APPROACHES TO LIVE IMAGE TRANSMISSION BETWEEN WORKSTATIONS OVER LIMITED-BANDWIDTH NETWORKS

A thesis submitted to the University of Manchester
for the degree of Doctor of Philosophy
in the Faculty of Engineering and Physical Sciences

2019

By
Paulo Lício de Geus
School of Computer Science

Contents

Word Count: 999,999

List of Tables

List of Figures

Abstract

Computing environments have developed rapidly in the past few years, providing personal workstations connected via high-speed local area networks, with sophisticated graphical user interfaces. This research studies the integration of live video display into this new environment, transmitted through local area networks. Efficient image compression is required, since most local area network implementations do not provide enough bandwidth to carry digitised video in real-time. Cost reasons rule out traditional compression techniques found in video teleconferencing systems.

The importance of this thesis relies on: i) the integration of live video display in a windowed environment and the implications this causes for the workstation architecture; ii) determining what sort of bandwidth figures can be expected from a typical network implementation, for this particular use; iii) the image compression techniques suggested to achieve the desired goal. The suggested compression scheme emphasizes integer processing as opposed to floating-point solutions. It is based on progressive image transmission, with movement detection features to reduce raw image data, and vector quantization for early block image transmission. Results for an experimental compression scheme are reported, as well as directions to further improve image quality and compression rates.

Acknowledgements

I would like to thank my supervisor, Dr. Mark van Harmelen, for his guidance, friendship, for his views on how to do the research and for his help with thesis' English.

Thanks are also due to CAPES (Brazil) and Universidade Estadual de Campinas (UNICAMP) for the financial support.

A number of people also contributed to this thesis one way or another, to whom I am greatly indebted:

> Almeriane Weffort, Álvaro Garcia Neto, Cid Aimbiré, Ciro Aimbiré, Eddie (Audio-Visual Dept.), German Cabrera, Klaus de Geus, Lek Ngoh, Marcos Pessoa, Mario Wolczko, Rob Dixon and Trevor Hopkins.

This thesis was typeset with LaTeX under OzTeX , an implementation of TeX for the Apple Macintosh by Andrew Trevorrow. The body was typeset with Times-Roman, from the Computer Modern family of fonts. Figures were generated with standard Macintosh applications and included in LaTeX using a version of *dvi2ps*. The PostScript output was interpreted by Freedom of Press and printed on an HP DeskWriter printer.

He who forms the mountains, creates the wind,
and reveals his thoughts to man,
he who turns dawn to darkness, and treads the high places of the earth—
the Lord God Almighty is his name.

Amos 4:13

Porque é ele o que forma os montes, e cria o vento,
e declara ao homem qual é o seu pensamento,
o que faz da manhã trevas, e pisa os altos da terra;
o Senhor, o Deus do Exércitos é o seu nome.

Amós 4:13

to Cristiane,
who lived it all...

Chapter 1

Introduction

Technology development has caused profound changes in computing environments in the past few years. The availability of microprocessors has allowed a whole new range of applications to be developed cheaply. The most important change in the computing environment came with the availability of computing power for personal use. At the same time computer communications were being developed at a steady pace, allowing for the sharing of information among locally available computers.

The framework above evolved very quickly to high-powered personal workstations connected through high-speed local area networks. As for the benefits of computing power itself, the old "glass tty" interface with cryptic commands to memorise gave way to powerful, intuitive graphics interfaces, made possible thanks to the availability of quality bitmapped displays. The *windowed* environments are commonplace today; among classical texts in this field are [SIK+82, Pik85].

The development of new devices gave rise to numerous ways of interacting with the computer, which received investigation from a number of researchers worldwide. A good tutorial about the ergonomic factors of *interaction techniques* can be found in [FWC84]. As more computing power became readily available other more complex ways of interacting with and using the computer started to appear, departing from the old text-based interface and database ability. *Multimedia* is a new field that studies the management of information represented in different shapes, such as graphics, sound, images and video.

This thesis is about integrating live video images into a windowed environment, the video information being transmitted via a local area network. Some

assumptions are made about the hardware architecture of the workstations involved: i) the display system must be able to handle grey-level brightness at the pixel level, preferably at 8 bits/pixel; ii) the computer's design should allow hardware expansibility through add-on modules; iii) a reasonably fast local area network connection must be available. A practical choice for the latter almost inevitably happened to be the ubiquitous Ethernet. Choosing a widely used network as a model helped to better define targets for the system, but will in no way restrict the usefulness of the system on other network configurations.

Architectural issues are studied to deal with the high flow of data expected, as well as with the real-time requirements imposed by the video process. The data rates involved with live video are well above the throughput capacity of the Ethernet and of most available local area networks. Thus, suitable image compression schemes had to be devised for operation of the required facility. Maximum data throughput achievable on the Ethernet for this particular application was also determined to establish the lower bound on the required compression rate. Most of the thesis is dedicated to report experiments with real world images, thus called due to the type of scenery and to the presence of noise. Cameras and digitisers introduce a certain amount of noise into the images (that depends on the particular devices used), both spatially and temporally, making the task of compression more difficult than it first appears. Cost reasons rule out traditional techniques found in video teleconferencing systems. The suggested encoding scheme emphasises integer processing instead of floating-point intensive solutions such as transforms, towards viable implementation. The solution found trades off some image quality, particularly during high movement bursts, so that the compression rates remain high enough, and still computationally cheap.

1.1 Information Exchange through LANs

The new computing environment provided by local area networks sparkled off a number of different uses for the communications channel: mail systems (both text- and sound-based ones), file transfers, remote access, and more recently complete filesystems.

Researchers doing group work already benefit from information sharing in a transparent manner. This sort of co-operative work is grouped under Computer-Supported Co-operative Work (CSCW) systems. For more information the reader

is referred to [I. 88, KK88].

Researchers in the above condition usually find themselves in ample environments, where some work mates might be a few yards away, but others might be quite far, in another block in the building, for example. People in this situation, in need to talk for some reason, may use electronic mail or *talk*-like[1] utilities. However, this means of communication is not perfect and its use is not always desirable or suitable, depending also on each person's preferences.

An alternative to provide a more comfortable conversation would be the use of voice systems. However, there are other uses that would benefit from sound as well as vision feedback. One such example is the case where one person tries to explain a mathematical point, or something that needs a quick illustration. Today's advanced interface systems still do not allow the same action to be performed remotely by electronic means. Some systems that do allow a similar action do not do it in real-time.

1.1.1 The Concept of a Video Viewer

For the situations described in the previous paragraph, the system being investigated in this thesis might be the answer. The *video viewer* intends to show a standard TV signal on a workstation's screen, in a rate as close to real-time as possible.

Besides the assumption of having 8 bits/pixel on the display system (cited in a previous paragraph), another restriction assumed is the absence of colour. The addition of colour would only cause further problems in devising the system, lending little overall benefit to the basic idea. Nevertheless, there is no *a priori* obstacle to the addition of colour to the grey-scale system, once the latter is devised. Throughout the remaining of this thesis I shall only consider grey-scale images with a depth of 8 bits.

Other likely uses for such a system are of more general nature. It might be used in teaching activities, either in group or individual sessions. But, more importantly, the video viewer could be used by a database system. Information storage has been quickly evolving recently, keeping up with the processing power made available to personal use. From simple text-based retrieval, database systems now handle information in multimedia form, such as sounds, graphics and images. The next step in this evolution is set to be the addition of video sequences

[1] *talk* is a UNIX utility that allows instant conversation between two terminals.

as storage material. The potentials of this facility for educational purposes are enormous.

What has so far hampered the introduction of video sequences in information systems is the high computational demands required. Firstly, the storage space taken by single images is already significant; the space required even for short takes of video quickly amounts to tens of megabytes. Secondly, managing the flow of information for real-time display is no trivial task, given the data throughput most computer peripherals today are able to present.

