 falls into category 5 and the ones complement of -18 is 01101, so this event is represented by $(6/5, 01101)$. If the Huffman code for the pair (6/5) is 1101, the overall code for this event is 110101101.

If there are more than 15 zero coefficients in a run, the special event (15/0) is used to denote a run of 15 zeros followed by a zero coefficient amplitude. The codeword for the last AC coefficient event must have a non-zero magnitude as the special symbol $(0, 0)$ is used to denote end-of-block.

The primary source of data compression in the JPEG scheme is the losses introduced by the quantization scheme. However, the relatively convoluted algorithm for entropy coding (DC prediction/Huffman, AC zig-zag/run-amplitude/length-magnitude/run-length-Huffman) is shown to be a very good compromise between entropy coding efficiency and the large complexity of full Huffman coding. In this way, any quality which is forfeited in the interests of compression can be seen to pay off very directly in a reduction in bit rate.

[2]The range $[-2047, +2047]$ is divided into 12 length categories where the i^{th} category contains each value of DC_{diff} which can be represented with i bits. In this way, only 12 Huffman codes are required in order to encode the length.

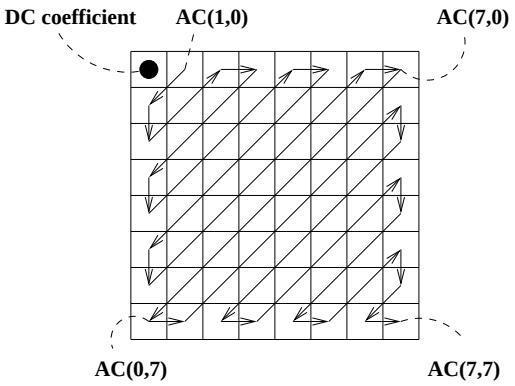


Figure 6.4: Predictive coding of AC DCT coefficients

6.3.3 JPEG coding of colour images

If there are multiple colour components for each pixel the JPEG scheme is to consider a particular colour component for each pixel as a separate rectangular array of component values and to code it just as a monochrome image. Other colour components are subsequently processed in the same way. This is a very straightforward way of extending a monochrome scheme to deal with multiple colour components, but it does not exploit the redundancy that may exist *between* the different colour components. Either 8 or 12 bits will be used for each colour component, just as in the monochrome case. The treatment of each colour component effectively as a separate image extends to allowing the different component images to be coded at different spatial resolutions, so that the component images do not all need to have the same dimensions. However, subsampling ratios in the range 1 to 4 only are allowed.

RGB components in particular will typically have substantial redundancy between the different components. Transforming to YUV substantially reduces the inter-component correlation and allows for the possibility of subsampling the U and V components. While the subsampling is supported by JPEG, the RGB-¿YUV transformation is not a part of the standard and would be considered a preprocessing/postprocessing step implemented by a particular application.

6.4 Non-baseline JPEG coding modes

6.4.1 Progressive Coding

In the JPEG progressive coding mode, the image is encoded in multiple scans. The mode works by buffering the output of the DCT so that during each scan only some of the DCT information is transmitted. The two procedures for selecting what information is to be transmitted are *spectral selection* and *successive approximation*.

In the case of *spectral selection*, the DCT coefficient blocks are divided into frequency bands and one frequency band is transmitted in each scan. For example, given the zig-zag scanning of the DCT coefficients, a typical arrangement is as follows:

Scan 1 DC coefficient and first two AC coefficients

Scan 2 Next seven AC coefficients

Scan 3 Next eleven AC coefficients

Scan 4 Remaining 45 AC coefficients

In the case of *successive approximation*, the most significant bits of the DCT coefficients are transmitted in the earlier scans, with the less significant bits following later. If necessary, the spectral and successive approximation procedures can be combined.

6.4.2 Hierarchical Coding

In the JPEG Hierarchical mode, the image is encoded as a sequence of frames each with progressively higher resolutions. In this way that lower resolution versions can be accessed without having to decompress the full image resolution. The first frame transmitted is usually a low-pass filtered, down-sampled version of the original image. The next frame has this low-resolution image subtracted from it (actually the low-pass filtered version before decimation) and the residual is what is encoded and transmitted. There may be several stages in the hierarchy which are calculated in the same way.

Either lossless or lossy coding can be used for the coding of individual frames in the hierarchical mode. The mode is intended to give advantages in usage rather than in compression.

6.4.3 Motion JPEG

While JPEG was developed for compressing still images, it has also been used as a crude way of coding digital video in a form called *motion-JPEG*. The idea is simply that each frame in a video sequence is coded as a single JPEG image using the baseline JPEG mode. The effect is roughly equivalent to MPEG-1 encoding where all of the frames are forced to be I-frames (see below). This was not a mode described in the original JPEG standard so the bit-stream syntax is likely vary from implementation to implementation.

Chapter 7

Fundamentals of Video Compression

7.1 Introduction

In this chapter we introduce the fundamental concepts underpinning video compression. Initially, the concept of temporal redundancy in the context of video sequences is explained. Two independent yet related processes which exploit this redundancy called motion estimation and motion compensation are then introduced and explained.

7.2 Redundancy in Video Sequences

As explained in Chapter 2, Section 2.2, digital video consists of a sequence of *frames*, corresponding to individual (colour) images of a scene scanned at successive time intervals, where the interval is determined by the *framerate* of the video. Considering video as a sequence of still images, our first attempt to compress the video data might be to simply apply the DCT-based still image compression techiques discussed in previous chapters, to each frame in the sequence [1]. In this way, we could exploit the *spatial* and *perceptual* redundancy present in each image. However, when applying such an approach in real applications, we would quickly discover that it would not produce the high levels of compression efficiency required by very low bitrate appplications. Luckily, however, there is another type of redundancy present in video sequences, called *temporal* redundancy, which can be exploited in order to achieve the demanding compression ratios required. The goal of video compression is therefore to exploit spatial, temporal and perceptual redundancy.

At high framerates, the individual frames of a video sequence differ very little from each other - even for dynamic scenes featuring significant motion/activity. For example, two images of the same scene captured only 40ms apart (i.e. at a framerate of 25 Hz) will be very similar. The approach used in video compression is to only code the *difference* between successive images, since this data has lower entropy than the image data itself. This is termed *inter-frame* coding. This concept is illustrated in Figure 7.1 which depicts two consecutive frames from a 25 Hz version of the Foreman test sequence (Figures 7.1(a) and Figure 7.1(b) are the first and second frames of the sequence respectively). Figure 7.1(c) depicts the difference between these two frames. Since most pixel values in this image are in and around 0, the image appears to be almost completely black and as such is not is not very easily interpreted by a human observer[2]. For this reason, in Figure 7.1(d), the pixel values in the difference image have been scaled to produce a viewable version of the image[3]. The entropy can be estimated for Figures 7.1(a)7.1(b)7.1(c). As can be seen from Figure 7.1 the entropy of the difference image is much lower than the entropy of the original images.

In actual fact, usually a prediction of the current image to be encoded is first formed, and the *prediction residual* is encoded. Assuming a good prediction, the residual will have even lower entropy

[1] This is exactly the approach taken in motion-JPEG – an extension of the JPEG standard which handles video by storing each frame as a JPEG image

[2] For example, due to 8 bit pixel underflow, negative pixel values appear as bright regions

[3] The scaling used is: $p_{view} = p_{diff} \times 2 + 128$, where p_{view} and p_{diff} are corresponding pixel locations in the viewable and difference images

(a) Frame 0

(b) Frame 1

(c) Difference image

(d) Viewable difference image

Figure 7.1: Illustration of temporal redundancy in video sequences using the Foreman test sequence: 7.1(a) and 7.1(b) entropy $\approx$ 7.15 bits/pixel, 7.1(c) entropy $\approx$ 4.38 bits/pixel

than a raw difference between successive images. Coding the prediction residual exploits the temporal redundancy present in the video sequence.

The prediction residual itself can be thought of as an image, and indeed an image with significant spatial correlation. For example, if we assume a good prediction of the current image to be encoded is available, then the prediction residual will consist of large groups of zero (or low valued) pixels. The *intraframe* DCT-based compression techniques described in previous chapters can be effectively applied to compress the prediction residual. In this way, the process of video compression can be viewed as a combination of both interframe coding exploiting temporal redundancy followed by intraframe coding exploiting the spatial and perceptual redundancy of the result. This combination of intraframe and interframe coding is usually referred to as *hybrid* coding.

In order to initialise the coding process, the first frame in a sequence must be coded in intraframe mode. Typically, intraframe coding is applied periodically after the initial image in order to "reset" the coding process to avoid error propagation and to provide random access points into the video bitstream.

The process of forming the prediction of a current frame involves two separate yet related processes known as *motion estimation* and *motion compensation*. In the following sections, these processes are explained in more detail.

7.3 Motion Estimation

In order to form a prediction of the current frame to be encoded, it is necessary to select a *reference* frame on which to base the prediction. The reference frame used is typically the previous frame in the video sequence, although some video compression standards (e.g. MPEG-1, MPEG-2 & MPEG-4) allow for predictions based on future frames[4].

The difference between the reference and current frame is assumed to be solely due to the presence of motion in the video scene. This scene motion could be as a result of camera motion (e.g. pan, zoom,

[4]In actual fact, the reference frame is the *reconstructed* (i.e. decoded) version of the previous (or future) frame – see Section 7.4

tilt, etc.) when the scene was being filmed, or due to the movement of objects present in the scene (e.g. a moving car, a football player, etc.). The approach taken is to estimate the motion present between the reference and the current and to subsequently "undo" this estimated motion in order to create a prediction.

One approach to estimate the motion between two frames would be to track the movement of each pixel in the image between the frames. However, a process such as this is very computationally expensive and in fact may produce unreliable results due to the presence of noise. A less computationally intensive approach is to estimate the motion of groups of pixels. The major video compression standards divide the current image into 16×16 blocks [5] of pixels and estimate the motion for each block separately.

In estimating the motion of a particular block, what is actually sought is the best match for the block in the reference frame. For this reason, this approach to motion estimation is sometimes called *block matching*. The current block is moved around its coordinates in the reference frame and at each point visited the difference between it and the block under consideration in the reference frame is calculated using a matching criterion. The matching process is restricted to a particular *search area* within the reference frame. The search area should be large enough to capture the motion present. Thus, very active video sequences, such as television sports broadcasts, require a larger search area than less active video content, such as a "talking head" video conferencing sequences. Block-based motion estimation is illustrated in Figure 7.2 where the search area is of size $\pm S$ pixels in both the horizontal and vertical directions relative to the upper left corner of the block.

Clearly, such an approach assumes that the motion in the scene is purely translational in nature. This is not true in general but the resulting simplification is again a good trade off between prediction accuracy and computational complexity. Once the best match is found, the displacement between the current block coordinates and the coordinates of the best match is found. This displacement is termed a *motion vector*. In Figure 7.2 the motion vector is denoted (V_x, V_y), indicating that is representated by two components – a displacement in the horizontal direction (V_x) and a displacmen in the vertical direction (V_y)[6].

In the context of a coding scheme, the individual horizontal and vertical components of the motion vector of each block undergo separate entropy encoding and are transmitted to the decoder. In the decoder, the motion vector is first decoded by reversing the entropy encoding, and subsequently used to construct a prediction for the associated block (see section 7.4). The prediction is also formed at the encoder and the prediction residual between the predicted and current block is encoded using the DCT-based approach as described in previous chapters.

Given a motion vector for every block, and hence every location (x, y), in an image, the prediction residual $E(x, y)$ is defined as:

$$E(x, y) = I_{curr}(x, y) - I_{ref}(x + V_x, y + Vy)$$

where $I_{curr}(x, y)$ is the current image and $I_{ref}(x + V_x, y + Vy)$ is the motion compensated prediction of $I_{curr}(x, y)$ (see section 7.4).

7.3.1 Matching Criteria

There are a number of different possible matching criterion which can be used when calculating the difference between two blocks. Perhaps the most popoular is termed the Mean Absolute Difference (MAD), which for 16×16 blocks is given by the formula:

$$MAD(B_{curr}, B_{ref}) = \frac{1}{16 \times 16} \sum_{i=1}^{16} \sum_{j=1}^{16} |B_{curr}(i, j) - B_{ref}(i, j)|$$

where, B_{curr} is the block in the current frame and B_{ref} is the block of pixels at the current search location in the reference frame. Another commonly used matching criterion is the Sum of Absolute Differences (SAD), given by:

[5] Performing motion estimation in a block-based manner implicitly imposes an (artificial) smoothness constraint on the motion present in a scene. Small block sizes are preferable in order to better fulfill this smoothness constraint. On the other hand, computationally efficient estimation algorithms work better on larger block sizes. 16×16 blocks were chosen because experimental results indicated a good tradeoff between prediction accuracy and computational complexity

[6] Typically, motion estimation is only carried out on the luminance (Y) component of a video frame and a suitably scaled version of the resulting motion vector is used for the chominance (U and V) components.

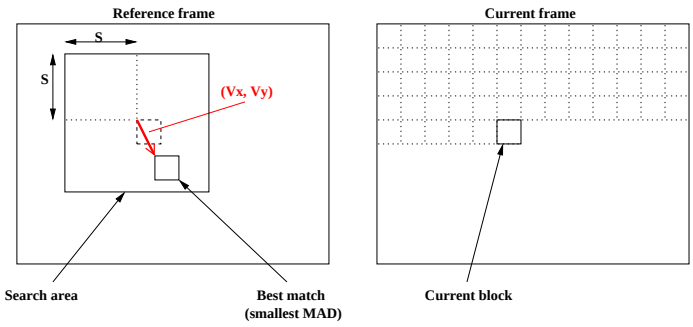


Figure 7.2: Block-based Motion Estimation

$$SAD(B_{curr}, B_{ref}) = \sum_{i=1}^{16}\sum_{j=1}^{16} |B_{curr}(i,j) - B_{ref}(i,j)|$$

Another option is to calculate the Mean Square Error (MSE) between the two blocks. However, experimental results have shown that the MAD and SAD are just as effective whilst being less computationally intensive[7].

The entire process of motion estimation for each block can be viewed as a minimisation process which attempts to minimise the function corresponding to the matching criterion over the search area. The point where this function is minimised indicates the position of the best match.

7.3.2 Search Strategies

A number of different search strategies exist for finding the position of the best matching block. In the following sections we discuss some of the more popular search algorithms.

Full Search

The most straightforward approach to searching is to simply search *every* position within the search area. Since the searching process is exhaustive, this guarantees that a minimum point in the matching function is found. However, the process is very computationally expensive.

Logarithmic Search

A logarithmic search is designed to reduce the number of positions which need to be searched in order to locate a minimum in the matching function. The approach attempts to iteratively converge on the minimum point. The algorithm is outlined below where s denotes the search step size in pixels at each iteration of the algorithm:

1. Set $s = 2(log_2 S - 1)$. Denote the origin of the search by (x, y) and intialise to (0,0)
2. Search the nine positions in the reference frame defined by s for the best match:

$$\begin{array}{ccc} (x-s, y-s) & (x, y-s) & (x+s, y-s) \\ (x-s, y) & (x, y) & (x+s, y) \\ (x-s, y+s) & (x, y+s) & (x+s, y+s) \end{array}$$

3. Set $s = \frac{s}{2}$. Set (x, y) to the best match position.
4. If $s = 0$ the search is finished, else go to 2.

[7]The matching criterion computation will have to be performed many times for each block in the current frame.

The most popular form of logarithmic search is when $S = 8$ so that , $s_1 = 4$, $s_2 = 2$ and $s_3 = 1$. This approach is called the *Three Step Search* and is illustrated in Figure 7.3(a). In the example of Figure 7.3(a), initially the search is centered around the original location of the current block with a step size of 4. The best match is found at position (4,0) (relative to the search origin). In the second step, the search centers around this point with a step size of 2 and the best match is found at position $(6, 2)$. Finally, in the third step, the search step size 1 and the best match is located at (5,3). Thus, for this block $V_x = +5$ and $V_y = +3$.

Using a logarithmic search algorithm is much more compationally efficient than performing a full search (see Bhaskaran). However, fast algorithms for motion estimation such as this have a tendancy to get caught at local minima in the matching function. This is because the assumption underpinning such approaches is that the matching function increases monotonically as the search position moves away from the minimum value. This is not true in general. Due to the reduced number of positions searched, it is possible that an incorrect search direction will be found during the one of the steps. Since subsequent steps only search around this position the process will converge on this local minimum. If this happens, then the motion compensated prediction will yield a higher prediction error than that obtained with a full search algorithm.

Parrallel One-dimensional Search

This searching strategy aims to use even fewer search positions than the logarithmic search with the added bonus of allowing parallel searching. The approach is to again iteratively converge on the best match but to perform independent searches in the horizontal and vertical directions. The algorithm is outlined below where s denotes the search step size in pixels at each iteration of the algorithm:

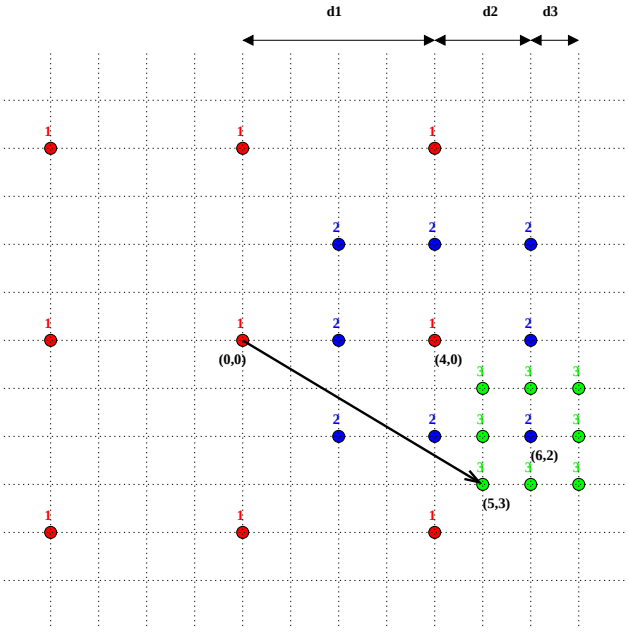
1. Set $s = 2(log_2 S - 1)$. Denote the origin of the search by (x, y) and intialise to (0,0)
2. In parallel compute:
   (a) x-axis local minimum by searching locations (x, y), $(x - s, y)$, $(x + s, y)$
   (b) y-axis local minimum by searching locations (x, y), $(x, y - s)$, $(x, y + s)$
3. Set $s = \frac{s}{2}$. Set (x, y) to best matching positions in horizontal and vertical positions respectively.
4. If $s = 0$ the search is finished, else go to 2

This search algorithm is illustrated in Figure 7.3(b) where $S = 8$. In this example, the best matching horizontal position after the first step is $x = 4$ and the best matching vertical position is $y = 0$, giving a new search origin of (4,0). In the next step, the best matching horizontal position is found to be $x = 6$ and the best matching vertical position is found to be $y = -2$, giving a new search origin of (6,-2) – even though this position is not explicitly searched itself. Finally the best matching horizontal position is found to be $x = 5$ and the best matching vertical position is $y = -2$, giving a motion vector with $V_x = +5$ and $V_y = -2$

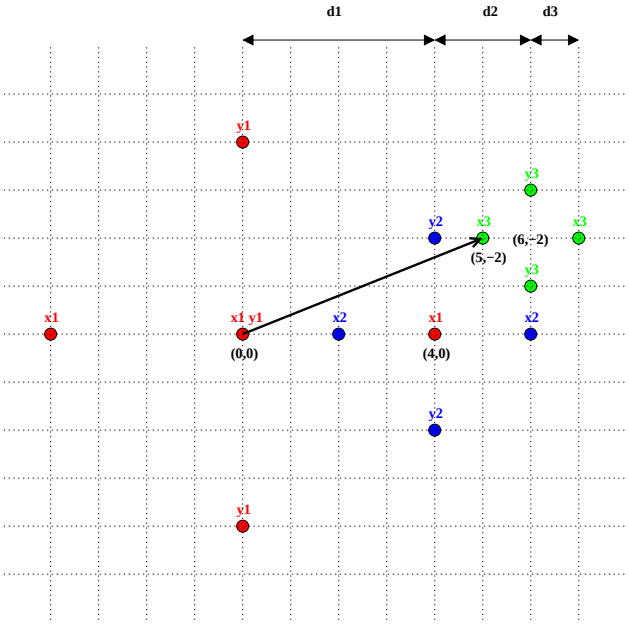
This search strategy is even more computationally efficient than the logarithmic search due to the reduced number of positions searched, but is at least as prone to converging on a local minimum in the matching function.