In the case of the *Video Viewer* being devised here, the slow peripheral where the video information must come from is the local area network. As a rough guide for the purposes of a quick comparison, an Ethernet network should in theory deliver around 1 MBytes/sec. This figure is also what can be expected from a medium-sized, local hard disk on a workstation. On the other hand, standard video of 512×512 resolution at 25 frames/sec results in a data rate of around 6 MBytes/sec. The data compression rates required for continuous display of long sequences is, in both cases, at least 6:1. To perform this task in real-time demands considerable processing power, not yet cheaply available. Even if peripherals of higher speed are made available in the near future, the purpose of a compression system will still be fulfilled. Storage space will always be in heavy demand by the users and their applications, as will the throughput capacity of the communications channel.

1.1.2 Integration with the Workstation Environment

Achieving the data compression required on the digitised video signal is not enough, however. The graphics environment presented by a workstation usually provides a consistent user interface, which users will no doubt be keen to see maintained. For that to be true, the user interface system has to be in control of the display system. The module that takes care of the tiling information of the screen is usually called *window manager*. Each window presents to the user an interface to a particular application. To maintain consistency the video viewer must also abide by the rules imposed by the window manager, displaying image information only on the tiles that the window manager allows.

This behaviour imposes severe design difficulties on the display architecture. Bandwidth problems appear at several points along the data path, and must be tackled in advance when designing the workstation's architecture.

1.2 Thesis Contents

Chapter 2 presents the video viewer system with more detail and analyses the flow of data in the topology of the system. Next it reports on the Ethernet experiments performed to determine what maximum throughput figures could be expected for the intended use.

Chapter 3 presents a survey of image coding and compression techniques, with an eye on the intended application. Rough compression figures and expected image quality are cited for each technique, as well as an evaluation of their computational requirements under the assumptions made at the beginning of the thesis.

Chapter 4 describes several experiments performed with real world images. These experiments involved three main issues: i) the presence of noise and its effect on real world images; ii) the possibility of using frame-freezing to help with image compression; iii) a first compression scheme based on hierarchical coding (quadtrees).

Chapter 5 introduces the Basic Compression Scheme (BCS), which combines some features of quadtree coding with a progressive transmission scheme that attempts to spread image updating information along a number of frame periods. Results obtained are discussed and weak points are identified, so that better approaches can be sought.

Chapter 6 examines suitable alternatives from the ones seen in Chapter 3, in order to improve the BCS coding scheme. The suggested compression scheme (MDPT/VQ) is implemented and overall results for some variations of the scheme are reported. The suggested compression scheme is based on progressive transmission, but uses movement detection to effectively reduce raw image data to be coded and uses vector quantization for an early description of image blocks.

Chapter 7 ends with overall conclusions drawn from the thesis. Suggestions for future research are presented.

Chapter 2

Preliminary Aspects of a Video Viewer

2.1 Introduction

The previous chapter presented a description of the desired system. In this chapter the topology of the system is examined with more detail and the implications for the overall architecture are accounted for.

Section 2.2 includes a more thorough specification of the desired system and studies the data flow the system has to sustain. The data rate of the image source, bottlenecks and other significant points in the topology are determined.

Having preliminary information on the data rates necessary for the operation of the system, a more precise measurement for typical communication channels is required. Section 2.3 discusses and presents throughput measurements obtained from a local area network implementation. The Ethernet, being the most widely available local area network, is assumed as the target communication channel. This choice, however, does not restrict the reasoning behind this thesis and is only taken as a practical way of implementing the system. Under most situations the video data must be compressed to be transmitted through local area networks. To determine the range of compression rates required it was necessary to find out the real throughput capacity of the target network. Experiments were done and results reported.

2.2 Overview of the Desired System

2.2.1 Loose Specification

Basically, what is being examined is a way of allowing better communication among researchers in a geographically extended environment by being able to display possibly live video images on the researchers workstations' displays. Among important items to that effect the following are cited:

1. The display of images could be done on a separate, dedicated monitor, but desk space is evermore at a premium and so the need to integrate the image display onto the station's monitor.

2. Such integrated image viewer puts a strong requirement for a grey-scale display instead of more common two-level ones. Grey-scales can be simulated on a monochrome display by dithering, but the level of image quality would not be at a par with the one required for some activities this system will be targeted at, as seen in Section 1.1.1.

3. On the user interface side, it is standard practice in the workstation field to provide a more graphical approach, based on the well-known window-icon metaphor. It is then quite desirable to integrate the image viewer with the window manager currently running on the machine.

4. Current approaches to video teleconferencing still demand considerable hardware to achieve real-time performance. One of the primary targets leading this project is lower cost, implying that trade-offs are certain to be made to fulfil the desired goal. These will range from conditionally decreased image quality, less liveliness (use of frame freezing techniques), temporarily lower resolution and whatever else contributes to less computational power and less dedicated hardware, therefore helping to attain the above mentioned target, i.e. "affordable" cost, in the sense of being an optional module for the workstation.

5. Sound is assumed to be conveyed by some other established means and will not be studied here. It is assumed that the sound provided will be good enough to not disturb an observer viewing the image.

6. Despite the above mentioned trade-offs, some of the purposes the equipment is intended for require good image quality, such as, e.g., the ability of a remote user to read what some workstation user might be writing on a blackboard. It is thought that with the availability of such equipment the blackboard would be a good method to convey an elaborate thought through to the remote user, apart from the normal use in a local environment. Examples of such elaborate thoughts can easily be imagined if one tries to convey some thoughts with standard keyboards currently in use. Some graphical deficiency is covered by the mouse, but most symbols used in communicating elaborate thoughts in written form (such as mathematical symbols) are plainly unavailable on standard keyboards. WYSIWYG[1] qualities sported by some personal computers (the Apple Macintosh in particular) demonstrate a step forward, but that is still a far cry from the naturalness of the human hand when quickly jotting a thought[2].

7. The apparent paradox between items 5 and 6 must be solved somehow, and that might be the fundamental idea heading the project. It is felt that the ability to satisfy both the above requirements might be solved if the required good image quality is not present at all times.

8. To achieve image quality good enough to display details more clearly the system is targeted to the full resolution that a standard TV picture allows. The rectangular shape is usually worked at 640x512 or at 768x512 pixels, but for the sake of compatibility with equipment available and with most work in the literature it was decided to base the system and all discussions on a square frame. This translates into a 512x512 pixel frame, although some thoughts are given to 256x256 frames due to the interest in subsampling as an asset to the compression scheme to be developed (see Section 6.2.2.1.

9. By "standard TV" it is meant here the British standard, the PAL system, which works at 50 fields a second, or 25 frames a second (interlaced video).

10. The desired number of grey-scale levels is 256 (8 bits/pixel), which avoids

[1] short for "What You See Is What You Get", a feature that system can have so that the appearance of a document on screen is very similar to its printed version.

[2] a stylus and a tablet, coupled with adequate software, is another alternative that makes freehand jotting easier.

contouring distortions and is standard in image processing. The number of grey-scale levels available, however, may change according to the compression method devised and its requirements at any given moment.

2.2.2 First Proposal

Figure 2.1 presents a first proposal for the desired system, detailing some of the modules that would need to take part in the system.

In part a) the physical environment is constituted by many machines and peripherals, all sharing resources through a LAN (local area network). Since the availability of a video communication service among workstations will certainly be responsible for a high volume of data, a possible setup would have two main cables in parallel, one for normal data transfer and the other dedicated to images and/or sound.

In part b) an example session is shown, as it would look to the user. The ability to show the image sequence under control of the window manager is fundamental, in order not to break with the whole paradigm presented by a windowed graphical interface.