Example 7.1 *Given a $N \times M$ block of pixels, we will calculate the motion estimation complexity (in terms of the total number of operations required to calculate the motion vector for this block) using the following algorithms:*

- *Full search with a search window of ± 8 pixels in the horizontal and vertical directions*
- *Three step log search with an initial step size of ± 4 pixels in the horizontal and vertical direction*
- *Parallel one dimensional search with an initial step size of ± 4 pixels and one search location in the horizontal and vertical directions.*



(a) Three-step logarithmic search



(b) Parallel One-dimensional Search

Figure 7.3: Motion estimation search strategies

We will assume there are three operations – a subtraction, an absolute value calculation and an addition – for each pixel location. Furthermore, we will ignore the computational cost of accessing pixel data.

In the full search algorithm every pixel location in the search window in the previous frame is searched for a best match. The search window in the previous frame is 17×17 *pixels (*± 8 *pixel in the horizontal and vertical directions + the current block location). There are* $N \times M \times 3$ *calculations per search position and there are* 17×17 *search positions giving a total of* $(N \times M \times 3) \times 17 \times 17 = (N \times M) \times 867$ *calculations.*

As can be seen from Figure 7.3(a), in the Three Step Search there are 9 positions searched in step 1 (labelled 1 in Figure 7.3(a)), 8 positions searched in step 2 (labelled 2 in Figure 7.3(a)) and 8 positions searched in step 3. Thus the total number of calculations can be calculated as $(9+8+8) \times (N \times M \times 3) = (N \times M) \times 75$.

In the parrallel one-dimensional search algorithm the horizontal direction and vertical direction are searched independently. In the first step of this algorithm only five pixel positions are searched (labelled x1,y1 in Figure 7.3(b)). In the second and third step only four positions are searched (labelled x2,y2 and x3,y3 in Figure 7.3(b)), giving a total of 13 positions searched. Thus the total number of calculations required is $(N \times M \times 3) \times 13 = (N \times M) \times 39$.

7.3.3 Hierarchical Motion Estimation

In addition to searching fewer positions in a motion estimation process, it is also desirable to use fewer pixels in the matching criterion. This can be achieved by using subsampled versions of the current and reference frames. A strategy which uses subsampling is usually referred to as hierarchical motion estimation. A generalised overview of such an approach is illustrated in Figure 7.4. The approach taken is to generate several lower-resolution versions of current and reference images in a hierarchical manner. Starting with the lowest resolution (Level 3 in Figure 7.4), motion estimation is carried at each level in turn with the motion vector from the previous level used to initialise the search in the current level. At each level any motion estimation strategy can be used.

Figure 7.4 illustrates a three level hierarchy with subsampling by a factor of two. If the block size in Level 1 (the original resolution) is $N \times M$ pixels , then it will be $\frac{N}{2} \times \frac{M}{2}$ in Level 2 and $\frac{N}{4} \times \frac{M}{4}$ in Level 3. Similarly, if the current block location is (x, y) in the original image (Level 1) then this block will be located at $(\frac{x}{2}, \frac{y}{2}), (\frac{x}{4}, \frac{y}{4})$ in Levels 2 and 3 respectively. If the search window in Level 1 is $\pm S$, then a scaled version of the search window, $\pm \frac{S}{4}$ can be used in Level 3.

The algorithm proceeds by performing a search starting at $(\frac{x}{4}, \frac{y}{4})$, with block size $\frac{N}{4} \times \frac{M}{4}$ and search window $\pm \frac{S}{4}$ in Level 3. This results in a motion vector (Vx_3, Vy_3). A search is then performed in Level 2 starting at $(\frac{x}{2} + 2 \times Vx_3, \frac{y}{2} + 2 \times Vy_3)$ with block size $\frac{N}{2} \times \frac{M}{2}$ and a step size of ± 1 pixel around this location[8]. This results in a motion vector (Vx_2, Vy_2). In Level 1, the search is centered around $(x + 2 \times Vx_2, y + 2 \times Vy_2)$ with a step size of ± 1 and a block size of $N \times M$ resulting in a final motion vector (Vx_1, Vy_1).

Hierarchical motion estimation can reduce the probability of the search getting caught at a local minimum in the matching function due to the low pass filtering effect of the subsampling process. The process is also very computational efficient. However, inaccurate motion vectors may be obtained for small regions since (in the worst case scenario) these regions may disappear altogether due to subsampling. Furthermore the approach requires extra storage in the encoder and decoder for the different resolution images.

Example 7.2 *Given a* $N \times M$ *block of pixels, we will calculate the motion estimation complexity (in terms of the total number of operations required to calculate the motion vector for this block) using the hierarchical motion estimation algorithm illustrated in Figure 7.4 and outlined above for* $S = \pm 8$. *As before, we will assume that there are three operations for each pixel location and we will ignore the computational cost of accessing pixel data.*

- *Level 3: In this level the search window is* $\frac{S}{4} = \pm 2$ *pixels in the horizontal and vertical directions. As such, there are a total of* 5×5 *positions searched and the block size used is* $\frac{N}{4} \times \frac{M}{4}$. *Thus, the number of calculations required is:* $\frac{N}{4} \times \frac{M}{4} \times 75 \approx N \times M \times 5$

[8] Alternatively, a scaled version of the original search window size $\pm \frac{S}{2}$ could be used. We use ± 1 for illustration purposes only

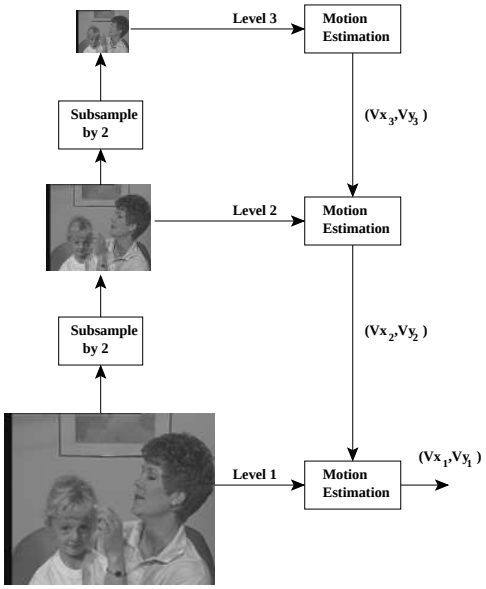


Figure 7.4: Hierarchical Motion Estimation

- *Level 2: In this level there are only 9 positions searched and the block size is* $\frac{N}{2} \times \frac{M}{2}$*. Thus, the number of calculations required is:* $\frac{N}{2} \times \frac{M}{2} \times 27 \approx N \times M \times 7$
- *Level 1: In this level there are again 9 positions searched and the block size is* $N \times M$*. Thus, the number of calculations required is:* $N \times M \times 27$

In this way, the total number of calculations required is $N \times M \times 39$*. This is the same number of calculations as required for the Parallel One-dimensional Search (see Example 7.2) but with the added bonus of the search being less likely to get caught at a local minimum in the matching function.*

7.3.4 Sub-pel Motion Estimation

The motion estimation algorithms described so far use integer pixel grids both for the search process, and as a consequence for the final motion vector. Of course, real motion in video sequences will not necessarily respect this artificial constraint (e.g. an object in the real world is not constrained to move a distance corresponding to an integer number of pixels). Clearly, it should be possible to achieve better motion estimates if we allow non-integer pixel postions in the search process. Such an approach is termed sub-pel motion estimation. The most popular form is half-pel accurate motion estimation whereby values for pixels at half-pel locations in the reference frame are formed via bilinear interpolation of their integer position neighbours. Note that it is not just the integer search positions which must be interpolated but any position used in the MAD/SAD calculation. Typically, the entire reference frame is bilinearly interpolated.

Very often a two step approach is employed in sub-pel motion estimation. In the first step, the best matching integer pixel position is sought. In the second step, this position is refined using sub-pixel accurate searching around this point. This process is illustrated in Figure 7.5.

7.4 Motion Compensation

The process of actually creating a prediction for the current image based on a set of estimated motion vectors is termed *motion compensation*. This is a relatively straighforward process compared to motion estimation. Given a motion vector for a current block it simply involves copying pixels from the displaced block position in the reference frame into the current block location. This process is illustrated in

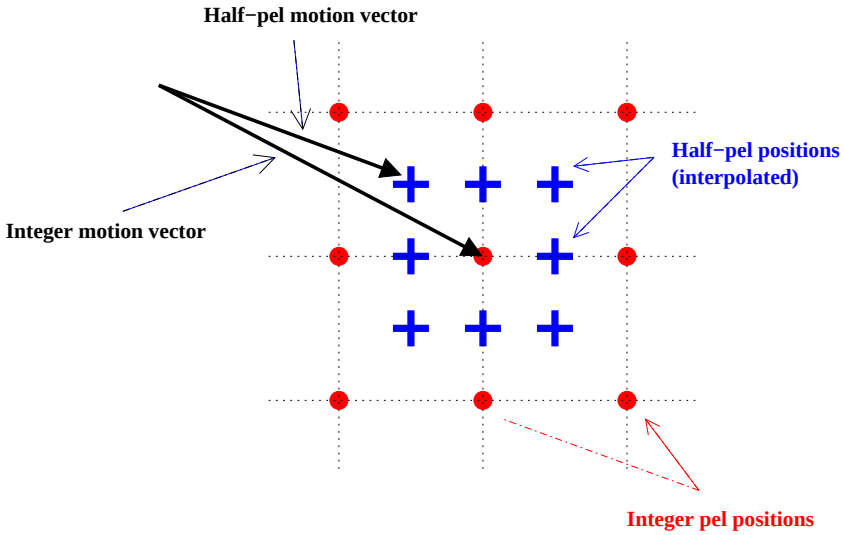


Figure 7.5: Half-pel precision motion estimation

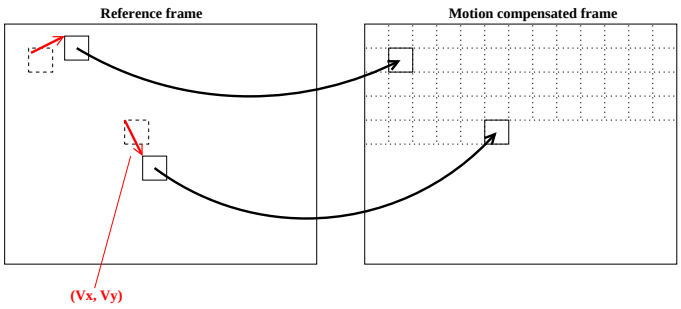


Figure 7.6: Block-based Motion Compensation

Figure 7.6. Clearly, this implies that the reference frame must be stored in the decoder. In fact, what is actually stored in the *reconstructed* (i.e. *decoded*) reference frame since the decoder only has access to decoded motion vectors and the decoded prediction residual. In order for the encoder and decoder to remain synchronised, it is important that the encoder uses the same image in the estimation procedure. For this reason, the decoding process is also carried out in the encoder and the resulting decoded reference frame used in motion estimation. Both motion estimation and compensation must be present in an encoder (see Figure 8.3), whilst only motion compensation is required in a decoder.

Example 7.3 *An example of the usefulness of motion estimation and motion compensation is illustrated in Figure 7.7. This Figure depicts the motion compensated frame and the prediction residual subsequently obtained for three different motion estimation algorithms applied to the first two frames of the Foreman test sequence. The first frame of the sequence, as illustrated in Figure 7.1(a), was chosen as the reference frame and used to form a motion compensated version of the second frame of the sequence. The original second frame of the sequence is illustrated in Figure 7.1(b). The prediction residuals illustrated in Figure 7.7 are the difference images (again, scaled to viewable versions) between the motion compensated frame and the original image for each algorith.*

The different motion estimation algorithms used were:

- *ME algorithm 1: a full search algorithm using a block size of 16×16 pixels and a search window of ±8 pixels in the horizontal and vertical directions.*

(a) Motion compensated frame using ME algorithm 1

(b) Prediction residual for ME algorithm 1

(c) Motion compensated frame using ME algorithm 2

(d) Prediction residual for ME algorithm 2

(e) Motion compensated frame using ME algorithm 3

(f) Prediction residual for ME algorithm 3

Figure 7.7: Illustration of the benefits of motion estimation and motion compensation: 7.7(b) entropy ≈ 2.61 bits/pixel, 7.7(d) entropy ≈ 3.08 bits/pixel, 7.7(f) entropy ≈ 2.91 bits/pixel

- *ME algorithm 2: a modified three step algorithm using block sizes of 16×16, 8×8 and 4×4 pixels and search windows of ±8, ±4 and ±2 pixels in each of the steps.*
- *ME algorithm 3: a modified three step algorithm using block sizes of 32×32, 16×16 and 4×4 pixels and search windows of ±8, ±4 and ±2 pixels in each of the steps.*

As can be seen from Figure 7.7 the motion estimation algorithm which results in the prediction residual with lowest entropy in this case is ME algorithm 1. This is because the other two motion estimation algorithms are actually designed to produce a dense motion vector field representative of the actual motion present in the sequence for use in object segmentation and tracking applications.

Chapter 8

Structure of a Video Codec: ITU-T H.261 & H.263

8.1 Introduction

In this chapter we discuss the high level structure of a typical video encoder/decoder. This is best illustrated by taking an example of a specifc coding scheme. The example chosen is a well known international standard for video conferencing.

8.2 Background

In response to the growing demand for visual telephony and video conferencing services, the International Telecommunications Union (ITU-T) developed video compression standards targeted at this narrow bandwidth real-time application. Recommendation H.261 is a video codec specification suitable for real-time audiovisual (AV) services at $p \times 64$ Kb/sec, where p is the range of 1 to 30. This choice of bandwidth was motivated by the bit-rate of baseline ISDN (Integrated Services Digital Network). The overall codec has a hybrid structure incorporating both intraframe and interframe coding. H.261 supports both CIF and QCIF resolution video sources and allows up to three frames to be interpolated in the decoder, thereby supporting framerates of 15Hz, 10Hz, 7.5 Hz.

8.3 Video Multiplex

The basic coding unit of H.261 is known as a macroblock. Each macroblock consists of four 8×8 blocks of luminance pixels and and two 8×8 chominance blocks (for U and V). The macroblock structure is illustrated in Figure 8.1.

However, a H.261 video bitstream itself actually has a hierarchical structure. The different layers in the hierarchy are referred to as:

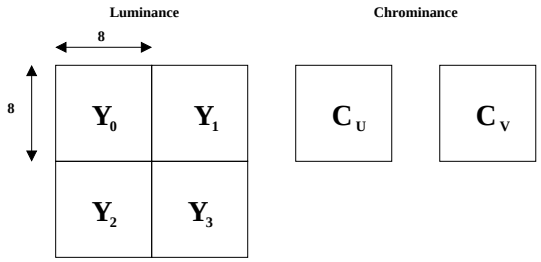


Figure 8.1: Structure of a macroblock

- Picture layer
- Group of Blocks (GOB) layer
- Macroblock layer
- Block layer

In the following, the more "important" bitstream syntax elements are outlined for each layer. For a full explanation of the bitstream syntax, the reader is referred to actual bitstream diagrams such as those illustrated in [3] and [4].

The picture layer in the bitstream consists of a fixed length code (FLC) known as the picture start code (used for synchronisation purposes) and some picture-level side information outside the scope of H.261 (e.g. temporal reference, etc. used in the systems bitstream). The picture layer contains the GOB layer.

The GOB layer consists of groups of 11×3 macroblocks arranged for CIF and QCIF as illustrated in Figure 8.2. The GOB layer has a FLC for the GOB start code (for synchronisation), a FLC for the group number code indicating one of 12 possible GOBs (see Figure 8.2) and a FLC for the group quantizer which remains the default quantizer used until subsequently updated in the macroblock layer. The GOB layer contains the macroblock layer.

The macroblock layer consists of a FLC for the macroblock address (see below) signalling the position of the macroblock in the GOB, followed by a variable length code (VLC) for the macroblock type (see below). The presence of other bitstream syntax elements in the macroblock layer depends on the type of encoded macroblock. Posssible syntax elements are a FLC for the macroblock quantizer (updating the previously transmitted GOB quantizer), VLCs for motion vector data and a VLC for the coded block pattern (CBP)(see below). The macroblock contains the block layer.

The block layer is the lowest level of the hierarchy and contains intra/inter encoded pixel data for the 8×8 blocks composing a macroblock. This layer simply consists of VLCs for zigzag scanned, quantized DCT coefficients and a special End of Block (EOB) fixed length code (FLC) to signal that no more data need to be decoded for a particular block.

8.4 Codec structure

A simplified diagram of the H.261 codec structure is illustrated in Figure 8.3. Each macroblock is encoded using motion estimation/compensation followed by an 8×8 DCT with zig-zag scan and quantization. Each macroblock is decoded and reconstructed in the encoder i.e. inverse quantisation, followed by a IDCT the result of which in the case of inter blocks, is added to the prediction residual obtained with the transmitted motion vector for that block. Decoded macroblocks are aggregated to form the previous reconstructed frame which is stored in the frame memory and used in motion estimation/compensation when coding the next frame.

8.4.1 Macroblock type

There are different types of macroblock possible. For example, not every inter macroblock may require motion compensation – the inter frame difference may be sufficiently small without motion estimation. Some blocks will require transmission of DCT coefficients, whilst for others a motion vector might suffice – the motion compensated prediction may produce a zero residual. Similarly, some blocks can be coded with the default quantizer whilst others will need to specify a new quantizer (for rate control purposes). The actual type of encoded macroblock must be signalled in the bitstream so that the decoder knows what information to subsequently expect in the bitstream for that macroblock. The different types of macroblock possible are listed below. The decision path followed in the encoder to decide a macroblock type is illustrated in Figure 8.4.

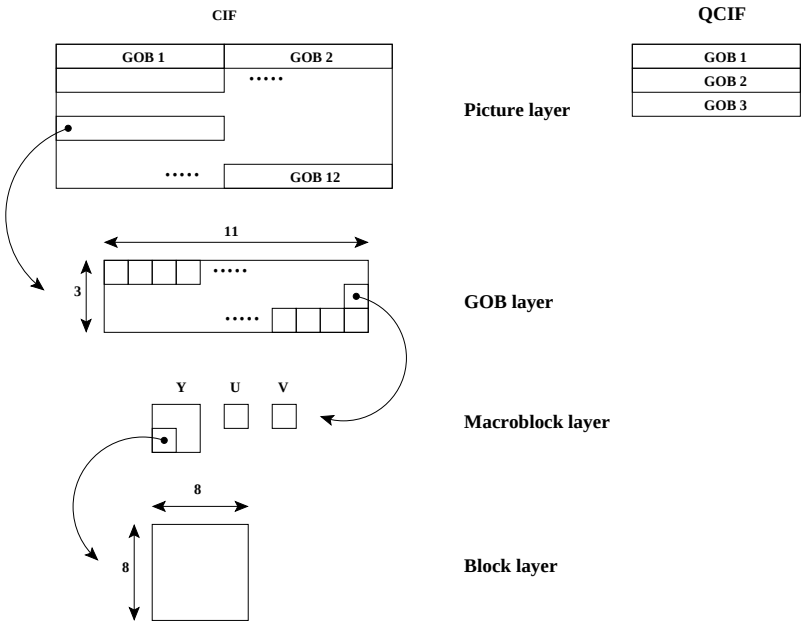


Figure 8.2: Pictures, GOBs, macroblocks and blocks

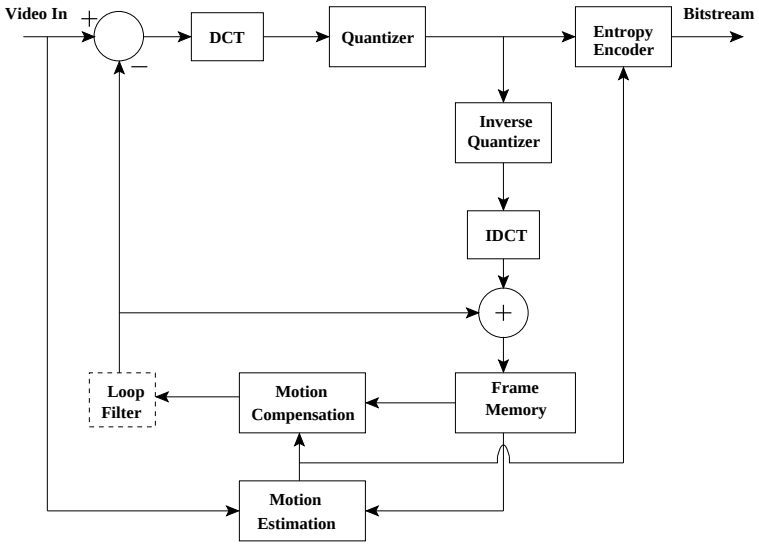


Figure 8.3: Structure of a H.261/H.263 encoder

- MC, not coded
- MC, coded
- MC, coded + Q
- intra coded
- intra coded + Q
- No MC, coded
- No MC, coded + Q
- skipped

The first decision taken for each macroblock (see Figure 8.4)is whether or not to use motion compensation. To this end, the motion compensated prediction residual (summed over the macroblock) is compared to the residual obtained using straight frame differencing (again summed over the macroblock). If the motion compensation residual is not substantially smaller then the non-motion compensated residual, then "No MC" mode is chosen for the macroblock. This saves the bit overhead associated with a motion vector which is not producing a substantial gain.

A decision to use motion compensation – MC mode – requires that a motion vector be coded and transmitted for the macroblock. Clearly, this also implies inter mode coding (intra mode coding is not possible with motion compensation). In this mode it must also be decided whether or not to code the prediction residual associated with motion compensation and if so what quantizer to use. If all DCT coefficents evaluate to zero after quantization, then it is not necesary to code anything except a motion vector for this macroblock. This is the "MC, not coded" mode listed above. Alternatively, the transform coefficients can be encoded with the default quantizer – "MC, coded". Optionally, the quantizer used to encode transform coefficents can be updated for each macroblock – "MC, coded + Q".

If a decision not to use motion compensation is made – No MC mode – then it must be further decided whether to use intra or inter frame encoding for this macroblock. This intra/intra mode decision is made in a manner similar to the MC/No MC decision, except that this time the variance of the macroblock is computed and thresholded to produce a decision (small variance implies inter mode coding). In the event of an intra decision, then the transform coefficents can be encoded with the default quantizer or with an updated quantizer – "intra coded" and "intra coded + Q", respectively. In the case of an inter mode decision, again the default quantizer or an updated quantizer can be used – "No MC, coded", "No MC, coded + Q".

If a decision is made not to use motion compensation and if the quantized coefficients of all six blocks in a macroblock are all equal to zero then no information need be encoded for this macroblock. This is termed a "skipped" macroblock. Upon determining that a macroblock is skipped, the decoder can simply copy the macroblock's content in the previous frame into the current frame.

8.4.2 Block addressing within a macroblock

Not every block need be coded in a coded macroblock. After quantization, some blocks will have all zero coefficients and thus require no data to be transmitted. However, the pattern of coded/not coded blocks in a coded macroblock needs to be transmitted to the decoder. Since there are six blocks in a macroblock there are $2^6 = 64$ possible combinations of coded/not coded blocks but skipped macroblocks require no overhead leaving 63 possibilities. The coded block pattern within a macroblock is encoded using a VLC called Coded Block Pattern (CBP)[1]. The CBP number (i.e. the code identifiying a particular pattern) is calculated as:

$$CBP_{number} = (32 \times Y_0) + (16 \times Y_1) + (8 \times Y_2) + (4 \times Y_3) + (2 \times C_U) + C_V$$

where each coded block is assigned a "1" or "0" in this equation depending on whether it is coded or not.

[1]The exception to this is intra macroblocks, for which data is transmitted for all blocks.

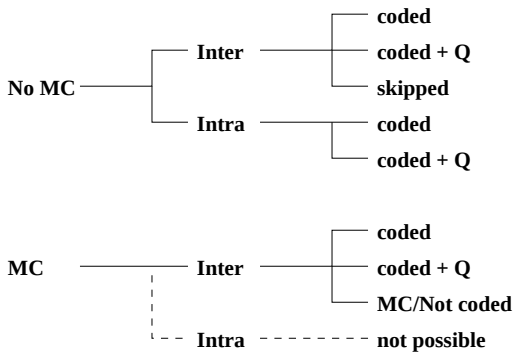


Figure 8.4: Decision tree and macroblock types

8.4.3 Macroblock addressing within a GOB

Since macroblocks can be skipped and no data is encoded for skipped macroblocks, it is necessary to send a macroblock address within a GOB for a coded macroblock. This address is very efficiently encoded using a run-length mechanism. H.261 was designed for videoconferencing applications which typically consist of a foreground (moving) person(s) against a static background. As such, typically skipped macroblocks will be clustered in the background whilst coded macroblocks will be clustered in the foreground. This clustering of similar macroblock types is exploited when encoding the macroblock address. Macroblock addresses are indicated in terms of the run of skipped macroblocks prior to the next coded macroblock[3].

8.4.4 DCT

A two-dimensional 8×8 DCT is used for spatial data compression. In the case of a inter macroblock type decision, the input data to this DCT is the prediction error for a block. In the case of an intra mode decision, the input data are the pixel values themselves. DCT coefficients are quantized, zig-zag scanned and entropy encoded. The intra DC coefficient is encoded using a fixed length code-word (FLC). There are 31 quantizers available for the AC and inter DC coefficients. The quantized coefficients are encoded by VLCs using RLC [3].

8.4.5 Motion Compensation

H.261 specifies one motion vector per macroblock. The luminance motion vector component are halved and truncated to zero to obtain the chominance motion vectors. The motion vector data is predictively encoded using VLCs. The prediction is the motion vector of the previous macroblock in the GOB (using an initial prediction os $(0, 0)$ for the first macroblock in a GOB). The prediction residuals for both the horizontal and vertical components of the motion vector are encoded independently [3].

8.4.6 Loop filter

A large quantizer must be used at low bitrates in order to meet bandwidth resuirements. This typically results in all but the DC and a few high AC coefficients reduced to zero. In the decoded picture this results in "blocking artefacts". The visual effect of these blocking artefacts occuring in different parts of the image over time is referred to as *mosquito noise*. Furthermore, motion compensation can produce high frequency artefacts in the image (due to blocks containing edges being slightly mis-aligned). For these reasons, an optional low pass filter may be introduced into the codec after motion compensation. The filter can help to reduce the quantization noise in the feedback loop and/or to reduce the effects motion compensation artefacts [3]. No filter coefficients are defined but a separable filter, similar to that used in JPEG, is recommended.

8.4.7 Encoder design

Selected processing steps and parameters of the H.261 compression scheme are not specified. These are normally parts of the compression algorithm which are non-normative and yet have a significant impact on overall image quality and compression efficiency. This provides a suitable forum for competition of H.261 products in the market-place. For example, the method of motion estimation in the encoder, the use of a loop filter, the arithmetic process for computing the DCT, the control of the data rate, and the method of mode decision for each macroblock can be defined by the encoder designer [2].

8.5 ITU-T H.263 Video Compression

In addition to H.261, the ITU-T has produced another standard for video compression, developed for very low bit-rate applications such as video telephony over the public services telephone network (PSTN) and mobile telephony. This recommendation, known as H.263, is very similar in concept to H.261, although with significant improvements which allow the video codec to operate efficiently at bit-rates lower than 64 Kb/sec [3]. Like H.261, the overall codec structure is a hybrid DPCM/DCT encoding system. The simplified diagram of the H.261 encoder in Figure 8.3 translates directly to H.263. In the following, the main differences between the two are outlined.

In H.263, motion compensation with half-pixel accurate motion vectors is allowed. Motion compensation is performed using bilinear interpolation. Also, motion vectors are allowed to point outside the current image. This latter is called unrestricted motion vector mode. In this case, pixels at the edge of the image are used to form the prediction [3]. Another motion compensation method known as advanced prediction mode is also available. In this case, each macroblock can have four motion vectors associated with it. The prediction is formed using a technique known as overlapped block motion compensation (OBMC) [3].

An optional encoding mode known as syntax-based arithmetic coding (SAC) mode is available in H.263. In this mode, all VLC operations of H.263 are replaced with more efficient arithmetic encoding/decoding operations [3]. An optional extra type of encoded frame, known as a PB-frame is also available. This is actually two separate frames. The first is a normal P-frame, predicted from the previous P-frame. The second is a B-frame (where B stands for "bi-directional") which is predicted from both the previous P-frame and current P-frame being decoded [3].

[2] Whilst, motion compensation in the encoder is optional in H.261 the decoder must be capable of accepting one motion vector per macroblock, even if it is subsequently discarded.

Chapter 9

MPEG-1: video compression for digital storage media

9.1 Introduction

The aim of this chapter is not to cover sufficient detail to allow an MPEG-1 codec to be developed, but sufficient detail to allow high-level application decisions to be made about what is and is not possible with an MPEG-1 stream.

MPEG-1 (also known as ISO/IEC 11172) was the first video and audio coding standard from the ISO Moving Picture Experts Group. It is effectively a technology for coding digital audio-video material for storage that builds and improves upon the earlier CCITT (now ITU-T) H.261 standard for video conferencing in telecomms applications. It was designed to deliver digitized and compressed video signals at the maximum sustained data-transfer rate that could be handled by CD-ROM drives at the time, i.e., up to about 1.5 Mbits per second. CD-ROMS were targeted because with a capacity of about 650Mbytes, they were the only removable media that could support the then enormous storage demands of digitized video. The time at which all of this happened, the early 1990s, was in turn determined by the convergence of the technology of video compression and the projected speeding up of the hardware that would be required to deliver this compression in real-time.

It is important to note that the MPEG group only standardized the *bit stream format* and the *decoder structures*. This leaves scope for manufacturers to develop unique selling points and products to meet specific needs while maintaining inter-operability with the industry standard decoder.

As with the JPEG standard, MPEG-1 encoding tries to exploit statistical and subjective redundancy *within* individual frames of a video sequence. This alone is not usually sufficient to deliver the required quality at the available bit rate, so it was clear that, like in H.261, it would be necessary to also exploit the redundancy that almost invariably exists *between* frames in a video sequence. Remember though, the structure of the MPEG-1 coding scheme is not inevitable: it is a compromise based on the trade-off between performance (the combination of compression rate and quality) on one hand, and implementational complexity given the expected availability of suitable hardware, on the other. The triumph of the standardization efforts was not to find a coding scheme that would give the best compression, but to find the most acceptable compromise between compression, quality and complexity, in a context where even though the target data rate was known, the quality that would be expected by or acceptable to the future users, and what hardware computing power would be available, could only be guessed at.

In the subsequent development of video coding standards for broadband telecommunications, the problem was even more difficult as the target data rate was also unknown, or at least the pricing structure that would determine the user's ability to get value for the content was unknown. The typical interaction between video engineers and telecomms engineers involved in the development of MPEG-4, the first convergence of the computer and the telecommunications video coding standards was "How much data rate can you give us?", which was answered with "Well, how much data rate do you want?". The circular discussion was only ended by MPEG-4 embracing data rates from 5kbits per second to 10Mbits per second.

9.1.1 MPEG-1 Standards Documents

Standards documents for standards in this field typically contain sections that fall into one of three categories:

- Normative sections: these define the standard and the requirements that it imposes;
- Informative sections: these give background material to help understanding and implementation;
- Comparative sections: these describe how an implementation should be tested to determine compliance.

In this context, the MPEG-1 standard has five parts:

- ISO/IEC 11172-1: MPEG-1 Part 1: Systems layer (normative & informative)
- ISO/IEC 11172-2: MPEG-1 Part 2: Video (stream) (normative & informative)
- ISO/IEC 11172-3: MPEG-1 Part 3: Audio (stream) (normative & informative)
- ISO/IEC 11172-4: MPEG-1 Part 4: Compliance testing (comparative)
- ISO/IEC 11172-5: MPEG-1 Part 5: Software reference model (informative)

Three terms that are used in the MPEG-1 standards documents with slightly different meanings in different places are *access unit*, *presentation unit* and *layer*. In an MPEG-1 audio bitstream, an *access unit* is the smallest independently decodable segment, and a *presentation unit* is a decoded audio access unit. In a video bitstream, the *presentation unit* is a picture, while an *access unit* is the coded representation of that picture. However, most pictures in a bitstream are not independently decodable.

The term *layer* is used to describe the structuring of the bitstream into units and sub-units, as in the packs and packets of the system layer, or the sequences, GOPs, pictures, slices, macroblocks and blocks of an elementary video stream. However, the term *layer* is also used to describe the three different algorithms that are used for coding MPEG-1 audio streams, which differ in their level of complexity, and only one of which would be used in a given coded elementary audio stream. Thus, for example, "MP3" is an abbreviation of MPEG-1 Layer 3 audio coding. Also, because of its desire not to use the term *frame*, the MPEG-1 standard promotes the prefixes *inter-* and *intra-* to adjectives.

Our primary interest in this course is in video, so here we treat MPEG-1 video first (and in some depth), even though it is a sub-layer of the standard. Later we deal with the higher level systems-layer that combines video and audio components. Note that even though MPEG-1 video and MPEG-1 audio are subsets of the overall MPEG-1 standard, most players can play MPEG-1 video-only or MPEG-1 audio-only streams, not just the combination of the two in an MPEG-1 systems layer stream.

9.2 Overview of MPEG-1 Video

MPEG-1 Video is a standard for the compression of individual (three-component) colour pictures occurring at fixed time intervals in a video sequence. Features of MPEG-1 are that it

- supports a wide range of (usually computer-based or network-based) applications;
- supports flexible picture size (typically, though not required to be $\leq$ 768x576) and frame rate (typically, though not required to be $\leq$ 30Hz), which covers video source parameters up to TV size including both PAL and NTSC video standards, and should be able to give similar quality to analogue VHS video recording;
- expects non-interlaced video input, but does not define the relationship between the fields and frames of interlaced TV signals and the rectangular arrays of coloured pixels (called *pictures* in the standard) defined as the inputs to the MPEG-1 encoding process;
- retains a high degree of commonality with H.261 without the telecomms demand for low coding delay and without bitstream compatibility with H.261;

- targeted multimedia CD-ROM-based applications with sustained bitrates typically up to, though not limited to 1.86Mbits/sec;
- supports picture-based random access of video, though doesn't guarantee it in all cases;
- supports fast-forward and fast reverse through compressed streams, though doesn't guarantee it in all cases;
- supports reverse playback of video, though doesn't guarantee it in all cases;
- supports editability of compressed stream, though doesn't guarantee it in all cases;
- is based on a macroblock structuring of an image (Y=16 × 16, U,V=8 × 8), motion compensation and the conditional replenishment of macroblocks. See Figure 9.8.

The key elements of MPEG-1 are motion compensation, the discrete cosine transform and visually-weighted quantization.

- The DCT gives coefficients that are strongly decorrelated, i.e., virtually independent of each other, except for the DC coefficient, and so they can be coded independently of each other, particularly for intra coding.
- Motion compensation is achieved by the decoder locating 8 × 8 blocks of pixels pointed to in a previous and possibly in a future reference picture, averaging them and then adding differences decoded from quantized DCT coefficients. The motion vectors are the same for for each block in a macroblock.
- DCT coefficients are quantized based on a visually-weighted model so that high frequencies are coded less-accurately. Quantization is implemented by simple integer division and can be crudely adapted from macroblock to macroblock to adjust the bitrate.
- Variable length codes (VLCs) were determined by Huffman coding and are fixed in the standard. VLCs are used to code quantized DCT coefficients as"symbols" or "events" consisting of a number of zeros in a sequence in the zig-zag scanned form of an 8 × 8 block followed by a non-zero value, or to code an EOB symbol that indicates that all remaining coefficients for a block are zero.
- DCT coefficients from intra and inter coding have very different characteristics and so different quantization tables and VLCs are used for each.

MPEG-1 uses three different types of picture:

I-picture These are coded independently of other pictures, like a JPEG image.
They are access points for random access.
They allow FF/FR functionality.
They don't exploit inter-frame redundancy so they are relatively bit-rate expensive.

P-picture These are coded on the basis of a prediction from the previous I or P picture, unless there is a requirement for I-coding of particular macroblocks.
They are not suitable access points for random access or edits.

B-picture These are coded on the basis of the combination of a prediction from the previous reference (I or P) picture and the next reference picture unless there is a requirement for I-coding of particular macroblocks.
They achieve high compression because they can use parts of the scene that will be un-occluded in a future picture, but at the expense of typically a two-picture delay.
They are not suitable access points for random access or edits.
They are *never used for prediction*, so they can be coded with large error (high compression) without this error propagating to future pictures. At the receiving end, B-pictures can be discarded (for example because of buffer overflow) without affecting the decoding of other pictures.

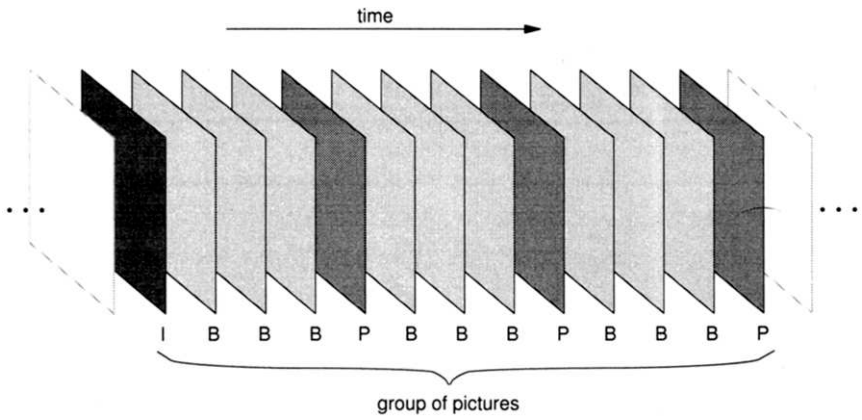


Figure 9.1: Diagram of a typical sequence of coded pictures types.