Part c) shows details of the connection of a workstation to the network responsible for carrying images. It would be desirable to devise a scheme based on expansion modules targeted to each workstation's bus, but this would not be an easy task for current workstation designs. After decoding the compressed data stream, the receiving module reconstructs the original sequence of images and the high volume of data is back again to be dealt with. The problem lies, of course, in how to transfer the image data to the workstation's frame buffer, at the high rate expected. It is very likely that such system will require a specific architecture design on the workstation, either by providing a high speed link to the expansion module or by allowing video overlay, i.e. the multiplexing of two video sources.

2.2.3 Details of the Proposed System

As represented in Figure 2.1a, the environment for which such system could be useful involves somewhat long distances between places where people can be working. It is not unusual to find several hundred yards between selected points.

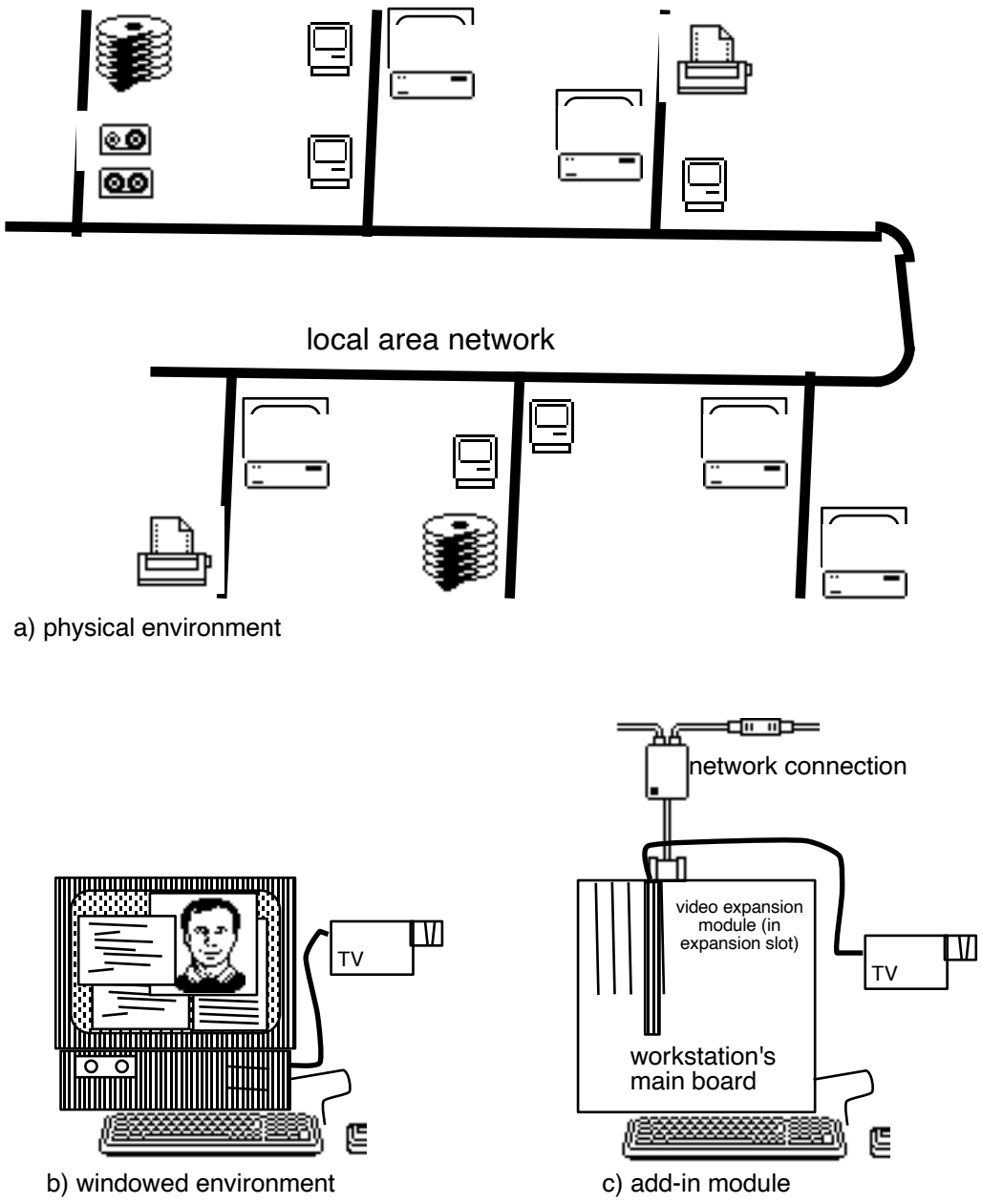


Figure 2.1: Overview of the desired system.

This fact rules out any attempt to base the system on a parallel port network, since the distances involved cannot be covered with normal parallel cables. Also, parallel cables are really not practical to make connections, particularly unexpected tappings.

In an ideal situation many stations may be connected to the image network, and, compression method warranting, the network may be carrying several simultaneous conversations between any two given users, or even between several users in a single conversation, though this requires careful thought on teleconferencing procedures.

Besides the workstations requiring grey-scale monitors, another piece of hardware that complements the system is the camera. Other uses for this imaging system may not require a camera, such as retrieving images from archives or image libraries.

Having the camera do what users want it to do is trickier yet. First of all, in a conversation the people involved are supposed to look at each other in a regular basis, at the very least. A local user will accomplish this by looking at the correct window in his monitor, but for the remote user to perceive someone as looking at him the local user must be looking at the camera. Unfortunately the monitor and the camera cannot take the same physical location; at best, careful positioning can only minimise such disturbances.

A way around this fundamental problem is to devise clever devices employing mirrors, but a compromise in text reading image quality cannot be avoided. Description of a device named "video tunnel" can be found in [AL87]. A method of conveniently inputting image so that conversations can take place the way humans do is, however, beyond the scope of this research. It is assumed that some inputting device exists, and that it performs appropriately.

2.2.4 Flow of Data Analysis

2.2.4.1 Video Acquisition Rate

This section discusses the data rates involved and requirements of the video acquisition hardware, which is sketched in Figure 2.2.

The PAL television standard has a 50 Hz vertical retrace. With the line interlaced video employed, a vertical retrace signals the start of a field (half of the total number of scan lines, picked as all odd or all even lines from the frame).

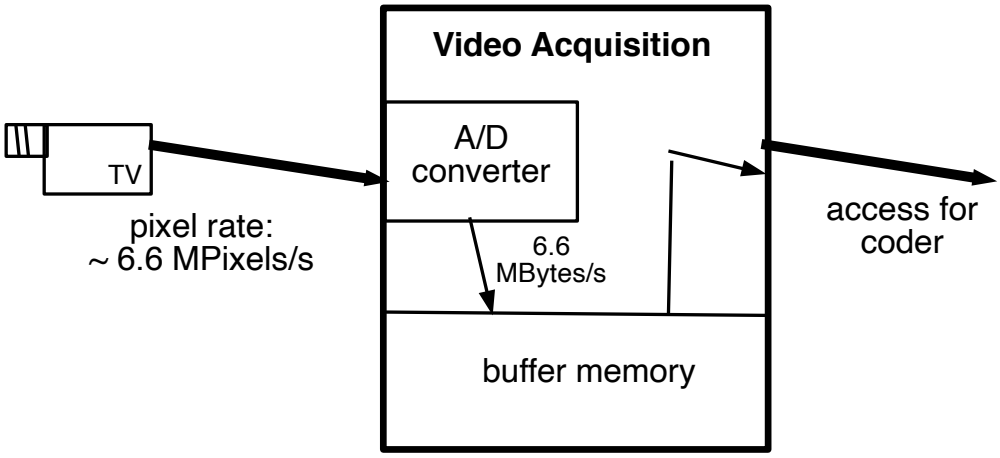


Figure 2.2: Video acquisition.

The next field is overlaid on top of the previous one to complement it and form the full frame. This results in a rate of a field every 20 ms, or a frame every 40 ms.