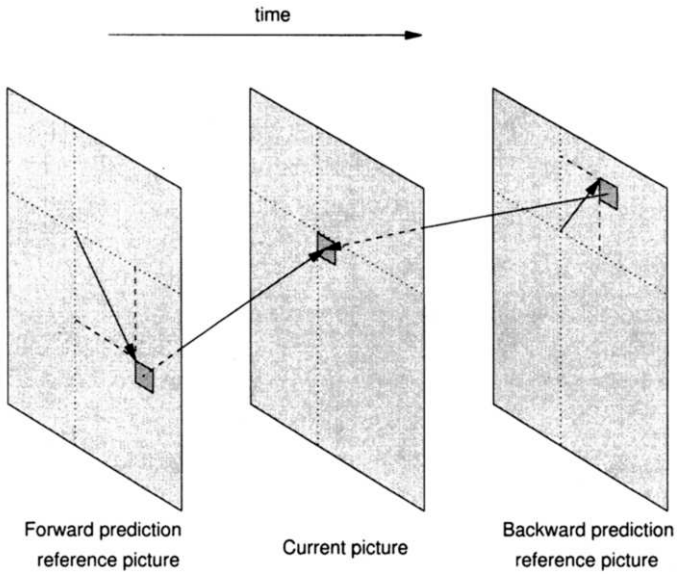


Figure 9.2: Diagram of the relationship between forward and backward prediction motion vectors and the macroblock being reconstructed.

The temporal relationship between the three picture types is illustrated in Figure 9.1, where the pictures are in display order. The first I-picture is coded independently of all other pictures. The first P-picture (the first mid-grey picture from the left) is predicted from the first I-picture, with any residual being DCT coded. The intervening B-pictures are predicted from a combination of the I and P-picture already referred to, with any residual being DCT coded. The prediction combines data from the I and P-picture with equal weighting, irrespective of the reference picture to which the B-picture is nearest. The macroblocks in the respective reference pictures that are used to predict a given macroblock are specified by so-called *motion-vectors* relative to the given macroblock. The relationship between the given macroblock and the corresponding two reference macroblocks specified by motion vectors is illustrated in Figure 9.2.

9.3 Video Encoding and decoding process

Remembering that the MPEG-1 standard only specifies the bitstream syntax and semantics, typical basic loop structures of the MPEG-1 encoder and decoder are illustrated in Figures 9.3 and 9.4. These

loops are primarily responsible for encoding and decoding the residual after Intra or Inter prediction and do not illustrate the image storage required for forward, backward or bi-directional prediction.

The storage required for the most general case of bi-directional prediction is shown in Figure 9.5. The first I picture of a sequence will be loaded into the *previous picture store* as it is decoded. It is typically displayed or otherwise stored as a decoded picture to disk at this point. A P picture will be decoded next and stored in the *future picture store*. It is *not* displayed at this point in time because it is out of sequence. As each one of the intervening B pictures is read from the bitstream, its macroblocks will be decoded by appropriate "forward prediction" references to the *previous picture store* and "backward prediction" references to the *future picture store*. These B pictures do not need to be stored for any decoding purposes, but are typically displayed or otherwise stored as a decoded picture to disk in turn. When the next P or I picture is detected in the bitstream, this is a signal that the first I picture is no longer needed. The contents of the *future picture store* (our first P picture above) are then used to overwrite the *previous picture store* and are displayed. The new P or I picture is then decoded from the bitstream into the *future picture store* with "forward prediction" references to the new contents of the *previous picture store* as required for this decoding and is not displayed at this point. Subsequent B pictures are then decoded and displayed as above until the next I or P picture is detected.

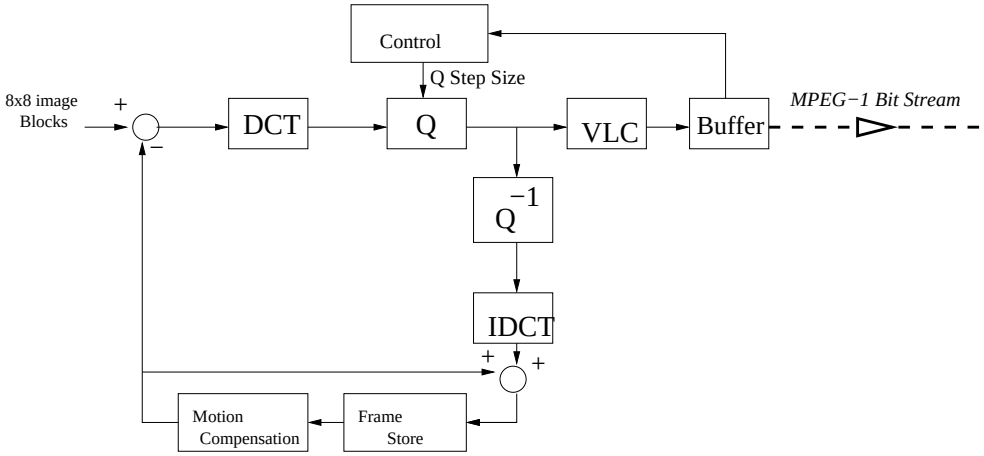


Figure 9.3: Diagram of the MPEG-1 transform coding and predictive coding loop structure.

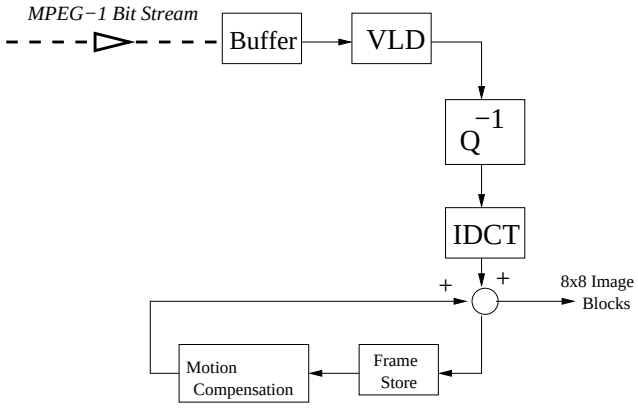


Figure 9.4: Conventional diagram of the MPEG-1 decoding loop structure.

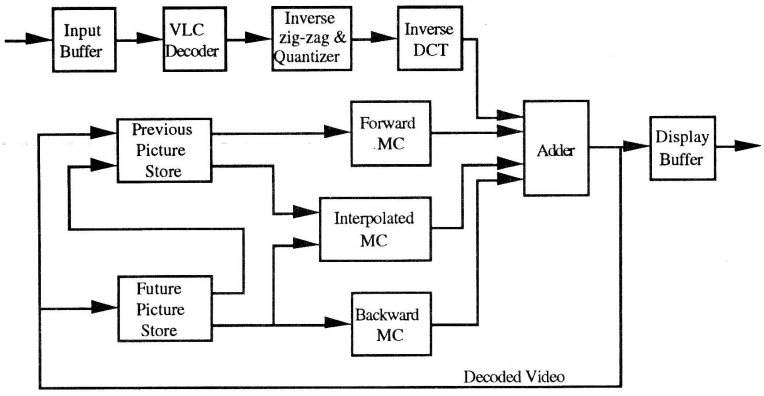


Figure 9.5: Diagram of the MPEG-1 decoding loop structure showing the individual framestores required.

9.3.1 Intra Coding

The first picture in a sequence is always coded as an I-picture. The user or the user's application determines which subsequent pictures will be coded as I-pictures, but the maximum gap between I pictures is 128.

Encoder

- For each macroblock, an 8×8 DCT is applied to Y0,Y1,Y2,Y3,U,V in turn.
- DCT coefficients are uniformly quantized (and quantizer stepsize is transmitted to the decoder). This is the only source of lossy compression in the coding of I-pictures.
- For Y blocks, the DC coefficients are coded by *lossless* differential prediction between the Y blocks within a macroblock and on to Y blocks in consecutive macroblocks.
- For colour blocks, the DC coefficients are coded by *lossless* differential prediction just between the corresponding U and V blocks in consecutive macroblocks. These two elements of DC coefficient coding are illustrated in Figure 9.6. The DC coefficient prediction is set to 1024 at the start of each slice.
- The AC coefficients are zig-zag scanned, run-length coded and entropy coded using VLCs, independently of each other because of their decorrelation.
- A copy of the picture is stored in a picture-store for prediction for the next P-picture or for neighbouring B-pictures.

Decoder

- Extracts and decodes the variable-length codes to give locations and values of the non-zero DCT coefficients.
- DCT coefficients are reconstructed by inverse quantization.
- IDCT used to get reconstructed Y1,Y2,Y3,Y4,U,V pixel values.

Aside on differential coding

The differentials mentioned above are coded as a VLC for the number of bits (e.g., *dct_dc_size_luminance*, followed by the given number of bits to carry the value. If a normal nine-bit binary representation of the differential values was used, then every DC differential would require nine bits. However, the smaller increments $0, \pm 1, \pm 2$ are much more frequent than others and in the standard scheme they are coded for luminance (Y) with respectively three bits, three bits (2 VLC + 1 binary value bit) and four bits (2 VLC + 2 binary value bits).

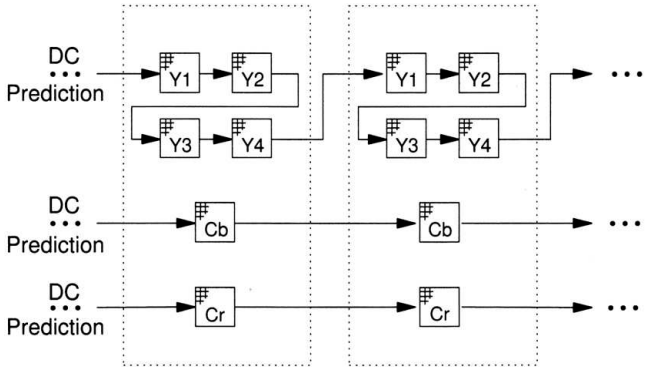


Figure 9.6: Diagram of the relationship between forward and backward prediction motion vectors and the macroblock being reconstructed.

Y code	C code	# bits	magnitude range
100	00	0	0
00	01	1	-1,1
01	10	2	-3...-2,2...3
101	110	3	-7...-4,2...7
110	1110	4	-15...-8,8...15
1110	1111 0	5	-31...-16,16...31
1111 0	1111 10	6	-63...-32,32...63
1111 10	1111 110	7	-127...-64,64...127
1111 110	1111 1110	8	-255...-128,128...255

9.3.2 Inter Coding

Encoding P-pictures

Encoder

- The previous I or P picture is stored in a picture store in the encoder (and the previous *reconstructed* I or P-picture is stored in a picture store in the decoder).
- Motion estimation is performed on a macroblock basis and the motion vector (one per macroblock) is encoded and transmitted

Motion compensation is the process of using the located motion vector to select a 16×16 block of pixels from the previous picture in the picturestore (obviously not necessarily aligned with macroblock boundaries) and subtracting it from the current macroblock in the current picture. Half-pel motion estimation is used.

- The residual from the subtraction of the previous picture pixels is then transformed by the usual six sequential 8×8 DCTs, quantized, run-length coded and VLC coded as before.
- A video buffer coupled with a decision-making process determining the quantization step size on a macroblock-basis is used on the output to control the output data rate. This process can be used to give constant or variable bit rate as desired. The rate control algorithm is not part of the MPEG-1 standard.

Decoder

- In the decoder, the VLD, inverse quantization, and inverse DCT operations give a reconstruction of the motion-compensated residuals.

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Figure 9.7: Default intra-coding quantization table.

- Using the transmitted motion vector, the appropriate 16×16 block of pixels from the reconstructed previous image in the picturestore is added to the reconstructed residuals to give a reconstruction of the current macroblock.

Note that this reconstructed macroblock will be in error (i) because of lossy encoding of the current macroblock residuals and (ii) because of reconstruction error in the 16×16 block of pixels pointed to by the current motion vector from the reconstructed previous picture in the picturestore.

Note also that the MPEG-1 scheme only goes through the process of coding pixel values or motion-compensated residual pixel values if the pixel values have changed sufficiently from the previous pixel values at this macroblock position to warrant it.

Encoding B-pictures
The encoding process for B-pictures is similar to P-pictures except that the decision-making is more complex. The encoder has to

- decide how to partition the picture into slices
- search for the best forward motion and backward motion vectors for each macroblock
- determine whether to use forward, backward or interpolated motion compensation, or just do intra coding of each macroblock
- set the quantizer scale

Also, B-pictures do not need to be stored as they are never used as a prediction for any other picture. This means that they can be safely coded with larger error, without any concern about this error propagating.

In both P and B pictures, the DC coefficient (for non-intra macroblocks) is already coding a differential value and there is no need to predictively code it, so it is treated the same as all the other DCT coefficients.

9.3.3 Quantization

The key data reduction step in the MPEG-1 scheme is its implementation of quantization. All the other features of the scheme such as picture differencing and motion compensation are designed so that the quantization process carried out on the array of 64 coefficients produced by the DCT make as many quantized coefficients as possible equal to zero, and reduce the number of bits needed to represent the remainder. As with the JPEG standard, the quantizer implemented for each coefficient is a uniform quantizer and each one has the same step size—only the number of steps changes from one coefficient to the next.

Intra Coding

As with JPEG, the fact that the human visual system is less sensitive to high spatial frequencies is exploited in the quantization step. For CIF video pictures viewed at normal viewing distance (4-6 times image height), the quantization matrix shown in Figure 9.7 has been shown to be suited for good quality coding of intra-coded content.

The quantization can be changed on a macroblock-by-macroblock basis. The way this is done is by defining two cases: *Intra-d* macroblocks, represented by the VLC "1", which use whatever the current

quantizer scale is, and *Intra-q* macroblocks, represented by the VLC "01", which uses a further five bits to specify a new quantizer scale between 1 and 31. The VLC definition in the standard for flagging the two different cases leaves open the possibility for extending to further cases in the future. The quantizer scale value multiplies into the quantization matrix to give the quantizer step size for each DCT coefficient, except for the quantizer step size of the DC coefficient, which is fixed at 8. This latter is the case because the human visual system is quite sensitive to a luminance step between areas of the size of a block and so the coding accuracy of the DC coefficient must be maintained. Given input pixel values in the range $[0, 255]$, the DC coefficient calculated as defined in the standard can take any value in the range $[0, 2040]$. Hence the integer division by 8 carried out in the quantization step reduces the dynamic range of the DC coefficient to one part in 256, which is still in excess of the dynamic range capabilities of the visual system.

The quantization process is implemented as a division by the quantizer step-size, followed by a rounding operation. In intra coding the result of the division is rounded to the nearest integer. The intra quantization and inverse quantization processes can be described by the following (integer arithmetic) expressions:

$$qDCT = \frac{16 \times DCT + sign(DCT) \times quantizer\_scale \times Q}{2 \times quantizer\_scale \times Q}$$

$$DCT = \frac{2 \times qDCT \times quantizer\_scale \times Q}{16}$$

The reconstructed valued are limited to the range $[-2048, 2047]$.

Remember that the DCT, $qDCT$ (quantized DCT coefficient) and Q (quantization divisor) variables stand for general elements of 8×8 arrays, so in an actual software implementation, these will each probably be doubly indexed arrays. The $quantizer\_scale$ variable represents a single value for each block of coefficients.

Inter Coding

Inter coded macroblocks default to using a flat quantization matrix because energy in the high-frequency residuals does not necessarily correspond to energy in high-spatial frequencies in the original image. It could arise from blocking artifacts or poor motion compensation. The default quantization value defined in the MPEG-1 standard is to divide each DCT coefficient by 16 (although a coder can include a full 64-element non-intra quantization table in the bitstream if it wishes). However, in distinction to the intra case, the result is rounded towards zero (the fractional part of the division result is discarded). This results in a wider interval around zero (called a *dead zone*) where small coefficients get quantized to zero. The inter quantization and inverse quantization processes can be described by the following (integer arithmetic) expressions:

$$qDCT = \frac{16 \times DCT}{2 \times quantizer\_scale \times Q}$$

$$DCT = \frac{(2 \times qDCT + sign(DCT)) \times quantizer\_scale \times Q}{16}$$

The reconstructed valued are limited to the range [-2048,2047]. In this case the Q variable usually takes the single value 16.

In this section we have described two different types of division, with rounding towards the nearest integer or rounding towards zero. A third type is also used in the standard involving rounding towards $-\infty$. The effects of the three different types, along with the symbols used to indicate them in the standard are shown in Table 9.1

9.3.4 Picture types and picture sequences

For source video material in CCIR-601 format, CIF is normally used as the image format that is coded. As described above, all input pictures are not treated in the same way: they can be encoded as I-pictures, P-pictures or B-pictures, depending on the requirements of the encoder from picture to picture. As B-pictures depend on a source picture later in time than their own position, they cannot be encoded until after this later picture has arrived. This is also the case at the decoder: a B-picture cannot be

Table 9.1: Integer divisions used in the MPEG-1 standard.

Operation	*Result*	*Operation*	*Result*	*Operation*	*Result*
Rounding to nearest integer		Rounding towards zero		Rounding towards $-\infty$	
4//4	1	4/4	1	4 DIV 4	1
3//4	1	3/4	0	3 DIV 4	0
2//4	1	2/4	0	2 DIV 4	0
1//4	0	1/4	0	1 DIV 4	0
(-1)//4	0	(-1)/4	0	(-1) DIV 4	-1
(-1)//4	-1	(-1)/4	0	(-1) DIV 4	-1
(-1)//4	-1	(-1)/4	0	(-1) DIV 4	-1
(-1)//4	-1	(-1)/4	-1	(-1) DIV 4	-1

reconstructed until a later P-picture that it depends on has been reconstructed. Hence it makes sense to transmit this later picture, which is encoded as a P-picture *before* transmitting the B-picture that depends on it.

In MPEG-1, a collection of interdependent B- and P-pictures and an I-picture are grouped together into a *Group of Pictures* (GOP), which is the basic unit for random access. The first picture in a GOP must naturally be an I-picture since this is the only type of picture from which decoding can start afresh. The GOP length is thus the distance between two I-pictures in a sequence. For applications where random access, FastForward or Rewind traversals of the sequence are required, GOP lengths should be short. Ideally, GOPs should start at the beginning of each new video shot.

A typical encoding of pictures of a video sequence input into an MPEG encoder with a GOP length of 15 might be as follows:

display order	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
picture Type	B	B	I	B	B	P	B	B	P	B	B	P	B	B	P
stream order	1	2	0	4	5	3	7	8	6	10	11	9	13	14	12

The first picture decoded from the stream is picture 2 (stream order 0). This (and indeed any I-picture) is decoded independently of any others.

The next two pictures decoded from the stream are the B-pictures 0 and 1. If this is the very beginning of a video sequence, the encoder may have encoded these B-pictures to depend on picture 2 (an I-picture) only (even though they are nominally bi-directionally predicted pictures). On the other hand, if this is the beginning of a section cut from a larger MPEG-1 compressed stream, the B-pictures 0 and 1 may originally have been encoded to depend on a previous reference picture (I or P) not present here. In this case, the editing program may include these pictures in the stream, but should set the *broken-link* flag in these two cases to indicate that they cannot be decoded. A *closed GOP* is one where all the pictures in the GOP are predicted only from other pictures in the GOP, and without reference to I or P pictures outside the GOP. An *open GOP* may have pictures that depend on others outside the GOP. Which type is the sequence illustrated above?

The next picture out of the stream is P-picture 5 (stream order 3). This is predicted from picture 2 and motion compensated residuals coded (usually). Following this we have B-pictures 3 and 4 (stream order 4 and 5 respectively). These are both predicted on the basis of an interpolation of picture 2 (I) and picture 5 (P).

The number of B pictures between an I-P or P-P pair is variable, and can be zero. For example, a user or user application could choose to code a video sequence in the format

I I I I I I I I I I ...

This is good for editing and FF/FR, but offers relatively low compression. (Essentially it is equivalent to, though not bitstream compatible with, "Motion JPEG").

Another possible coding sequence is

I P P P P P P I P P ...

This offers moderate compression. (It is roughly comparable with H.261 video coding, though again not bitstream compatible).

An observed sequence began with the ten-picture GOP

display order	0	1	2	3	4	5	6	7	8	9
picture Type	I	B	B	P	B	B	P	B	B	P
stream order	0	2	3	1	5	6	4	8	9	7

and continued with the twelve-picture GOP

display order	10	11	12	13	14	15	16	17	18	19	20	21
picture Type	B	B	I	B	B	P	B	B	P	B	B	P
stream order	11	12	10	14	15	13	17	18	16	20	21	19

9.3.5 Structure within pictures

Slice

- It is a group of consecutive macroblocks.
- It acts as a limit on channel error propagation in the reconstructed picture.
- For a noisy channel use shorter slices.
- First slice begins at the first picture macroblock.
- Last slice ends on the last picture macroblock (the slice(s) fill each picture).
- Otherwise a slice can begin and end on any macroblock.
- A slice can consist of anything from 1 to 396 macroblocks (CIF pictures) but is often configured to be 22 macroblocks (16 CIF image lines in a 352x288 picture).
- Each slice starts with a *slice start code*, a *position code* and a *quantization step size*.
- This slice header information (minimum 32 bits) can be a significant overhead.
- Many encoders fix the slice pattern, but the standard allows it to be adapted to the requirements of a picture, or a user (channels or applications).

Macroblocks

A macroblock is a 16×16 square array of coloured pixels represented by the Y0,Y1,Y2,Y3,U,V structure of 8×8 monochrome pixel blocks. A macroblock address is a relative address expressed as an increment from the last macroblock (the *macroblock address increment*, which is always 1 for I-pictures).

Macroblock Types:

Skipped Macroblock The decoded values for this macroblock are taken as is from the same macroblock position in the previous picture, no motion vector or coded residuals are transmitted. Macroblocks in I-pictures are never skipped. Skipped macroblocks in P pictures have a motion vector of zero, skipped macroblocks in B-pictures have the same motion vector and the same direction of prediction as the previous macroblock, which cannot be intra coded.[1]

Intra Macroblock Prediction from the previous picture is not used (regardless of the type of I, P or B picture involved) This is like any block in an I picture.

Inter Macroblock Full motion compensated prediction is used.

[1] Skipped macroblocks are one of the secret weapons that MPEG-1 has in the fight to reduce data. Once the decision is made by the coder that it is acceptable to skip a macroblock, the corresponding video data takes little or no further space in the coded bitstream because its presence is *implied* in a macroblock address increment that skips this macroblock, rather than being directly coded.

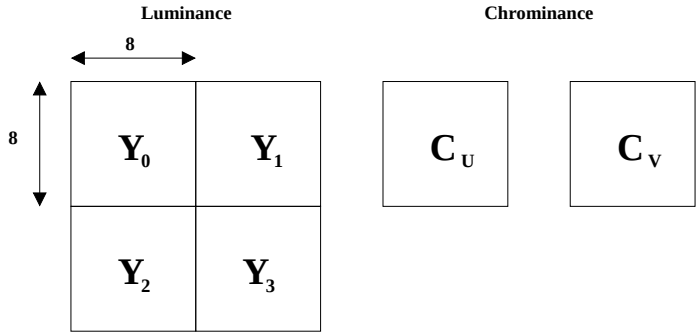


Figure 9.8: Diagram of the standard macroblock structure. The luminance components of a 16×16 pixel image block are structured as a sequence of four 8×8 blocks. The colour components of the same 16×16 pixel image block are represented by a pair of half-resolution 8×8 data blocks, each one representing one colour component for the original 16×16 pixel area. (The sampling scheme is often referred to as 4:1:1).

Blocks

A block is an 8×8 square array of monochrome pixels that can represent luminance (Y) or colour (U,V) pixel values. Blocks can be skipped within a macroblock and macroblocks can be skipped within a slice, but the first block in any slice must be coded.

9.3.6 Motion Estimation

In MPEG-1, motion estimation is an integral part of the codec. It is expected to be able to deal with high motion video content so the search range is larger than for H.261.

The region that is matched between two pictures in MPEG-1 for the purposes of motion estimation is a macroblock and the same values for motion displacements are used for each pixel in a macroblock. Thus the same motion vector is used for all four luminance blocks in a macroblock, while the corresponding displacement for the two chrominance blocks is scaled down by a factor of two. The 16×16 motion matching region is a tradeoff between increasing coding efficiency (having smaller regions) and the overhead in representing motion vectors (having fewer regions). In MPEG-2, the bit rate is not as critical as MPEG-1 and the motion matching region is set at 8×8 pixels. Thus in MPEG-2 it is possible to have different displacements for the individual blocks in a macroblock.

Between an I- or P- picture and the next P- picture, there will usually be at least two B-pictures. Hence large movements from picture to picture would imply large motion search ranges between an I or P and the next P. This can be avoided by doing motion estimation from picture to picture in the picture sequence I B B P, calculating the three motion vectors V_1 (for I to B), V_2 (for B to B) and V_3 (for B to P). The I to P motion vector is then $V = V_1 + V_2 + V_3$.

After the best location for a motion estimation block match is found to a pixel resolution, a further eight-position half-pel-shift motion estimation block search is carried out. To do this, previous-picture pixel regions of macroblock size are interpolated using a linear interpolation in each of the four cardinal directions and bi-linear interpolation in the four diagonal directions. The coder can then decide if it is worthwhile coding the larger half-pel motion vector or if an integer-pel displacement will suffice.

Note that the standard does not indicate how the coder should implement the block motion displacement estimation process. The simple, but computationally expensive, approach is to do a full search of all possible displacements within the range of the maximum displacements allowed, using, for example, the *mean absolute difference* (MAD) to compare the relevant sets of pixels in the two pictures. Fortunately there are alternatives that provide a faster location of a good, if not actually the best, match within the search region.

For B-pictures, the coder has a choice of using either the forward motion vector, the backward motion vector or a combination of the two. Where bi-directional prediction *is* used, a simple average of the two

motion compensated macroblock-sized blocks of pixels from the earlier and later pictures is taken

$$\texttt{pel}[i][j] = (\texttt{pel\_forward}[i][j] + \texttt{pel\_backward}[i][j])//2$$

and the current macroblock is subtracted from it. It is speculated that part of the benefit of using bi-directional prediction is that noise in the two reference pictures gets averaged, so that the result is closer to the part of the original scene captured in the current picture. This might also explain why the scheme does not weight the average towards the temporally closer reference picture.

Coding of Motion Vectors

While it is not always the case, there will often be large areas of a scene being videoed where macroblock matching will give consistent displacements for neighbouring macroblocks. This is particularly the case for a camera pan, where apart from new regions appearing at one side of the frame and parts of the scene disappearing at the other side, much of the difference from one video frame to the next could be described as a parallel shift in the opposite direction to the pan. So in general, adjacent motion vectors will be highly correlated, and consequently predictive coding of motion vectors from one macroblock to the next is a significant advantage. In MPEG-1, this is implemented as differential coding of motion vectors within a slice, with the prediction vector being set to zero at the start of each slice (P,B), at each intra-coded macroblock (P,B), at each skipped macroblock (P only), and at each macroblock with zero motion vectors (P only).

Motion vectors consist of a motion displacement in the horizontal direction (right = positive) and one in the vertical direction (down = positive), and can be forward (prediction from previous pictures) or backward (prediction from future pictures). In this explanation we omit the time (forward, backward) and direction (horizontal, vertical) qualifiers.

The maximum motion vector size (effectively the motion vector dynamic range) is coded on a picture-by-picture basis. The motion vector coding scheme used in MPEG-1 allows long range motion (such as ± 64 pels) to be represented where necessary, without introducing an unnecessary bit overhead when coding pictures that only have short range motion (such as ± 16 pels). The `f_code` picture header parameter effectively describes a radius shared by horizontal and vertical motion vector components. However, subsequent to the standard it has become industry practice to have a greater horizontal search range than vertical, since motion tends to be more prominent across the screen than up or down. To reflect this, MPEG-2 has separate horizontal and vertical range specifiers (two versions of `f_code`).

The motion-related variables coded in the MPEG-1 video stream are

- `full_pel` (1 bit): in the picture header — flag for full-pel or half-pel motion estimation.
- `f_code` (3 bits): in the picture header — values in the range 1-7, zero forbidden.
- `macroblock_motion`: derived from `mb_type` — flag to indicate presence of motion vector for macroblock.
- `motion_code` (1-11 bit VLC): in the macroblock header — decodes to a value in the set $\{-16, ..., 16\}$ — is related to the principal part (see below).
- `motion_r` (1-6 bits — size determined by *r_size*): in the macroblock header — is related to the residual part and will be a value in the range $\{0, \ldots, (f-1)\}$ (f is described below).

The relationship between `f_code`, f, and the maximum motion vector range is given in Table 9.2. Because the coding of motion vectors can happen for each macroblock, it is a significant overhead in the scheme, and every effort was made to reduce the number of bits required. Now, each motion displacement (MD) is prediction coded:

$$dMD = MD - PrevMD$$

which would normally increase the dynamic range of dMD over MD by one bit. This additional bit was saved (eliminated) by utilizing the fact that the range of a motion vector is constrained for each picture. The following example from Part 2 of the MPEG-1 standard illustrates the point.

Suppose a slice has the following set of motion vectors (with `full_pel=1`):

Table 9.2: Relationship between `f_code`, f, and the maximum motion vector range

			Motion Vector Range $-16 \times f \leq dMD < 16 \times f$	
`f_code`	f	Modulus	`full_pel=1`	`full_pel=0`
1	1	32	-16 ... 15	-8 ... 7.5
2	2	64	-32 ... 31	-16 ... 15.5
3	4	128	-64 ... 63	-32 ... 31.5
4	8	256	-128 ... 127	-64 ... 63.5
5	16	512	-256 ... 255	-128 ... 127.5
6	32	1024	-512 ... 511	-256 ... 255.5
7	64	2048	-1024 ... 1023	-512 ... 511.5

3 10 30 30 -14 -14 27 24

Assuming that there are no larger motion vectors in the rest of the picture, an f value of 2 can be used. The initial prediction is zero, so the prediction values (dMD) are:

3 7 20 0 -44 -2 43 -3

Where the differential values are outside the permitted range of -32 ... 31 for $f = 2$, we can add or subtract the corresponding *Modulus* value, which is 64, to give the differential values to be coded:

3 7 20 0 20 -2 -21 -3

During decoding, these can be used without ambiguity to reconstruct the original motion vector values because of the range restriction associated with $f = 2$.

The coded version of each motion displacement difference (dMD) consists of two parts, the principal part and the residual. The idea is that the principal part is coded by a VLC, while the absolute value of the residual is coded as a fixed length binary value that is concatenated to the VLC. The principal part is

$$dMD\_p = motion\_code \times f,$$

where motion_code is one of $\{-16, ..., 16\}$ and the scaling factor $f = 2^{r\_size}$ is one of $\{1, 2, 4, 8, 16, 32, 64\}$, as

$$r\_size = f\_code - 1,$$

and f_code is one of $\{1, ..., 7\}$. The scaling factor f is chosen so that:

$$-(16 \times f) <= (largest\ +ve\ or\ -ve\ dMD) < 16 \times f.$$

In other words, in the coder, the signed integer part of the quotient

$$\frac{dMD + sign(dMD) * (f - 1)}{f}$$

is used to find a variable length codeword. For example, for $dMD = 3$ and $f = 2$ the quotient is

$$\frac{3 + 1 * (2 - 1)}{2}$$

which is 2 with a remainder of 0. So `motion_code` = 2, giving a VLC of 0010 from the corresponding table (not shown here), that is concatenated with the remainder 0, which is the residual.

In the division notation given above

$$motion\_code = \frac{dMD + sign(dMD) \times (f - 1)}{f}.$$

The residual is the remainder of this integer division, but is more precisely defined as

$$comp\_r = abs(dMD\_p) - abs(dMD),$$

and is coded ones complement as $motion\_r$, where

$$comp\_r = f - 1 - motion\_r.$$

Finally, the addition or subtraction of the *Modulus* loosely described above is more precisely written down as follows: if $MD < min\ value$, then $MD = MD + 32 \times f$; if $MD > max\ value$, then $MD = MD - 32 \times f$.

To decode:
residual is zero if $motion\_code = 0$ and also if $f = 1$, else

$$comp\_r = f - 1 - motion\_r.$$

$$dMD = motion\_code \times f - sign(motion\_code) \times r$$

$$MD = (PrevMD + dMD + 16 \times f)\%(32 \times f) - 16 \times f.$$

Equivalently:

$$MD = PrevMD + dMD;$$

if $MD < min$, then add $32 \times f$ or if $MD > max$ subtract $32 \times f$.

9.3.7 Elementary Video Stream Layers

The elementary video stream is structured into layers to fulfill various application and decoder objectives. The overall layer structure is illustrated in Figure 9.9. The role of each layer in the structure is made clearer by examining the allocation of bitstream elements in the header or data of the various structural elements.

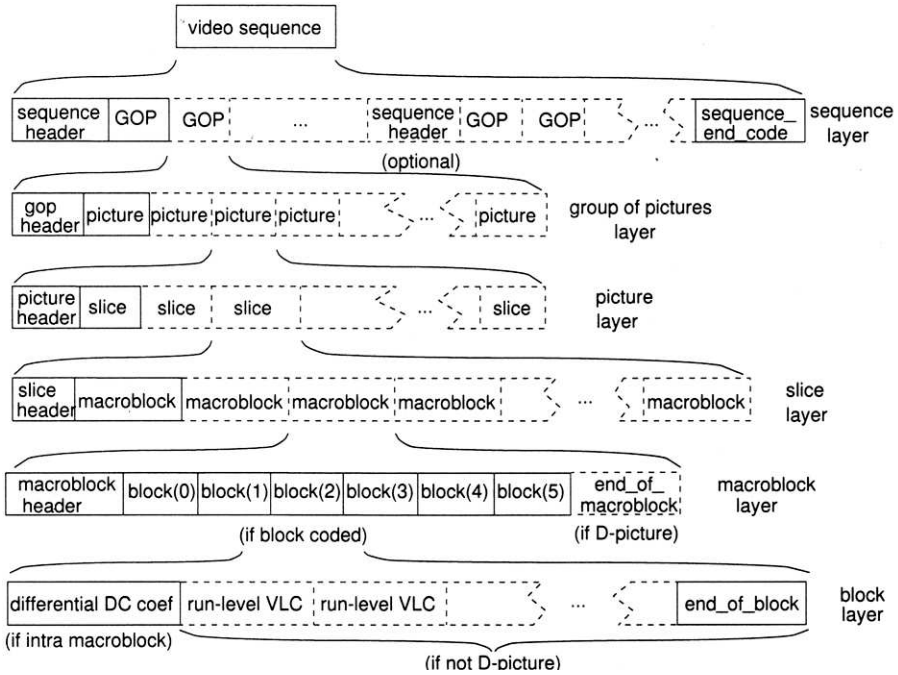


Figure 9.9: Diagram of layered structure of the MPEG-1 elementary video stream.

The first element is the sequence header, at least one of which must be contained in each elementary video stream. For random access or video editing reasons, the sequence header may be repeated, but with the exception of the quantization matrix data, their parameters should be identical to the first one. It is not unusual for a sequence to have a sequence header prior to each GOP. The standard-defined content in the sequence header is illustrated in Figure 9.10. Most of the elements are self-explanatory, with binary coded numerical values, or binary labels corresponding to values defined in the standard.

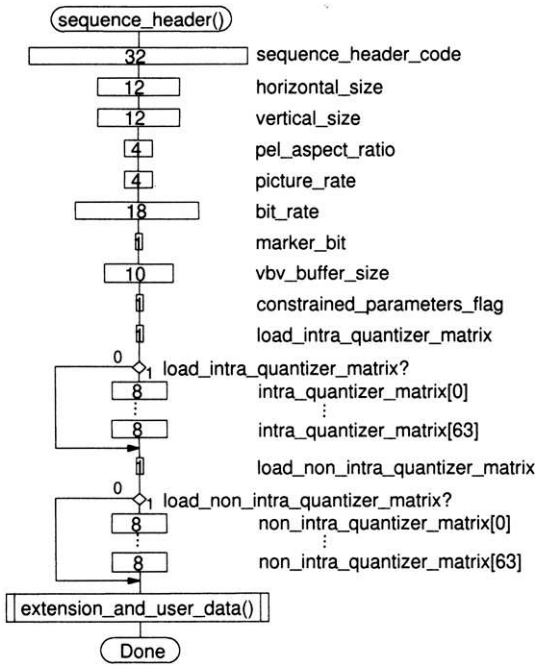


Figure 9.10: Diagram of the information contained in the sequence header.

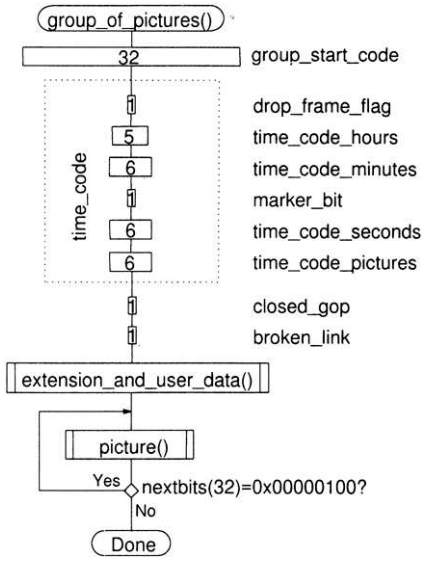


Figure 9.11: Diagram of the information contained in the GOP header.

One element of note is the marker bit with the constant value "1" which is inserted into the bitstream to avoid the accidental generation of the 23 bit binary string of zeros corresponding to a start code prefix. The *bit_rate* and the *vbv_buffer_size* values allow the decoder to allocate a sufficiently large memory buffer to decode any picture combination occurring in this sequence.

The content in the GOP header is illustrated in Figure 9.11. Apart from the timecode value that applies to the first *displayed* picture in the GOP, the header also contains the *closed_GOP* and *broken_link* flags mentioned above, and the *drop_frame_flag*. This latter is intended to convert a 30Hz picture rate to 29.97Hz by dropping pictures 0 and 1 at the start of each minute (i.e. 1798 pictures instead of 1800 per minute) but not dropping them at the start of minutes that are divisible by 10. If there is no *extension*

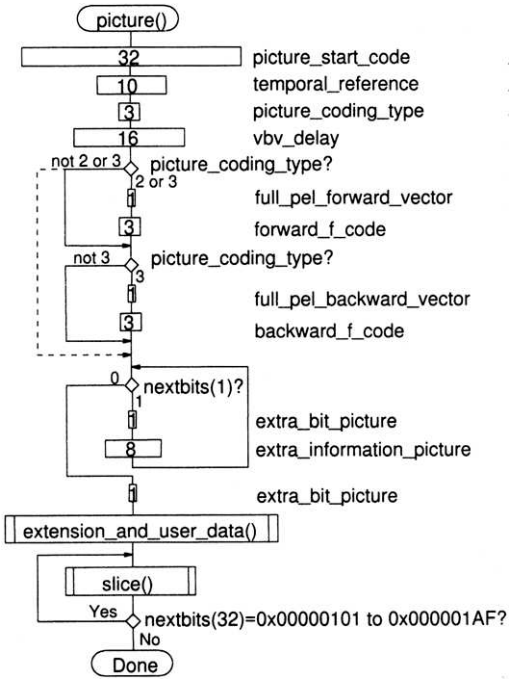


Figure 9.12: Diagram of the information contained in the picture header.