A frame in the PAL standard is made up of 625 scan lines, of which some are wasted in retracing the electron beam and in overscanning (the part of the image not visible on screen, i.e. the very top and the very bottom), which leaves us with around 530-550 lines [Pra78]. Of those, the central 512 lines are usually selected for display in digital systems, since rounding to a power of 2 generally simplifies most equipment and hardware implementations. The scan line time is then

$$\frac{40\ ms}{625\ lines} = 64\ \mu s$$

In the horizontal direction one can think of a scan line as a 680 dot line (anywhere between 640 and 768 dots is possible, though) to give a symmetrical pattern matching the 4:3 aspect ratio [Wat85]. Binary factors again suggest the use of a 512-dot central section of the line.

From the total scan line time, 15% is wasted during the horizontal retrace, plus some more time for the side margins. Considering a 20% total wasted time, the visible scan line takes then

$$0.8 \cdot 64\ \mu s = 51.2\ \mu s$$

The instant dot rate is then given by

$$\frac{51.2\ \mu s}{680\ pixels} \approx 75\ ns/pixel$$

Similarly the average dot rate during a line can be calculated to be (remembering that now only 512 dots are significant)

$$\frac{64\ \mu s}{512\ pixels} = 125\ ns/pixel$$

and the average dot rate in a frame to be

$$\frac{40\ ms}{512 \times 512\ pixels} \approx 153\ ns/pixel$$

Assuming then the desired 8 bits/pixel brightness range, the steady-state data rate from the video acquisition module is

$$\frac{512 \times 512\ bytes}{40\ ms} \approx 6.6\ MBytes/s$$

2.2.4.2 Image Reconstruction End

By however means the image information is sent through a LAN, the receiving station faces a formidable task. The main concern is not with the image reconstruction task itself, which is going to be performed by the dedicated receiving module, but instead with how the image information received and reconstructed to the normal data rate seen on acquisition is going to be fed to the workstation's frame buffer.

The data rate expected is theoretically the same as on acquisition, as said above, but in practice this can be lowered to more acceptable levels. It must be remembered that the workstation's frame buffer is generally already under strain on its own, in terms of memory bandwidth, to cope with the high refresh rates. These high refresh rates are normally needed to avoid flicker. Later designs are made much more easily with the availability of VRAMs (Video RAMs), which present a dual port architecture internal to the chip and allow regular (e.g. sequential) output to be performed in parallel with the normal read/write cycles. However, the main CPU is continually issuing screen updating commands, as processes keep running, demanding quite a share of the available bandwidth at

times.

The data rate of the imaging process can be lowered through intelligent updates of the image. The first and quick measure is to update only the parts of the image that changed since the last reconstructed frame. This checking is easily accomplished by the reconstruction module by keeping a full image in its private buffer. Nevertheless, the data rate can be too high and may severely affect responsiveness of the video for other activities, as far as users are concerned.

Other more elaborate techniques might be used instead and completely avoid the demand for any of the frame buffer's available bandwidth. The display of images can be done, for instance, by either analogue or digital video overlay, i.e. dynamically switching in real time the source for the video signal generator, pointing to the correct memory buffer as appropriate, according to the tiling information given by the window manager. This, of course, is only effective if the data for the image is physically located in a different memory module from the rest of the screen normally controlled by the window manager, so that low level accesses for imaging and window manager processes do not compete with each other.

In the worst case one can expect the original acquisition data rate of ≈ 6.6 MBytes/s. Most workstation designs based on a standard bus such as the VME bus access their video controllers or frame buffers through the main bus (there may be an auxiliary bus), which is usually not good for sustaining high data rates, since the main bus is meant to behave as a general access to all of the machine's resources. Typical board cycle times are currently of the order of 300 ns or longer, giving at most just over 12 MBytes/s overall bandwidth. A workstation can be performing several tasks involving many processes, such as disc and other peripheral accesses, which could be badly affected by the imaging load on the bus. Even the imaging process itself might not be performed at the required frame rate.

It is then apparent that, unless more efficient bus architectures are employed, video overlay or a similar technique must be employed to avoid tying up the machine while live video is being displayed.

2.3 Experiments with Ethernet Throughput

This section introduces the Ethernet as a representative network on which to base the project and to perform experiments to determine what net data rate can be made available in a typical existent network, given its nominal bit rate.

2.3.1 Ethernet as a Model Network

Short of delving into experimental or other rare network implementations, the ubiquitous Ethernet was chosen to carry out practical experiments, as well as to serve as the primary target on which to base the system and to set goals for compression rates to achieve.

A naive approach to the problem of getting the images appropriately transmitted is to supply enough installed network bandwidth. Without going into details (which are presented later in this chapter), such approach could base the design on an upcoming network standard called FDDI[3], which relies on fibre optic cabling to boost transmission rates up to 10 times faster than today's Ethernet. After examining the data flow figures, such system would almost certainly be considered feasible, but at the expense of keeping captive the most part of the available bandwidth in the communication channel.

Clearly this is rarely acceptable as regarding efficient sharing of expensive resources. It must be pointed out that although digital communications are evolving and providing more usable bandwidth as technology develops, the demand for bandwidth is very likely going to stay overwhelmingly higher than the installed capacity, no matter how fast technological advances are achieved. This is only one case among several others in computing which present the same kind of dispute between supply and demand, e.g. solid-state memory, fast temporary storage media and raw processing power.

It is concluded that a certain degree of compression will always be welcome, if not required, even with higher speed channels of the future. The present effort towards more efficient channel utilisation will not only allow experimental systems to be implemented but also pave the way for broader use when bandwidth becomes more generously available.

It is believed that the choice of Ethernet as a model network is both representative (by being in widespread use today) and anticipative, since restrictions on

[3] *F*ibre *D*istributed *D*ata *I*nterface.

volume of data transmitted are bound to stay in force for the foreseeable future.

2.3.2 Reasons for a Dedicated Experiment

In order to build a solid background on which to base the whole project, better knowledge of the characteristics of the chosen network (Ethernet) were needed. This is mainly concerned with the actual throughput one can expect from an existing implementation.

The need for this experiment to take place is the apparent scepticism with which most people greet the idea of very high speed data communication on an Ethernet. For although performance measurements [HM87,SDRC82] and packet-voice implementations [Gon83, NB82] do prove that an Ethernet can actually be pushed to full saturation with useful data, everyday experience shows much lower figures than the theoretical maximum bandwidth (10 Mbits/s or just over 1 MBytes/s).

Two situations that put users face to face with this issue are found in file transfers and network disc implementations. File transfers using a reliable transfer program like 'FTP' (see Section2.3.5.5) are likely to give effective rates in the range 50-100 KBytes/s, using hosts with the power of a VAX or a SUN. On network disc implementations such as SUN's system, measurements made by Stearns [Ste87] yielded effective data rates in the range 183-230 Kbytes/s.

The set of experiments described in the following sections were intended to measure transfer data rates between hosts connected via an Ethernet link, and also to try and identify causes for any apparent slowness that could possibly be found, as compared to the theoretical maximum limit. The main goal was to investigate if a simple protocol, yet suitable for the intended application, could be made to transfer data at close to the maximum limit of around 1 MBytes/s.

2.3.3 Notation Used

It is useful to define a few terms used throughout the next sections. The main characters in this play are, from one side, a *server*, and, from the other side, a *client*.

A *client* is a program running on a host that has the task of transmitting a known number of data packets (the packet size being a parameter) to a predefined host hooked off the same network, using a specified port number of the partner's

communication port structure. A client initiates requests.

A *server* is a program running on a host (set to run prior to the client program) whose task is to receive all incoming data packets through a specified port number of its structure, and to log under a unique filename the sequence number carried by each logical data packet received. A server responds to client's requests.