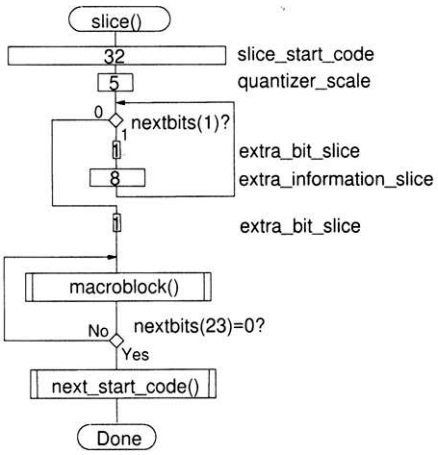


Figure 9.13: Diagram of the information contained in the slice header.

and user data, (and there never is in MPEG-1), there are usually five zero bits at the end of this header to re-establish byte alignment.

Within the GOP, we have the individual pictures, each prefixed by the picture header shown in Figure 9.12. The *temporal_reference* is the picture display order (starting at zero and modulo 1024) within the sequence of pictures following a sequence header. The *picture_coding_type* is 1 for an I-picture, 2 for a P, 3 for a B and 4 for a D-picture. Zero is forbidden (to avoid start code emulation) and all other values are reserved. For predicted pictures (P or B), motion vector scale information for the picture is included. The *extra_information_picture* is currently reserved.

The content of a slice header is shown in Figure 9.13. The main numerical element is the five-bit *quantizer_scale*, which can thus be adjusted from slice to slice. The key aspect of the slice in decoding terms is that it only requires sequence-wide parameter information to decode the pixel content covered by the slice. It does not depend on pixel content decoded in previous slices in this picture, so any errors

due to channel glitches in decoding prior slices will not effect each new slice. Of course, in P and B pictures, there will be prediction from other images, so errors in decoding them would effect the current picture. The *slice_vertical_position* is the code byte of the slice start code (values in the range 0x01 to 0xAF). It specifies the macroblock row in which this slice starts (and more than one slice can start on the same macroblock row).

The next level in the hierarchy is the macroblock information shown in Figure 9.14. The *mac-*

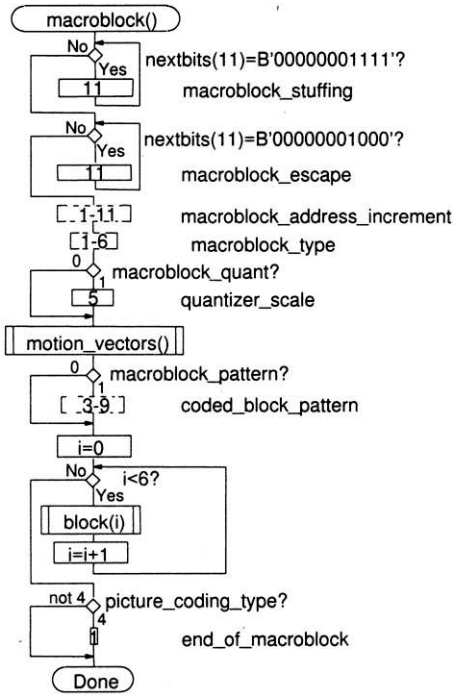


Figure 9.14: Diagram of the information contained in a macroblock.

roblock_stuffing is a fixed 11-bit pattern (0000 0001 111) that can be inserted into the stream by the encoder to increase the output bitrate to that required by a storage or transmission medium. It is discarded by the decoder.[2] The *macroblock_address_increment* is a VLC value giving the difference between the address of this macroblock (*macroblock_address* is a variable giving the absolute position of the current macroblock) and *previous_macroblock_address*. If the difference to be coded is to be greater than 33, this parameter is preceded by the 11-bit *macroblock_escape* code 0000 0001 000. Increments greater than 66 will be indicated by two *macroblock_escape* codes. *previous_macroblock_address* is a variable giving the absolute position of the last non-skipped macroblock, except at the start of a slice, where it is set as

$$previous\_macroblock\_address = (slice\_vertical\_position - 1) \times mb\_width - 1,$$

in other words, the right-most macroblock on the previous macroblock row (whether it was coded or not) The macroblock position is then computed as

$$mb\_row = macroblock\_address/mb\_width$$

$$mb\_col = macroblock\_address\%mb\_width.$$

Notice that the five-bit *quantizer_scale* may reappear in the macroblock header and can thus be adjusted from macroblock to macroblock if necessary. The *quantizer_scale* is a key parameter in the

[2]Because of problems that the unlimited looping involved in macroblock stuffing causes for hardware implementations, because the notion of adding in useless stuffing bits is anathema to data compression and because any extended requirement for stuffing bits is an indication of a problem with the buffer control algorithm, the use of macroblock stuffing is actually outlawed in MPEG-2

moment-to-moment control of the bitrate being generated in the encoder and the quality of the video information being encoded in the stream.

macroblock_type is a vlc giving the method of coding and the macroblock content as shown in the tables below.

Macroblock types in I-pictures

TYPE	VLC	QUANT
intra-d	1	0
intra-q	01	1

Macroblock types in P-pictures

TYPE	VLC	QUANT	INTRA	MOTION FWD	CODED PATTERN
pred-mc	1	0	0	1	1
pred-c	01	0	0	0	1
pred-m	001	0	0	1	0
intra-d	0001 1	0	1	0	0
pred-mcq	0001 0	1	0	1	1
pred-cq	0000 1	1	0	0	1
intra-q	0000 01	1	1	0	0
skipped					

Macroblock types in B-pictures

TYPE	VLC	QUANT	INTRA	MOTION FWD	MOTION BKD	CODED PATTERN
pred-i	10	0	0	1	1	0
pred-ic	11	0	0	1	1	1
pred-b	010	0	0	0	1	0
pred-bc	011	0	0	0	1	1
pred-f	0010	0	0	1	0	0
pred-fc	0011	0	0	1	0	1
intra-d	0001 1	0	1	0	0	0
pred-icq	0001 0	1	0	1	1	1
pred-fcq	0000 11	1	0	1	0	1
pred-bcq	0000 10	1	0	0	1	1
intra-q	0000 01	1	1	0	0	0
skipped						

9.3.8 MPEG-1 Audio

IS 11172 Part 3

- Three "layers" of increasing complexity — only one of the three is used at any one time
- Uses psychophysical properties of human hearing, specifically simultaneous masking, where the presence of one audio signal can mask the perception of a smaller signal nearby in time and frequency.
- Audio data is segmented into windows (blocks) 384 samples wide
- Layer I and II use a filter bank to decompose each window into 32 subbands with width of 750Hz for a sampling rate of 48kHz. Each subband is decimated to a sampling rate of 1.5kHz per subband and 12 samples per window. The FFT of the audio input is used to compute a global masking threshold for each subband and a uniform quantizer is used to minimize audio distortion at the required bitrate. Layer II uses a higher resolution FFT, finer quantization and a more efficient way of sending scalefactors.
- Layer III (aka MP3) uses pre-echo filtering and a modified DCT of the subbands to get finer frequency subdivision. It also uses non-uniform quantization, entropy coding and dynamic window switching for better time resolution.

9.3.9 MPEG-1 Systems Layer

ISO 11172: Part 1

- description of how audio, video and data are combined into a single data stream,
- consists of a compression layer and a systems layer, the latter of which is a structure for carrying and synchronising the streams in the compression layer.

Systems Layer:

- It is a description of a packet structure for multiplexing audio and video data in a single stream and keeping them synchronized.
- It consists of two sub-layers called the *pack* and *packet* layers, that are respectively concerned with multiplex-level parameters and elementary stream-level parameters.
- It is responsible for initialization of buffering for playback startup, and for preventing underflow and overflow of buffers. Underflow occurs when the decoder removes data too quickly from the buffer (poor compression). Overflow occurs when the decoder does not remove data quickly enough from the buffer (good compression). These are controlled by the decoder time stamp (DTS). The presentation time stamp (PTS) controls the synchronization of audio and video.
- It is responsible for buffer management.
- A pack consists of a pack header giving the systems clock reference and the bitrate of the multiplexed stream, followed by one or more packets.
- Each packet has its own header, plus the elementary data that it carries. A packet will carry one of video, audio or textual data only, not any combination of these.
- Up to 32 audio, 16 video and two data streams can be multiplexed together.
- Elementary streams (video, audio, data) are synchronized with *presentation time stamps*, which are derived from time-stamps recorded during elementary data capture.
- Because presentation time stamps and system clock references are needed to allow random access to the bitstream, they are usually sent in packets near to the header for an I-picture.

9.3.10 Digital Video Formats and picture Rates

Where data rate or compression rate is at a premium, the first step in most video coding systems is to reduce the number of pixels in each picture and often to reduce the number of pictures per second. There is the added complication of trying to deal with two analogue video standards with different characteristics.
PAL: 625 lines, 25 fps, horizontal scan frequency = 15kHz
NTSC: 525 lines, 30 fps, horizontal scan frequency = 15kHz

Image Format	Resolution	Scan Type	picture rate	Use
ITU-R(CCIR) 601 (NTSC)	720×480	Interlaced	30 fps	JPEG
ITU-R(CCIR) 601 (PAL)	720×576	Interlaced	25 fps	JPEG
ITU-R SIF (NTSC)	352×240	Progressive	30 fps	MPEG-1, JPEG
ITU-R SIF (PAL)	352×288	Progressive	30 fps	MPEG-1, JPEG
CIF (NTSC & PAL)	352×288	Progressive	30 fps	H.261
QCIF (NTSC & PAL)	176×144	Progressive	30 fps	H.261

The need to be compatible with 8×8 blocks for DCT transform coding means that the image dimensions needs to be divisible by 8, which 720/2=360 is. However, the need to be compatible with 16×16 macroblock motion compensation means that the image dimensions needs to be divisible by 16. Hence the significant pixel area if SIF is taken to be 352×288 rather than the default half resolution of 360×288.

If substantial compression is demanded, it is straightforward to reduce the resolution to one half of CIF, i.e. quarter CIF, or QCIF with 176x144 pixels. The human visual system has substantially lower spatial acuity for colour-only changes than it has for luminance-only changes. Consequently most digital video coding systems convert the original sampled RGB signal to YUV components (one luminance component and two colour components)[3] and subsample the colour components by a factor of one half down from the luminance samples. Thus in a colour CIF picture, Y will have 352×288 pixels, while U and V will have 176×144 pixels each.
Note.
$720 \times 480 \times 30 = 10,368,00$
$720 \times 576 \times 25 = 10,368,00$

Before subsampling the 720 pixels of CCIR-601 to get SIF or CIF images, it is necessary to low-pass filter the pixels to avoid aliasing. For the luminance component, a typical seven-tap filter would be:

$$[\ -29 \quad 0 \quad 88 \quad 138 \quad 88 \quad 0 \quad -29\ ]/256$$

For the colour components, a suitable four-tap filter would be:

$$[\ 1 \quad 3 \quad 3 \quad 1\ ]/8$$

9.3.11 The Constrained Parameter Flag

The `constrained_parameter_flag` is a one-bit flag in the sequence header used to indicate if the bitstream conforms to a set of constraints that significantly reduces the complexity of the coder and decoder to implement. The main constraints are:

- `horizontal_size` ≤ 768 pels
- `vertical_size` ≤ 576 pels
- Number of macroblocks ≤ 396
- (Number of macroblocks)$\times$`picture_rate` $\leq 396 \times 25$
- `picture_rate` ≤ 30 pictures per second
- `f_code` ≤ 4

In MPEG-2, standard operating points such as the one indicated by the MPEG-1 constrained parameter flag matured into the concept of "levels". The MPEG-2 conformance point corresponding to the above MPEG-1 constraints is the *Main Level*.

9.4 Other uses for the MPEG-1 bitstream

The MPEG-1 standard was clearly designed with a single objective in mind—that of compressing video data to meed storage and bandwidth requirements while maintaining an acceptable quality. However, because the bitstream contains vastly less data than the original uncompressed video frames, and because in the process of compressing the data, aspects of the video content are made explicit that were only implicit in the original video, the coded bitstream is a very useful source of information for a range of non-compression purposes such as information retrieval.

Examples of the types of data that can be directly extracted from the compressed stream are include, DC coefficients, motion vectors, macroblock type and inter-frame dependency.

DC coefficients are useful because they can be used to quickly construct a course luminance and/or colour histogram of the underlying picture. They can be used to construct a thumbnail image of the picture that is reduced in resolution by a factor of 8 or of 16 in both dimensions. They can even be added together to get a quick estimate of the overall brightness or dominant colour in a picture. Each of these can be used to help search through a large quantity of video to find some that matches a given

[3] Y=(0.299R+0.587G+0.114B)/3; U=0.492(B-Y); V=0.877(R-Y);

example or meets other information retrieval requirements. The fact that DC coefficients are relatively easy to locate in the bitstream and to decode is an additional bonus.

With regard to "Motion Vectors", it is clearly important to realize that they do not measure actual scene motion captured in the video—they are completely spurious block matches between pictures whose only purpose in compression is to minimize the difference between matched blocks. Nevertheless, useful indications of the amount of motion or the gross direction of camera motion can be extracted from the full motion vector field. The amount of motion calculated over a number of pictures can in turn be used to estimate the overall activity level of the original video and can, for example, allow a piece of video to be classified as an "action" scene or a "mood" scene, and so on.

Most information retrieval of video centers around the *video shot* as the basic unit of classification and retrieval. A *video shot* is a single piece of continuous video captured with a single camera. The fully-produced video that is usually broadcast or archived has typically been edited into a sequence of video shots, each of which conveys a fragment of the overall programme. The method for "cutting" between shots can vary widely, but standard techniques include *hard cuts*, which take place over one or perhaps two frames, and *dissolves* or *wipes*, which can take place over 10 to 20 frames, or more. The hard cuts are particularly interesting in the context of MPEG-1 compressed video, because of the inter-picture dependency that is intrinsic to the compression scheme. Thus, for example, any B picture will normally depend roughly equally on the two reference pictures (I or P) on either side of it temporally. In a B picture we would expect to see a mix of some skipped macroblocks, some I macroblocks, some P macroblocks, but mostly B macroblocks. Where this pattern is disrupted—where most if not all the references in a B picture are to just one or other reference frame—we have have an indication that a very significant disruption, such as a hard cut, has taken place in the video.

These are just some examples to indicate the wealth of information it is possible to "mine" from a compressed video bitstream for purposed other than compression.

Chapter 10

MPEG-4: a new representation of audiovisual content

10.1 Introduction

Recently, a large number of new application areas for encoded AV data have been proposed. These application areas include concepts such as interactive digital TV, mobile multimedia, virtual TV studio, networked video games, streaming internet video [1]. One common feature of all these application areas is that interaction with AV content is fundamental for the envisaged application or service. For multimedia applications of the future, it is no longer enough for the user to be a passive observer of AV data. Rather, it is desired that the user be able to react to what is seen or heard, in order to tailor the AV application or service to his/her own needs. Another fundamental concept encapsulated by these new application areas is the notion of re-using AV encoded data in an intuitive and flexible manner. Consider an application allowing the editing and manipulation of AV data where it is desired to create new AV content based on AV content in a library. In order to do this, so that the resultant AV content is seamless, the stored AV content should have an encoded representation which facilitates re-use. In this chapter, the third standardisation work item of ISO's Motion Picture Experts Group, normally referred to simply as MPEG-4, is described. This new standard addresses the needs of future AV applications and services by supporting *content-based functionalities*.

Considering the AV encoding standards described in previous chapters, it is clear that interaction is limited to the temporal aspect of AV sequences, whereby the user can linearly traverse the sequence at variable speeds (in other words, fast-forward, reverse, pause, etc. applied to an entire scene). Re-use of coded AV data is restricted as it corresponds to extracting a rectangular segment from a sequence and inserting it into another application, when perhaps only a particular object in the scene and not the entire scene is desirable in the new application. These deficiencies of the existing standards form the basis of the MPEG-4 standard. At the core of MPEG-4 is a new representation of AV data. The MPEG-4 standard has adopted a scene representation in terms of the individual AV *objects* which make up a scene, and each AV object is encoded independently [5]. As explained in the following discussion, this new representation of AV data enables MPEG-4 to achieve its main objectives of support for content-based functionalities and re-use of encoded AV data.

10.2 Objectives of MPEG-4

The major goal of MPEG-4 is to provide a new form of interactivity with AV coded data [6]. When viewing a scene, a user may wish to interact with the content of the scene for a variety of reasons. The user may require increased quality or temporal resolution for a particular object because it is more important to him/her than the rest of the scene. They may wish to remove an object from the scene because it is obscuring what they really want to see/hear. Perhaps they would like to insert an object in the scene in order to create a new scene. The exact nature of the interaction will depend on the associated application. An object-based scene representation facilitates this type of interaction. Since each object is

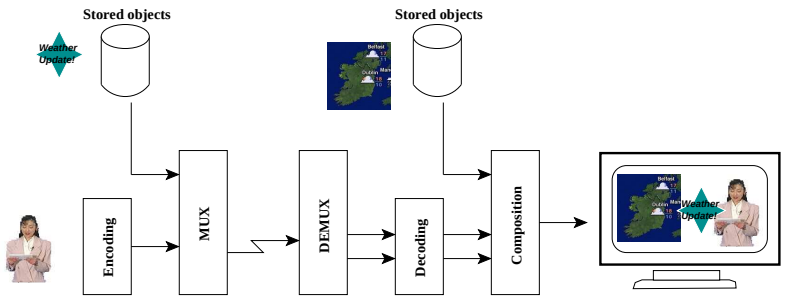


Figure 10.1: An MPEG-4 codec for visual objects

encoded independently it becomes possible to interact with that object alone, without affecting the rest of the scene. Furthermore, it allows the re-use of encoded AV data, as a stored AV object can be easily added to the scene, or the objects in the scene can be used at a later date in a different scene. Also, since the objects are treated independently, it is possible to encode each object using a coding scheme best suited to the nature of the object. By adopting an object-based scene representation, the standard effectively removes the responsibility of *scene composition* from the decoder and allows the possibility of providing it as a functionality of the user's application [5].

The standard has other objectives such as scalable coding, transmission across heterogeneous and error-prone networks, hybrid natural/synthetic content coding and coding of multiple concurrent data streams [6]. Such objectives are all also greatly facilitated by an object-based scene representation.

10.3 The Flexibility of Coding Objects

Figure 10.1 shows a complete MPEG-4 encoding and decoding system for visual objects. Input objects are encoded and may be multiplexed with other stored objects for transmission. At the decoder, the objects are de-multiplexed and decoded. A display is created by combining these decoded objects. They may be combined with other objects stored at the decoder. User interaction is allowed at any stage of the encoding/decoding process. The user may choose which objects to encode, how to encode them, and to what quality. The user can also decide which objects to decode and at what quality level to perform this decoding. Finally, the user is instrumental in creating the desired display as he/she can decide which objects are to be used to form the display, as well as how to display these objects (this is the process termed scene composition in section 10.2). Whilst MPEG-4 supports both natural and synthetic object-based audio and visual compression, for the remainder of this chapter we focus on object-based compression of natural video.