For *backbone* is meant the main departmental network (a single cable) that links all the main machines in this department. A network that is seen as a branch of the main one, such as the connection between a fileserver and discless workstations, will be called a *spine*.

2.3.4 Existing Work on Ethernet Channel Utilisation

Section 2.3.4.1 presents performance figures and network behaviour that are quite satisfactory for the utilisation intended for this project. However, figures more likely to be seen by an end user, as presented in Section 2.3.4.2, are not so good. To have a clearer view of the whole problem and find out the reason for such discrepancies, Section 2.3.5 and subsequent ones present an overview of network protocols and the experiments performed. These experiments were tailored closely to the requirements of this project.

2.3.4.1 Ethernet Performance under Voice-Data Transfer

One way of addressing the problem of net bandwidth available in a network is to study it under real-time constraints. Some work has been done concerning the transfer of voice data packets or combined voice and data packets [NB82, Gon83, HM87]. Voice data packets are fixed, medium-sized packets generated at a regular pace. In order to properly reconstruct the original sound the synthesizer at the receiving end has to obtain data within critical time windows. FIFOs at the reception help this but they cannot be too deep, as this would degrade the interaction lag to unacceptable levels.

The test bed was assembled as several tens of voice generating stations to measure delays and actual load on the net. The works cited above show that the delay is kept well under the specified limit and only becomes unsettled above 90% of the theoretical maximum load. Incidentally, there is then over 90% of the maximum throughput of useful data being transmitted.

Another experiment on measuring an Ethernet's performance [SDRC82] shows

that by using reasonably large packet sizes, for instance 512 bytes, the throughput is increased and stabilises at $\approx 96\%$ of the channel utilisation as the offered load is increased to 100% and above. Another point is that the load can be generated by varying the number of hosts from one to several tens without visible degradation in the channel utilisation.

The studies mentioned in the first paragraph of this section [Gon83, NB82, HM87] also go further regarding real-time aspects of the processes involved, allowing a better picture of the delays that packets are subjected to under very high offered loads. While this load is kept below some critical limit (which usually corresponds to channel utilisation above 90%), the maximum delay a packet is subjected to increases fairly steadily within acceptable thresholds. However, the maximum delay quickly increases when the critical limit is crossed. This is well related to the fairness achieved by the back-off algorithm utilised (i.e. the time for a node to make another attempt to gain control after a collision on the network is detected). It should be noted that even with unrealistic offered loads well above 100%, when real-time characteristics are poor, channel utilisation is still very high, approaching 100%.

2.3.4.2 Ethernet Performance under File Transfer

Today there are many computing sites using Ethernet-linked systems, either in fileserver configurations or as plain communication links. A few quick measurements on this department's network gives for the FTP utility data transfers in the range of 50-100 KBytes/s, which is very discouraging as regards the interests of this project. It is known, however, that file transfer protocols are quite complex, with several hierarchical levels to ensure reliability and compatibility with different operating systems and hosts.

One situation much more common nowadays is the fileserver link, in which a disc based system supplies data to a number of discless stations via a high speed link. For example, SUN servers use device drivers and Ethernet links as virtual file systems by discless workstations, and this configuration, by being fairly popular, gives rise to the opportunity for easier measurements and comparisons. Indeed, Stearns [Ste87], as part of an experiment comparing discless and standalone stations, measured user file operations and obtained the following figures for disc throughput with a SUN-3 client on a SUN-3 server during off-hours, to ensure an otherwise light Ethernet load:

$$\begin{aligned} \text{minimum data rate} &\implies 183\ KBytes/s \\ \text{maximum data rate} &\implies 230\ KBytes/s \end{aligned} \tag{2.1}$$

These results show greatly improved figures over the ones cited in the first paragraph of this section, but show also that there is a long way to go yet, as this is below 20% of the theoretical maximum of the Ethernet. Now, is the Ethernet link the bottleneck in the above cited file system?

To attempt to answer this question let us look at an experiment performed by Meyer [Mey87]. The disc throughput figures below are approximate and refer to a stand-alone SUN-3 with a SCSI disc subsystem and a VAX 8800 (no more details, but assumed to have its standard disc subsystem). These figures were taken by performing user file operations as well:

system	minimum (KBytes/s)	maximum (KBytes/s)
SUN-3	228	235
VAX 8800	401	439

These data give rise to two important points:

- by taking these results for the SUN-3 and the information at 2.1 above, a comparison can be made between two units of the same machine, each with different disc I/O subsystems (a physical disc and a network disc). If a hypothesis that the network disc performance is being affected by limitations of the Ethernet holds true, then these performance figures should show some differences; in fact, they are quite similar. There is, however, the odd possibility that the Ethernet throughput limit would be around 240 Kbytes/s, a highly coincidental event that could break the above reasoning.

- the results for a VAX 8800, known to have better I/O subsystems than SUN systems at the time those reports were written, are significantly better than those for the SUN systems. However, these results are still well below the Ethernet theoretical maximum. This indicates that some common, more significant feature is levelling file operations performance down.

Also, the VAX disc subsystem is known to give higher performance figures with low level tests. This knowledge, combined with the second item above,

reinforces my suspicions that the common operating system (UNIX) might be responsible for these systems not delivering higher throughput, either directly from a physical disc or from an Ethernet-based network disc.

Many factors contribute in this kind of measurement to the final result, e.g. disc operating system tasks that demand processing power, raw disc access time, data transfers between media and controller and between controller and operating system and so on. Similar factors take part in the network-based system: there still are transfer operations between controller and network node and between network node and operating system, although network operations tend to be inherently slower due to packet conflicts, acknowledgements and other protocol issues.

In summary, it is concluded that the figures from Chart 2.1 do not represent meaningful data to determine throughput limits for practical, end-user operations over Ethernet. To gain knowledge of such limits requires some other source of information.

The next section will look at network protocols, so that a suitable throughput measuring experiment can be devised.

2.3.5 Existing Protocols

In this section it is shown how protocols are arranged at different levels in an Ethernet, one on top of another, together with the respective overheads expected of their use. Figure 2.3 gives a glimpse of the hierarchical arrangement of these protocols.

A network such as the Ethernet provides the first level of communications, i.e. the connecting hardware and the delivery of information framed into packets of limited length [Com87]. On top of that sits IP, providing unlimited-length datagram delivery, thus allowing the realisation of higher level protocols such as UDP, TCP, FTP and others. Explanations for the above abbreviations will be given in the next sections.

2.3.5.1 Data Link Layer

Let us first take a look at the lowest protocol level (apart from the physical level), the so called data link layer, the second layer in the OSI (*O*pen *S*ystems *I*nterconnect) model [Pos82] of protocol structure. Figure 2.4 shows the packet

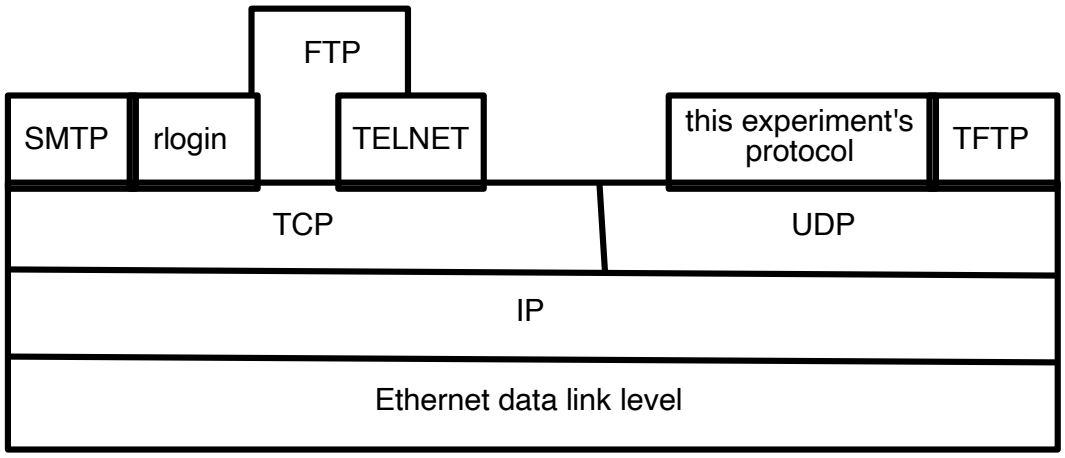


Figure 2.3: Protocol layering.

format for this level as found in the Ethernet Specifications [Eth80, SDRC82].