10.4 Video Objects and Video Object Planes

In MPEG-4 video compression, objects are referred to as Video Objects (VOs). A VO is a specific object in the video scene with which the user may wish to interact and which should be coded independently. A VO can be an arbitrarily-shaped object, such as a person (e.g. an anchorperson in a news programme, a player in a football match) or "thing" (e.g. a car in a Formula 1 race) in the scene or can be a rectangular object corresponding to the entire scene[1]. A VO evolves temporally through a video sequence and a snapshot of the state of the VO at a particular time instant is termed a Video Object Plane (VOP). VOPs are analogous to frames in frame-based coding techniques – each VOP is a video frame of a specific object of interest [4].

At each time instant, each object's shape (defining the object's borders) and texture (defining the "inside" of the object) must be represented. For this reason, each VOP consists of four pixel components which collectively represent the object's shape and texture. Luminance and chrominance information

[1]In which case, MPEG-4 video compression very closely resembles MPEG-1/-2 or H.261/H.263, albeit with a different set of target bitrates and more efficient coding tools

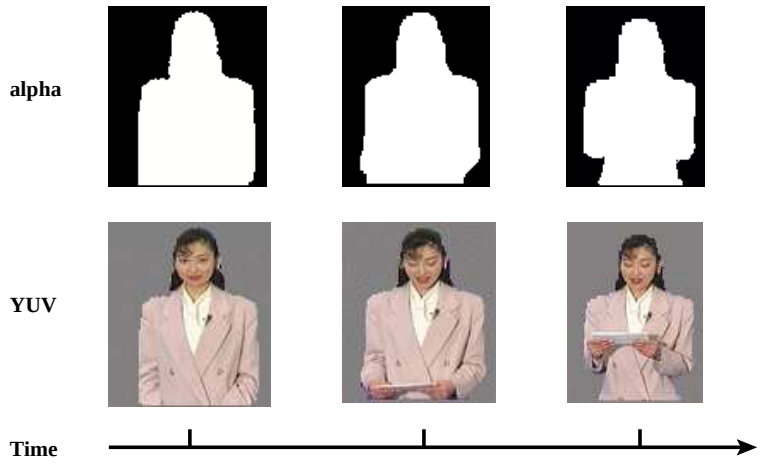


Figure 10.2: A Video Object(VO) and its Video Object Planes (VOPs)

(i.e. Y,U and V components) is used for texture exactly as in a rectangular video frame. The object's shape is represented using a pixel component termed the *alpha plane*. This is an image which defines the object's shape and transparency information at every pixel location within the frame. A non-zero value in the alpha plane indicates that that pixel position is part of (or "inside") the object, whereas a zero value indicates a pixel position which not part of (or "outside") the object.

Objects can be opaque or semi-transparent. If an opaque object were to be placed in a scene and overlaid on other objects, it would occlude these objects. A semi-transparent object, on the other hand, would only partially occlude these objects. Transparency is useful when it is required to *blend* an object (or at least its borders) into a different background (e.g. a logo or subtitles). Alpha planes are typically represented with 8 bits per pixel where a value of 255 indicates a completely opaque pixel and values in the range [1-254] indicate decreasing levels of transparency. An alpha plane consisting only of the values zero and 255 is termed a *binary alpha plane* and indicates a completely opaque object. A *gray scale* alpha plane indicates a semi-transparent object.

The concept of VOs and VOPs is illustrated in Figure 10.2. The VO in Figure 10.2 is a television announcer[2] and is represented as a sequence of VOPs at discrete time intervals. The time intervals are defined by the *framerate* of the input sequence. Each VOP has a YUV pixel component and (in this case) a binary alpha plane associated with it.

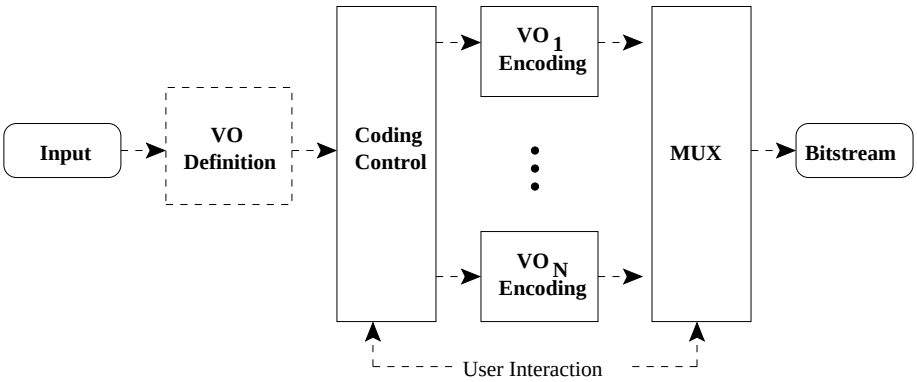
10.4.1 Generalised structure of a codec for MPEG-4 video

A diagram of the generalised structure of an MPEG-4 video encoder is shown in Figure 10.3(a). As can be seen muliple VOs can be independently encoded. Although it is not shown in the diagram, it is implicitly assumed that in the case of multiple VOs, some composition information specifying the VOs' spatial and temporal position in the composited scene is also encoded.

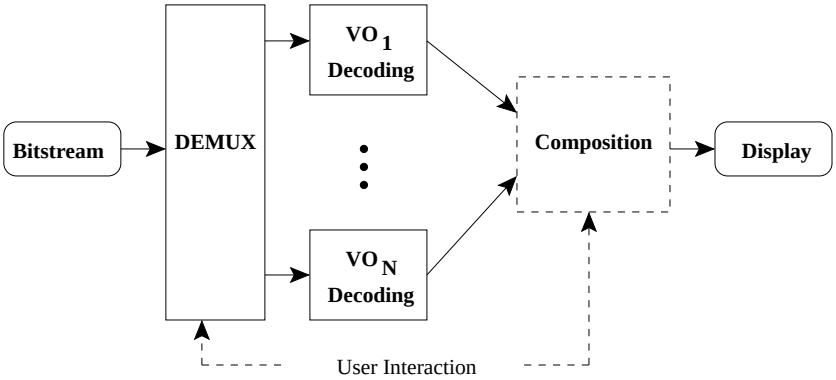
The VO definition block has the task of defining meaningful objects in a scene with which the user may wish to interact. It is the process of creating a sequence of VOPs for a particular object, which corresponds to an image processing task known as *object segmentation*. The segmentation process may be automatic (i.e. performed with no user interaction) or semi-automatic (i.e. performed with some minimal amount of user interaction) or completely manual or carried out via a chroma-keying process (i.e. blue screen technology in a studio). In any case, MPEG-4 does *not* standardise this segmentation process. MPEG-4 simply specifies a coded representation of objects, but not the way in which these objects were obtained from source video (for more information see section 10.6).

Figure 10.3(a) shows the possibility of user interaction at the encoder . For example, user interaction may occur at the coding control block whereby the user may request that a particular VO be coded to

[2]In fact, this example is derived from an MPEG-4 test sequence of a weather forecast known as "Weather Girl"



(a) Encoder



(b) Decoder

Figure 10.3: High-level structure of a MPEG-4 video codec

higher quality than other VOs. User interaction can also occur at the multiplexer whereby the user may wish to change a VO's composition information prior to transmission. Alternatively, due to bandwidth limitations for example, the user may request that a particular VO is not transmitted at all.

The generalisd structure of an MPEG-4 video decoder is illustrated in Figure 10.3(b). Each VO is demultiplexed from the bitstream and independently decoded. The composition block has the task of combining the decoded VOs to create a scene. This is actually an MPEG-4 systems issue and as such, the exact nature of this block is not outlined, except to note that user interaction may occur during composition. User interaction at the decoder may also occur at the demultiplexer. The user may request, for example, that only a subset of the available VOs[3] be decoded, thereby lightening the load on the decoder.

10.5 MPEG-4 Compression Tools

In this section, the actual tools adopted by MPEG-4 in order to encode a VOP (i.e. the contents of *one* of the $VO_{1...N}$ boxes in Figure 10.3(a)) are described. The description is limited to tools used for encoding natural arbitrarily-shaped VOPs and even then only a subset of the available tools available to MPEG-4 is explained. Since the extra information source to be encoded in MPEG-4 with respect to previous standards is the shape of the object, special consideration is given to the shape coding tools. A more complete description of MPEG-4 coding tools can be found in [1, 4, 5].

10.5.1 I-,P- and B-VOPs

In a manner similar to the picture types used in the MPEG-1 and MPEG-2 standards [3], there are three types of VOP possible in an MPEG-4 coded bitstream. The first is an I-VOP (intra VOP) which is encoded completely independently using transform coding. I-VOPs provide a random access point in the bitstream since they can be decoded without reference to a previous or future VOP. They can be used for predicting the other types of VOPs. The second type of VOP is a P-VOP (predicted VOP) which is encoded with refence to a previous I-/P-VOP using forward motion compensation followed by transform coding of the prediction residual. Both I- and P-VOPs can be used for predicting the final type of VOP known as a B-VOP (bi-directional VOP) which is encoded with reference to a previous I-/P-VOP, or a future I-/P-VOP, or both. Coding is performed using either forward or backward motion compensation or both as appropriate. B-VOPs are never used for prediction. As in MPEG-1 and MPEG-2, this necessitates the VOPs being transmitted and decoded in a different order to which they are displayed. For example, if the display order of VOPS was IBPBP... then the bitstream would be ordered as IPBPB... The different types of VOPs and the relationship between them is illustrated in Figure 10.4, where the direction of the arrows indicates the direction of prediction[4].

10.5.2 VOP Bounding

Prior to any encoding, a bounding box is placed around the VOP, which limits the amount of data to be coded for each VOP. The bounding box is the tightest rectangle which includes all non-zero pixels in the VOP's alpha plane. This bounding box is divided into non-overlapping 16×16 blocks. Each block is referred to as an alpha block. At this point a procedure known as Shape Adaptive Region Partitioning (SARP) is carried out. This is where the bounding box is moved around its origin (by convention the upper left corner) and a new origin is searched which results in a bounding box with fewer alpha blocks to be encoded (see below). The resulting block grid is applied to the texture component of the VOP and each texture block (referred to as a macroblock, as in H.261, H.263, MPEG-1 and MPEG-2) is encoded independently.

An alpha block (macroblock) may be completely outside the shape of the VOP and thus need not be coded at all. Alternatively, an alpha block (macroblock) may be completely inside the shape of the VOP. These are encoded in a manner very similar to existing block-based standards. Finally, an alpha block (macroblock) may be partially inside the shape of the VOP. These blocks need special consideration as

[3]... or indeed parts thereof – see section 10.5.6

[4]Whilst Figures 10.4,10.5 and 10.7 ostensibly depict binary alpha planes, for ease of representation, the pixel values used are 0=inside object, 128=outside object

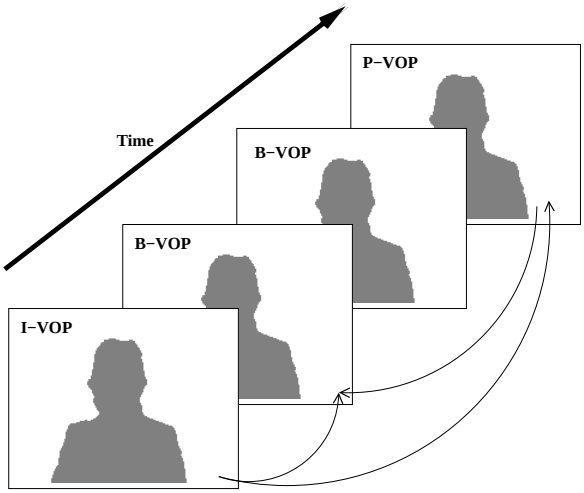


Figure 10.4: I-,P- and B-VOPs

they require the transmission of shape information, and also the use of this shape information in motion compensation and texture encoding. VOP bounding and the different types of alphablocks (macroblocks) possible are illustrated in Figure 10.5.

10.5.3 Shape Coding

There are two approaches to shape coding, and which is used depends on the nature of the alpha plane.

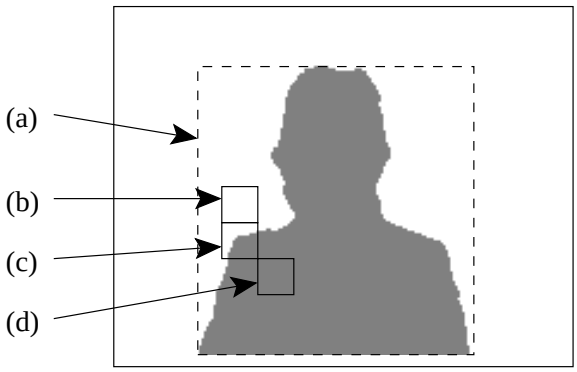
Binary shape coding

In this case, an alpha block is referred to as a Binary Alpha Block (BAB). BABs are encoded using motion compensation and Context-based Arithmetic Encoding (CAE)[5]. To generate motion vectors, a simple motion estimation procedure at full pixel resolution is carried out for each BAB. Overlapped compensation, half-pixel resolution, etc. tools are *not* employed. Each BAB's motion vector is predictively encoded using neighbouring shape or texture motion vectors. Thus, what is actually encoded for shape motion vectors is MVDs - the motion vector prediction difference. Each BAB is encoded in one of the following seven modes [5, 4, 1]:

1. MVDs = 0 and "no update"
2. MVDs != 0 and "no update"
3. all_0
4. all_255
5. intraCAE
6. MVDs = 0 and interCAE
7. MVDs != 0 and interCAE

The first two modes indicate that the shape information will not be updated, thus only motion information is encoded – where the information is the motion vector prediction residual (mode 2) or the fact that the predictor is sufficient (mode 1). The next two modes indicate BABs completely outside or inside the object respectively and thus require neither motion nor shape information to be encoded. The next mode indicate that shape information is encoded using the intra mode of the CAE algorithm – i.e.

[5]The CAE algorithm was developed in DCU by Dr. Noel Brady whilst working as a researhc assistant for Teltec Ireland



(a) VOP bounding box, (b) alpha block completely outside VOP, (c) alpha block on VOP boundary (d) alpha block completely inside VOP

Figure 10.5: VOP bounding and MPEG-4 alpha blocks (macroblocks)

encoding without reference to a previously reconstructed alpha block. The last two modes indicate that both motion information is encoded as above and shape information is encoded using the inter mode of the CAE algorithm (i.e. with reference to a previous reconstructed alpha plane).

The basic idea behind CAE is to use efficent arithmetic encoding to encode the state (transparent/opaque) of each pixel in a BAB[6]. The probability of a pixel being transparent (0) or opaque (255) depends on a pre-defined local neighbourhood or context. The point is that some configurations of 0/255 values are more likely than other. In intra mode, CAE proceeds by first computing an *context number* on the basis of the neighbourhood template illustrated in Figure 10.6(a) according to:

$$C = \sum_{k=0}^{9} c_k \times 2^k$$

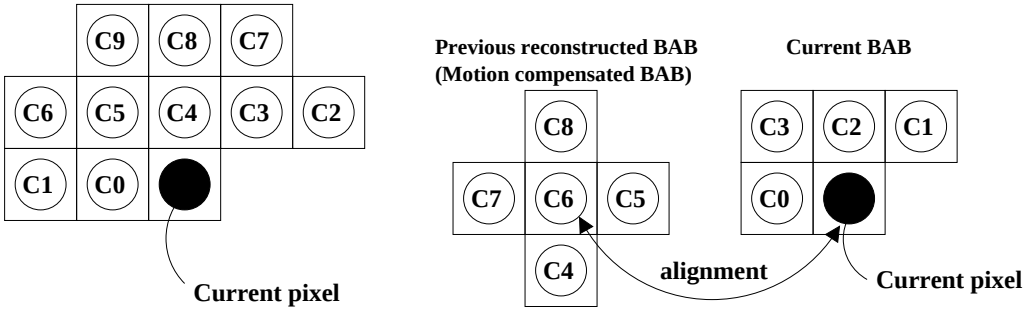
where $c_k = 1$ if that pixel position is 255 and $c_k = 0$ otherwise. This context number can then be used as an index into a pre-defined table of probabilities[7] to generate the probability of the pixel under consideration being 0 or 255. This probability is used to drive an arithmetic encoder. Figure 10.7 illustrates an example of intra CAE. In this case the context would be computed as: $2^0 + 2^1 + 2^3 + 2^4 + 2^5 + 2^7 + 2^8 = 443$ and the probability of the pixel under consideration being transparent (0) would be obtained from the intra probability table[8] as $P_{intra}[443]$, with $1 - P_{intra}[443]$ as the probability of the pixel being opaque (255). In fact, in this case $P_{intra}[443] = 0.03$ and $1 - P_{intra}[443] = 0.97$, These values are to be expected since the pixel is most likely to be opaque (255) given the values of the majority of pixels in its neighbourhood.

The inter mode of the CAE algorithm proceeds in a very similar manner except that it is attempted to exploit both spatial and temporal correlation in the shape information. This is achieved by using a different neighbourhood template. The template used is illustrated in Figure 10.6(b). As can be seen, in this case the template consists of pixels in the local neighbourhood of the pixel under consideration and pixels in the previous reconstructed binaray alpha mask (with alignment as shown).

Grey scale shape coding

A grey scale alpha plane requires coding of its two basic components: support (i.e. shape) and grey level values within that shape. To encode the shape, the alpha plane is thresholded with a value of 0 to produce a binary alpha plane which is encoded as outlined above. The grey level values are encoded

[6]The arithmetic encoder is "initialised" for each BAB - so that the decoder knows when to stop decoding
[7]Clearly, these tables must be present in the encoder and decoder
[8]Different tables exist for intra and inter probabilities due to the different contexts used



(a) Intra (b) Inter

Figure 10.6: Templates for calculating context numbers for CAE

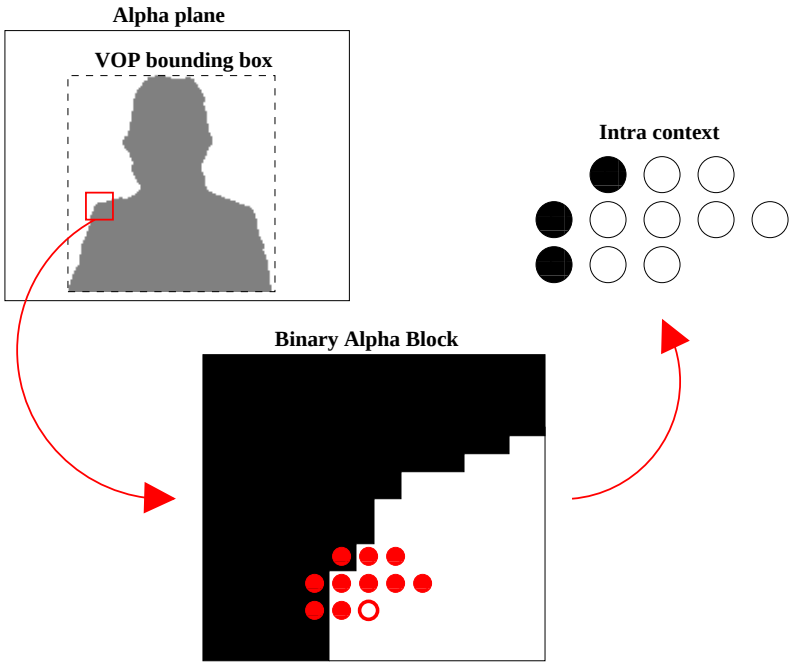


Figure 10.7: Example of Intra CAE encoding

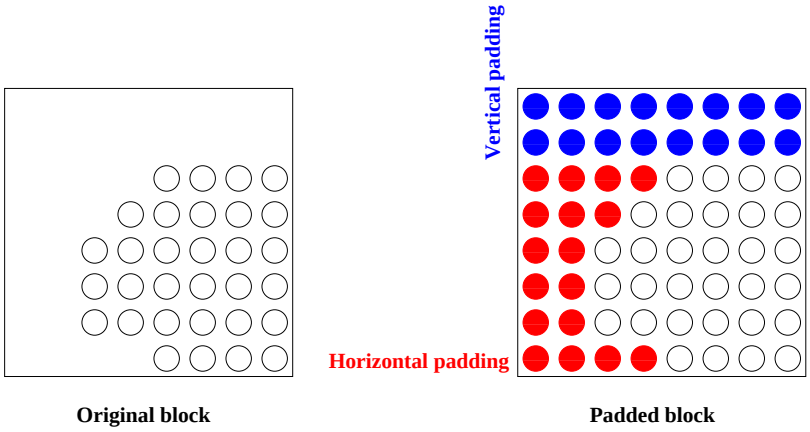


Figure 10.8: Boundary block padding

in exactly the same way as a texture luminance component with arbitrary shape. The grey scale alpha information is appended to the texture information for the associated macroblock in the bitstream [5].