At this level there is a fixed overhead of 26 bytes, comprising the preamble, the header and the CRC (*C*yclic *R*edundancy *C*ode) code. The net data field is by specification required to be from 46 to 1500 bytes long, to allow for an easier distinction between a valid packet and a collision fragment (which is promptly discarded). If not enough data has to be transmitted the sender interface will pad the data field to match the requirements. The minimum packet spacing is 9.6 μs, i.e. the time that must elapse between any two consecutive transmissions on the channel. With these figures it is now possible to calculate the overhead at the data link level. Unfortunately the overhead that is of interest is dependent on the actual fragmentation of a hypothetical long transmission, therefore the following calculations are only to give an approximate idea:

$$\text{minimum packet spacing} = 9.6\ \mu\text{s} \longrightarrow 12 \text{ bytes}$$
$$\text{total overhead in bytes per packet} = 12 + 26 \longrightarrow 38 \text{ bytes}$$

The best case for the overhead (in percentage) for three data packet sizes (in bytes) is then calculated using

$$\text{overhead} = \frac{38}{38 + datapacketsize}$$

which gives

size (bytes)	512	1024	1500
overhead	6.9	3.6	2.5

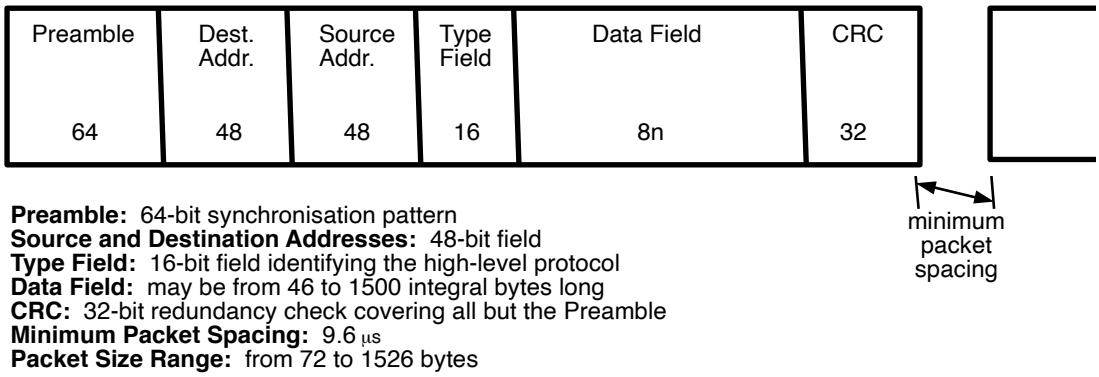


Figure 2.4: Ethernet data link layer (link-level) packet format.

The *overhead* above is given as percentage, encompassing the Ethernet data link layer.

2.3.5.2 IP Protocol

The *I*nternet *P*rotocol (IP) was developed by DARPA[4] sponsored research, together with TCP (*T*ransmission *C*ontrol *P*rotocol). IP, together with TCP (referred to as TCP/IP), is commonly used in most network implementations today, as a level-3 protocol in the OSI model [Pos82]. It is used on top of the basic level of a variety of networks such as the ARPAnet, radio and satellite networks, as well as other local networks.

The IP is a datagram[5] protocol that provides addressing and routing information in an internet, that is, a collection of interconnected networks. It deals only with data transfer between hosts in an internet; this should be compared with the role of UDP (*U*ser *D*atagram *P*rotocol) in the next section.

The IP protocol is meant to be simple and fast: it has no provisions for error control on the data portion of the message (a checksum is made on its header only); there is neither flow control nor acknowledgements of IP messages. Everything necessary to ensure a reliable ordered delivery of data has to be supplied by some higher level protocol on top of it (the TCP protocol, for instance).

[4] *D*efense *A*dvanced *R*esearch *P*rojects *A*gency.

[5] datagrams are one-shot simple messages that travel one-way and are not acknowledged, and that may also arrive in a different order than sent; thus they are inherently unreliable.

Figure 2.5a shows the IP header structure. There is an initial overhead of 20 bytes plus up to 40 bytes in case special options are used. The data field has *a priori* no restriction on its size, for the IP protocol provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

The IP header is transmitted once for each IP packet, which can be longer than the maximum allowed by the Ethernet. After all, one purpose of the IP level is to hide differences in low level packet sizes across networks. Therefore the overhead depends on a logical data packet size chosen by whoever is using this protocol.

The best case for the overhead for three data packet sizes (in bytes), supposing the minimum number of link level packets is used, as well as no special IP options, is:

size (bytes)	512	1024	4096
overhead	10.2	5.4	3.2

The *overhead* above is as percentage, combining the data link and the IP layers.

2.3.5.3 UDP Protocol

The UDP protocol, also part of the DARPA protocol suite, adds some more functionality to the previous protocol, and is commonly viewed at the same level as TCP is (level 4 in the OSI model). At the IP level the destination address only identifies a host computer; no further distinction is made regarding which process on that computer will receive the datagram. The UDP protocol extends the qualification to distinguish between datagrams arriving at a host, but directed to different processes [Com87].

Logical *protocol ports* are made available, identified by a positive number. The separation of functions then becomes clear: only the IP header identifies the source and destination hosts, and only the UDP header identifies the source or destination ports within a host. The use of conceptual layers like these isolates each module and allows for easy substitution of a given module. In the case of 4.2BSD Unix the protocol ports correspond to socket numbers; so, as IP packets are addressed to hosts, UDP packets are addressed to sockets.

The UDP packet is entirely encapsulated within the data field of the IP packet, as is the IP packet within the data link layer packet (in this case the Ethernet data