10.5.4 Motion Estimation and Compensation

Motion estimation and compensation of macroblocks partially within the shape of the VOP uses two tools known as polygon matching and padding [7, 1], which are further described below. For macroblocks which are completely inside the shape of the VOP, the motion estimation procedure is very similar to that of existing standards. The estimation can be carried out at integer, half-pixel or quarter-pixel resolution. Bilinear interpolation is used to generate half-pixel resolution values, with the same interpolation procedure applied on this result to produce quarter-pixel accurate values.

The advanced prediction mode of H.263 (corresponding to four motion vectors per macroblock) is available and in this case, Overlapped Block Motion Compensation (OBMC) is used to form the prediction. The padding scheme is also used to extend the reference VOP beyond its bounding box in order to provide unrestricted motion estimation, similar to H.263 (i.e. motion vectors which can point outside the VOP bounding box). Motion vector information is predictively encoded in a manner similar to other standards, although special rules are required to cope with the different macroblock cases and the arbitrary shape of the VOP.

Padding

In order to perform motion estimation, the contents of the VOP are padded beyond the VOP boundary. This padding must be performed for 16×16 macroblocks on the VOP boundaries, but also for some blocks completely outside the shape of the VOP – this referred to as extended padding. Boundary blocks are padded first and the result is used in extended padding.

Padding for boundary blocks takes the form of pixel repetition in both the horizontal and vertical directions. Horizontal pixel repetition is carried out first. Boundary pixel values (i.e. pixels on the object's contour) are repeated in either the left or right (or both) directions as appropriate in order to fill transparent pixel locations. Where there are two candidate repetition values, the average is used. After horizontal padding, the remaining transparent pixels in the block are filled using vertical pixel repetition, where all pixel locations already filled in (i.e. either as part of the object or as a result of horizontal padding) are used. Boundary block padding is illustrated in Figure 10.8.

Extended padding is used to pad macroblocks outside the VOP which are adjacent to boundary macroblocks. To this end, pixel repetition is again employed, however, this time using the pixels at the edges of the padded boundary block. If there are more than one adjacent boundary blocks, then the padded boundary block to be used is chosen on a priority basis. The priority is illustrated in Figure 10.9. To fill the block, pixel values are simply repeated upward, leftwards, downwards or rightwards depending

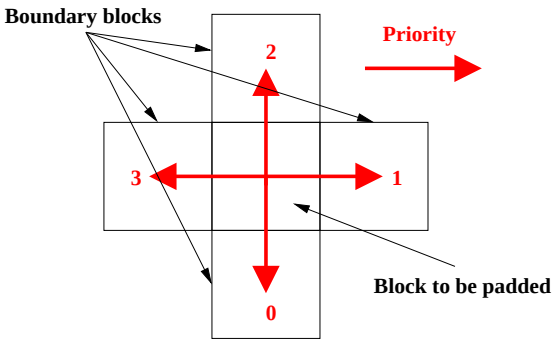


Figure 10.9: Extended padding

on the boundary block used. All other macroblocks in the VOP bounding box (i.e. not adjacent to a boundary block are padded with the value 128).

Polygon Matching

In polygon matching[9], only pixels within the shape of the VOP are considered in the matching criterion used in the motion estimation search strategy. In other words, only pixels with a non zero value in the alpha plane are considered. For example, if the Sum of Absolute Differences (SAD) was to be used as the matching criterion, then for blocks on the boundary of an object, the calculation of this metric would be modified as follows:

$$SAD(B_{curr}, B_{ref}) = \sum_{i=1}^{16} \sum_{j=1}^{16} |(B_{curr}(i,j) - B_{ref}(i,j)) \times A_{curr}(i,j)|$$

where B_{curr} is the block under consideration in the current frame, A_{curr} is the corresponding Binary Alpha Block (assuming values of 1/0 for inside/outside VOP) and B_{ref} is the block at the current search location in the reference frame.

10.5.5 Texture Coding

Texture coding consists of compressing raw pixel data in the case of I-VOPs and compressing the prediction error in P- and B-VOPs. Macroblocks completely within the shape of the VOP are coded using a conventional DCT scheme. Two alternatives exist for encoding macroblocks partially within the shape of the VOP [21]. In the first approach, a subblock is padded using the same technique as in motion estimation and a conventional DCT transform is then applied. In the second approach, a DCT transform modified to cope with arbitrarily-shaped image regions, known as the Shape Adaptive Discrete Cosine Transform (SADCT), is employed. DCT coefficients are quantized, zig-zag scanned and entropy encoded using VLCs as in H.263, MPEG-1, etc.

10.5.6 MPEG-4 Scalability

A bitstream is scalable if decoders of different complexities can decode appropriate subsets of the bitream (commensurate with decoder complexity) in order to produce complete pictures [5]. Typically a layered coding approach, consisting of a *base layer* and an *enhancement layer*, is adopted in order to produce a scalable bitstream. The Base Layer is encoded indepently and produces encoded video of particular spatial and temporal resolution. The enhancement layer, on the other hand, is encoded with reference to the base layer to produce video with enhanced (i.e. higher) spatial and/or temporal resolution. Low complexity decoders need only decode the base layer to produce viewable video, whilst more complex decoders can also decode the enhancement layer in order to produce better quality video.

[9]Sometimes referred to as Modified Block Matching

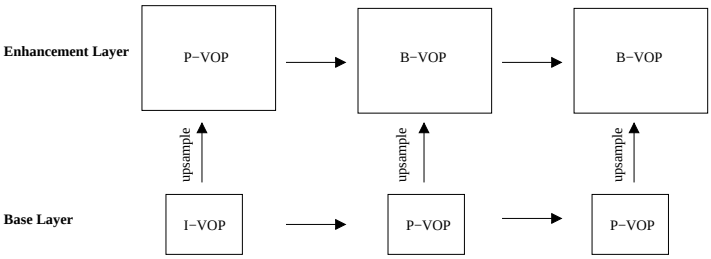


Figure 10.10: Spatial scalability

MPEG-4 supports both temporal and spatial scalability. Temporal scalabilty is supported for both arbitrary shaped and rectangular VOPs, whilst spatial scalability is supported for rectangular VOPs only. In the bitstream, each layer is represented as a Video Object Layer (VOL), where for a particular VO, VOL0 refers to the base layer, whilst VOL1 refers to the enhancement layer.

Spatial Scalability

The concept of spatial scalability is illustrated in Figure 10.10. Typically, the base layer is a down-sampled version of the source video which is coded in the normal way. The Base Layer is used to predictively encode the enhancement layer. Each decoded VOP in the base layer is upsampled and used to predict the corresponding VOP in the Enhancemnt Layer. If the upsampled VOP is an I-VOP, then this results in an P-VOP in the enhancement layer. Successive VOPs in the enhancement layer are encoded as B-VOPs with reference to both the previous Enhancement Layer P-VOP and the upsampled current base layer P-VOP.

Temporal Scalability

With temporal scalability, decoding the enhancement layer produces decoded video with a higer frame rate than the base layer. There are in fact two slightly different types of temporal scalability, which are illustrated in Figures 10.11 and 10.12.

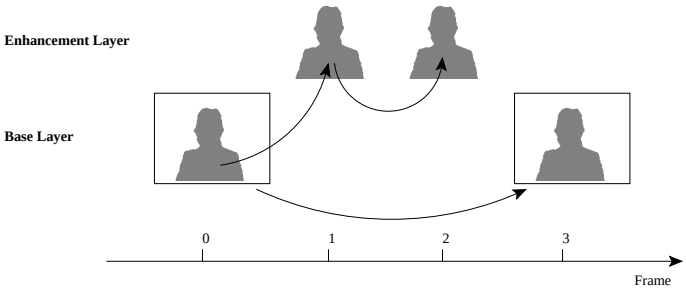
Figure 10.11 illustrates the case where the VOP in the base layer is a rectangular VOP, but where the VOP is the enhancement layer is arbirtrarily shaped. In this way by decoding both the base layer and the enhancement layer a particular object in the scene will have a higher framerate than the rest of the scene. In this scenario, there are two possibilities for encoding the VOPs in the enhancement layer – they can be coded as P-VOPs with reference to the previous VOP in the base/enhancement layer (as shown in Figure 10.11(a)), or they can be encoded as B-VOPs with reference to two VOPs in the base layer (as shown in Figure 10.11(b)).

Figure 10.12 illustrates the case where there are two VOPs in the base layer, one arbitrary shaped (VO1) corresponding to a foreground object, the other a rectangular VOP (VO0), corresponding to the background of the scene, for example. In this case, the enhancement layer provides enhanced temporal resolution of the foreground object by encoding the VOPs in the enhancement layer as P-VOPs with reference to the VOPs in the base/enhancement layer.

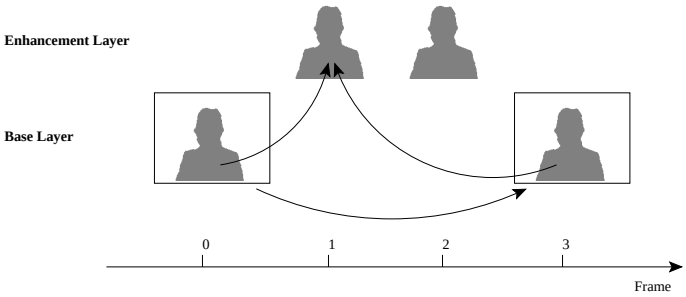
10.6 MPEG-4 and Segmentation

MPEG-4 restricts itself to standardising the way in which AV objects are represented. It does not define the way in which these objects are generated prior to the encoding process. MPEG-4 only specifies the bitstream syntax necessary to represent a video object so that it is MPEG-4 compliant. The process of obtaining video object parameters, (i.e. the VO Definition block of Figure 10.3(a)), is an image analysis task, corresponding to the segmentation of a semantic object at each time instant of a video sequence. In other words, it is the process of creating arbitrarily-shaped VOPs.

The coded representation specified by MPEG-4 is independent of the way in which the parameters describing the video object are obtained. For example, the shape coding technique divides each VOP's



(a) Enhancement layer uses P-VOPS



(b) Enhancement layer uses B-VOPS

Figure 10.11: Temporal scalability Type 1

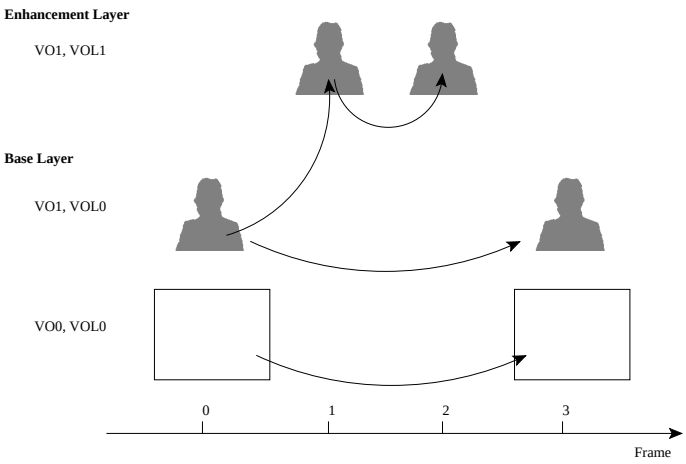


Figure 10.12: Temporal scalability Type 2

shape into blocks and treats each block independently. This is a very generic approach to shape representation which can be applied to any shape: the shape of an image region, the shape of a person present in the scene, or even the shape of text in the scene. Irrespective of the form of the video object to be encoded, its shape and texture can be represented using the techniques specified by MPEG-4. In this way, MPEG-4 combines the advantages of both segmentation-based and block-based compression. The basic coding units are objects (in the form of a sequence of arbitrarily-shaped VOPs), thereby facilitating content-based functionalities, but they are encoded in a block-based manner to facilitate many other functionalities (compression efficiency, low-delay, error robustness, etc.).

The way in which the MPEG-4 standard restricts itself to simply specifying an encoded representation for a VOP is not a limitation on the part of the standard. In fact, it is a necessary restriction to ensure its commercial success. A specification of an encoded representation of VOPs ensures interoperability between MPEG-4 products. What will differentiate MPEG-4 products in the market place, however, is the manner in which the VOPs are created. In the absence of a segmentation process during video capture (e.g. if chroma-keying technology is unavailable), an MPEG-4 application allowing the creation of VOPs in a flexible, robust and exact manner will be more attractive to users than one in which VOP creation is inaccurate, or fails for certain scene types. A further advantage of the approach taken by MPEG-4 is the fact that the standard will be able to avail of future technological advances in the field of video segmentation.

Chapter 11

MPEG-7: a description of audiovisual content

11.1 Introduction

The *Multimedia Content Description Interface*, more commonly known as MPEG-7, is a new work item within the ISO MPEG standardisation process. This work item was initiated in recognition of the increasing availability of AV data of all kinds in many different locations. In order to use this AV content in some way, it must first be located. However, due to the proliferation of this type of information (e.g. via the Internet), it has become increasingly difficult for a user to locate and obtain AV content particularly suited to his/her needs. The overall objective of MPEG-7 is to specify a description of multimedia information which can then be used to index the data for storage and subsequent retrieval. This description will be content-based reflecting the semantic content of the AV data. In this way, flexible, efficient queries tailored to a user's exact needs can be made on databases containing AV data described using MPEG-7 [8].

The MPEG-7 standard will support a broad range of applications. Application areas such as film, video and radio archives, professional multimedia editing, entertainment and broadcast media selection have all been targeted as areas which will benefit from MPEG-7 [8]. Specific applications within these areas such as journalism (e.g. searching a database for a video/audio clip of a politician or celebrity, searching for AV content on a specific historical event or a range of similar events), tele-shopping (e.g. searching an on-line catalogue), interactive TV (e.g. "clicking" on a rock singer for more information/merchanise) have already been identified [8].

A very generalised illustration of an MPEG-7 system is shown in Figure 11.1 (reproduced from [8]). This indicates that features must be first extracted from the AV data. These features are then used to form the description of the data. MPEG-7 will not standardise the way in which the feature extraction is performed. This is because interoperability does not depend on the actual feature extraction method, but on a common description of features [1]. Similarly, MPEG-7 will not standardise the search engine used to locate AV content, nor indeed the way in which the results of queries are dealt with. As with MPEG-4, these *non-normative* aspects of the standard promote competition in the market place. A good feature extraction method for a particular application will ensure a commercial edge. Furthermore,

[1] This is analogous to the way in which MPEG-4 does not standardise VOP creation

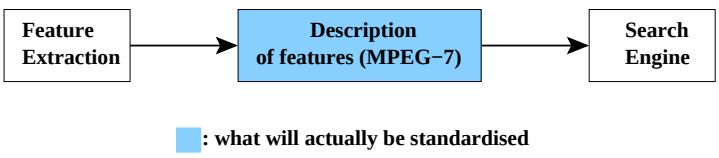


Figure 11.1: A very generalised MPEG-7 system

by not tying itself to specific feature extraction technology, MPEG-7 will be able to avail of any future advances in this field.

11.2 Objectives of MPEG-7

In this section, the main objectives of MPEG-7 are outlined in terms of:

- the data it aims to describe;
- the form that the description will take;
- the side-information required to make the description useful in a real application.

11.2.1 Multiple Data Types

Whilst a number of proprietary solutions exist for searching a database based on content, these solutions tend to be limited to a particular data type or a particular type of query. MPEG-7 intends to extend these solutions to incorporate more flexible queries based on a large range of data types such as still images, video clips, audio clips and 3-D models. The description of the AV content will be independent of the encoding process used to produce the content such as JPEG, MPEG-1, MPEG-2, MPEG-4, etc. Although the description will be independent of the encoding process, MPEG-7 will borrow heavily from MPEG-4. The object-based scene representation and subsequent encoding of MPEG-4, for example, could be a useful mechanism for enabling content-based descriptions. It should be noted that MPEG-7 will also specify a way of linking the description of the AV data to the actual location of the data itself. In this way, the AV content need not be co-located with its description.

11.2.2 Multiple Levels of Abstraction

MPEG-7 will ensure that the features used to describe AV content can be tuned to a particular application. This will be achieved by the use of several levels of abstraction for the descriptive features. The description will consist of both low-level (signal-based) features of the data as well as high-level (semantic) features. Examples of low level features for visual content are shot boundaries, keyframes, and the size, shape, colour and motion of image regions and/or objects in these shots and keyframes. Examples of high-level features are plot synopsis, characters, actors, director, etc.

Low-level features may be extracted automatically in most cases, whilst high-level features will require user interaction. Thus, depending on the *granularity* of the description required by a particular application, an MPEG-7 indexing process could be performed either in a completely automatic, or in a semi-automatic manner.

11.2.3 Non-content-based Information

MPEG-7 will support the description of *non-content-based information* which may be useful and/or necessary for the envisaged applications. These descriptive features will contain such information as:

- the form of the AV data (i.e. the coding scheme used – JPEG, MPEG-1, MPEG-2, ...)
- the access conditions (e.g. copyright on the AV data)
- classification (e.g. parental rating or classification into pre-defined classes/genres of content)
- the context of the data (in the case of non-fiction AV content, it is very often necessary to know when, where and why the content was recorded)

Bibliography

[1] A. Puri and T. Chen, *Multimedia Systems Standards and Networks*, Marcel Dekker, New York, U.S.A., 1999.

[2] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards*, Kluwer Academic, Boston, U.S.A., 1997.

[3] K.R. Rao and J.J. Hwang, *Techniques and Standards for Image Video and Audio Coding*, Prentice Hall, New Jersey, U.S.A., 1996.

[4] M. Ghanbari, *Video Coding: an introduction to standard codecs*, The Institution of Electrical Engineers, London, United Kingdom, 1999.

[5] K.N. Ngan, T. Meier, and D. Chai, *Advances in Image Communication 7: Advanced Video Coding Principles and Techniques*, Elsevier, Amsterdam, The Netherlands, 1999.

[6] R. Koenen, F. Pereira, and L. Chiariglione, "Mpeg-4: Context and objectives," *Signal Processing: Image Communication*, vol. 9, no. 4, pp. 295–304, May 1997.

[7] T. Ebrahimi, "Mpeg-4 video verification model: a video encoding/decoding algorithm based on content," *Signal Processing: Image Communication*, vol. 9, no. 4, pp. 367–384, May 1997.

[8] R. Koenen and F. Pereira, "Mpeg-7: a standardised description of audiovisual content," *Signal Processing: Image Communication*, vol. 16, no. 1-2, pp. 5–13, Sep. 2000.

Copyright