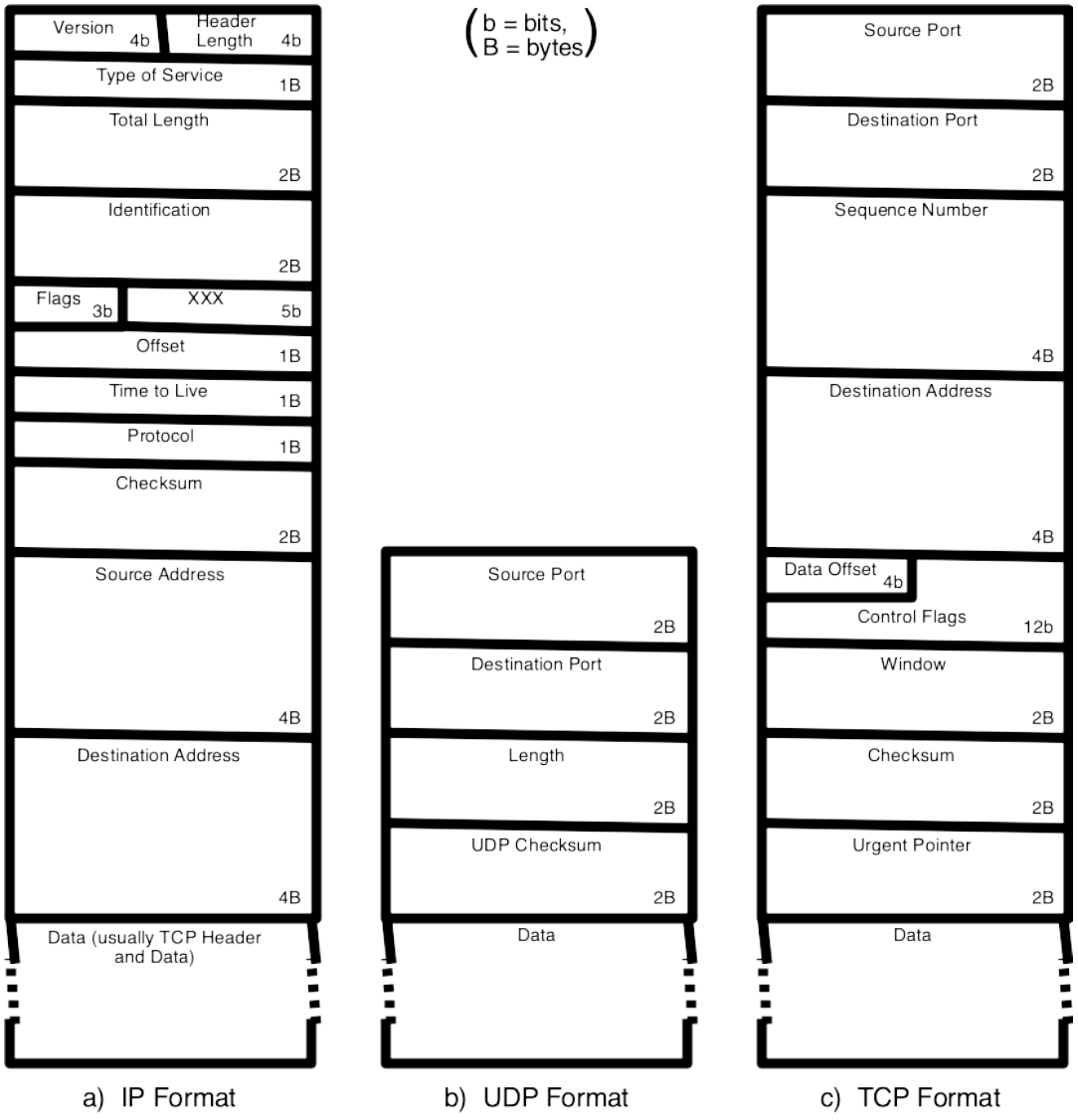


Figure 2.5: IP, UDP and TCP packet formats.

packet). UDP still provides unreliable delivery: there is no acknowledgements to make sure messages arrive safely, nor does it order incoming packets.

Figure 2.5b shows the UDP header structure. The overhead incurred is 8 bytes, and a checksum covering the whole UDP packet is made, though it can be disabled.

The best case for the overhead for three data packet sizes (in bytes), supposing the minimum number of link level packets is used, is:

size (bytes)	512	1024	4096
overhead	11.4	6.1	3.4

The *overhead* above is as percentage, combining the data link, the IP and the UDP layers.

2.3.5.4 TCP Protocol

TCP uses end-to-end mechanisms to ensure reliable ordered delivery of data over a logical connection, such as the one provided by IP [Pos82]. It uses flow control, acknowledgement windows with time out and retransmission and sequence numbers to achieve a virtual circuit path.

The TCP header is also 20 bytes long, but if options are used it can be up to 60 bytes long. TCP provides another two byte port identifier, allowing for a host to have several addresses. The TCP header structure can be seen in Figure 2.5c.

It is difficult to present overhead figures in this case. TCP provides reliable packet delivery, making use of retransmissions, acknowledgements and other complementary actions. Basic figures can be presented by taking into account only the extra bytes needed by TCP, not considering any extra packets needed by the protocol, likely to cause real figures to be much worse than the ones below. Under those safeguards, the best case for the overhead for three data packet sizes (in bytes), supposing the minimum number of link level packets is used and no special options at both IP and TCP levels, is:

size (bytes)	512	1024	4096
overhead	13.2	7.1	3.6

The *overhead* above is in percentage, combining the data link, the IP, and the TCP layers.

2.3.5.5 Other Higher Level Protocols

Most higher level protocols are built on top of TCP, due to its characteristics of reliable ordered delivery of data in a transparent manner.

TELNET, short for *TEL*etype *NET*work, part of the DARPA set of protocols, allows a user at one site to establish a TCP connection to a login server at another, passing keystrokes from the local machine directly to the remote machine [Com87].

The 4.3BSD Unix *rlogin* is a remote login service that allows system administrators to choose a set of machines over which login names and file access protections are globally shared, and to establish equivalences among user logins. Users, having different login names on different hosts, can then set things up so as to be able to do remote logins without having to type a password each time. It makes extensive use of TCP connections.

FTP is a *f*ile *t*ransport *p*rotocol from the DARPA set. Authorized users are allowed to log into a remote system through identification and to transfer files between the systems. Basic commands can be remotely executed to help the user. Statistics and transfer rates about the transactions are also provided. It is built on top of TCP.

SMTP (*S*implified *M*ail *T*ransport *P*rotocol) is a mail transport protocol that relies on TCP and specifies how connections are made and how mail is transferred from one machine to another.

Other simpler protocols such as *TFTP* (*T*rivial *F*ile *T*ransport *P*rotocol) use UDP instead of TCP. Figure 2.6 gives the resultant packet lay-out when dealing with a combination of Ethernet-IP-UDP, a framework upon which the experiment described next is based.

2.3.6 Description of the Experiment

2.3.6.1 Environment

The experiments related below were targeted at identifying the overhead or bottleneck that causes such a small fraction of the nominal bandwidth to be generally available, even with considerably high-powered hardware.

Given the ready availability of a BSD Unix environment (4.2 and 4.3 implementations), with several SUN fileservers and discless stations together with several minicomputers such as VAXes (8600, 750) and other similar machines, it

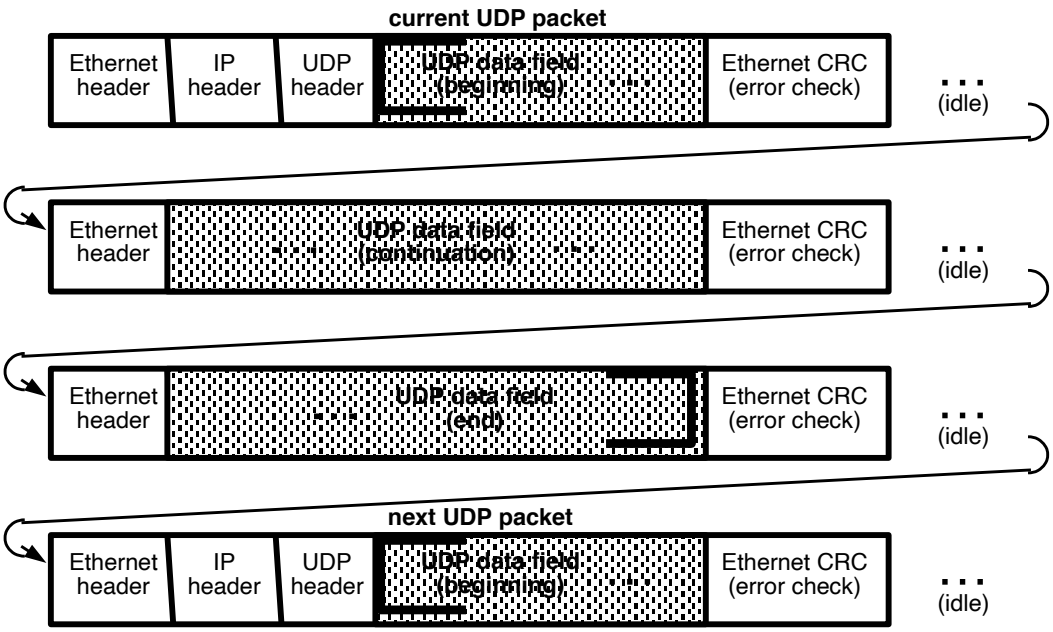


Figure 2.6: General lay-out of packets at UDP level (large packet break-up).

was decided to base the software on sockets (Berkeley extension). The network to be used consists of a backbone connecting all single machines (minicomputers) and all SUN fileservers. From each SUN fileserver a private network (or *spine*) connects a group of discless stations in a standard fashion (all Ethernet as well).

2.3.6.2 Details of Programs

Due to concerns about likely overheads when using high level protocols such as TCP, it was decided to have the programs use sockets based on the UDP protocol. The UDP protocol is very simple and causes only extra bytes to be added to the basic overhead of the IP level. The statistics in Section 2.3.5.3 show that the total overhead will not be significant, allowing above 90% net channel utilisation. The reason for opting for UDP instead of straight IP was that by using IP the programs would have to run with privileged status, potentially causing service disruption throughout the network, upon which many people's work depends.

To run these experiments two programs were needed running on different hosts, so that Ethernet use could take place. Though the model of communication requires no asymmetry, one of the programs does take the function of a

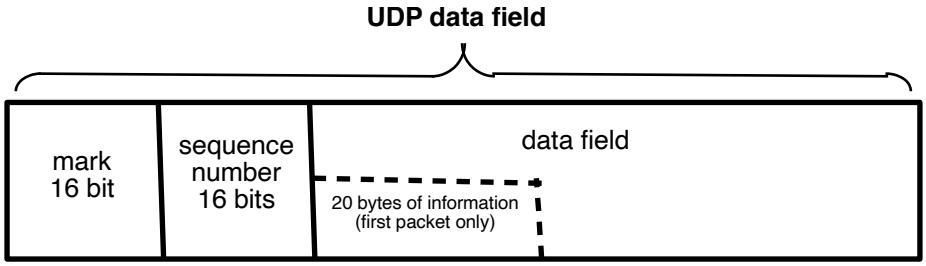


Figure 2.7: Packet format for the dedicated protocol used.

server, waiting for packets arriving at a specified port to conveniently receive them (preferably avoiding buffer overwriting). Sequence numbers of successfully received packets are recorded in a unique file for later analysis. The number of successfully received packets is printed on screen for a quick check.

The companion program takes the role of a *client* that requires service from the *server*. The service in this case is to consume packets sent to the server. The *client* establishes a connection with an argument-specified host (where the *server* is running) through a socket and sends some information in an isolated packet to set up the experiment, such as which host is requiring service and the number of packets to be transmitted. Some time is allowed for the server to do some housekeeping (reading the information passed and opening the file).

After the initial setup, time is recorded and fixed-size packets are transmitted through the socket according to the number specified as argument in the invocation. Each packet carries in the beginning a sequence number to allow identification by the *server*. On finish, time is read again and the resultant interval is printed on-screen by the server.

Figure 2.7 shows the dedicated protocol packet format used, which comes encapsulated in the UDP data field.

It should be noted that the dedicated protocol used here makes no attempt to ensure reliable delivery of data. As will be seen in Section 2.3.8, the overhead caused by maintaining acknowledgement windows and ordering the stream of data can be kept small, due to the simplifications a dedicated protocol can afford.

2.3.7 Results of the Experiments

To present the results a rough comparison of the machines involved must also be shown, as well as their disposition on the network. Four kinds of SUN machines (3/50, 3/60, 3/180, 3/280) were used, and also two VAXen (750 and 8600). Table 2.1 gives a convenient but not precise processing power index, based on manufacturer's figures and broadly accepted as is. The discussion of the merits of the table is out of the scope of this research, but see SUN and DEC product literature for further information.

host machine	speed (VAX 11/780 MIPS)
750	0.7
3/50	1.5
3/180	2.0
3/60	3.0
3/280	4.0
8600	4.0

Table 2.1: Comparative speeds of machines used.

As stated previously in section 2.3.6.1, the SUN fileservers (3/180, 3/280) and the VAXen (750, 8600) are in a single cable, the departmental *backbone*, whereas experiments with SUN discless stations were performed in their *spine* only, to avoid difficult interpretation of the results caused by transmission of packets across an internetwork gateway (i.e. a SUN fileserver).

See Table 2.2 for the results. Throughput figures are already net, that is, take into account only the amount of useful data transmitted (mark and sequence number are not included). They are calculated with the number of packets sent during the time interval recorded by the client program. The percentage of accepted packets is given by the server program, which logs all accepted packets and is so able to present such data. One piece of information that this table does not present is that although packets may be dropped according to the load on each machine and network usage, all packets accepted during all instances of the experiment were ordered. Notice that although the throughput column considers only the 1024 bytes of net data inside the dedicated packet, the channel utilisation column also considers only those 1024 bytes, instead of the actual 1028 bytes being carried by the UDP packets. The real figures for the last column should

source *client*	destination *server*	network branch	accepted packets (%)	throughput (KBytes/s)	channel utilisation at UDP level (%)
SUN 3/50	SUN 3/50	SUN spine	88.6	343	29.1
SUN 3/280	SUN 3/180	backbone	49.8	402	34.1
SUN 3/280	SUN 3/50	SUN spine	22.2	498	42.2
SUN 3/50	SUN 3/60	SUN spine	100.0	283	24.0
SUN 3/280	SUN 3/60	SUN spine	85.6	443	37.5
SUN 3/60	SUN 3/280	SUN spine	99.8	486	41.2
SUN 3/60	SUN 3/50	SUN spine	2.7	489	41.4
VAX 750	VAX 8600	backbone	96.0	63	5.3
VAX 8600	VAX 750	backbone	11.7	655	55.5
VAX 8600	VAX 750	backbone	–	(top) 833	70.6
SUN 3/280	VAX 8600	backbone	97.5	386	32.7
VAX 8600	SUN 3/280	backbone	21.1	806	68.3
VAX 8600	SUN 3/280	backbone	–	(top) 917	77.7

Table 2.2: Results of experiments.

then be slightly higher.

The overhead of 6.1% mentioned in Section 2.3.5.3 refers to a total of 106.1% of transmitted bits. Scaling 106.1% down to 100% the overhead value becomes 5.7%. The maximum available bandwidth at the UDP level can now be calculated as

$$\textit{available bandwidth} = (1 - 0.057) \cdot 10 \text{ Mbits/s} \rightarrow 1.18 \text{ MBytes/s}$$

The experiments were performed during calm hours, to avoid much of the traffic everyday use causes. The presence of any extraneous traffic on the networks was easily seen with varying performances, and when such situations occurred the experiments were redone to give smoother results (little variance). All values are averages of several runs of the same experiments, usually ten of them, except for the entry marked with *top*, which is the best result recorded.

2.3.8 Analysis of the Results

One sees for the slowest SUN machines an average throughput of 343 KBytes/s, which is not quite impressive for the data rates expected to be found. A significant improvement comes with faster machines like the 3/60 or 3/280, where speeds close to 500 KBytes/s were recorded. I was still short of reaching the desired throughput figures when it was decided to extend the experiments to the

larger VAX. These results were greatly rewarding, with an average throughput of 806 KBytes/s when paired to the fastest SUN server (though throughput is not dependent on receiver speed) and 833 KBytes/s to a VAX 750. Top throughput value recorded was 917 KBytes/s, which was the sort of value sought. To complement the spectrum, figures for the VAX 750 were quite low, in the range of 60-80 KBytes/s at most.

Though the values are not proportional to the processing power of the sending hosts, they do show a clear relation to machine speed. What was surprising was the difference between the 8600 and the 3/280, which are of comparable power. A SUN fileserver has considerable work to do in the background to maintain its discless workstations' environment, often using the network to do so. It is not easy to quantify such load, but it is assumed that there is an extra overhead when the SUN fileserver is compared with the VAX (the VAX has only ASCII terminals to service, the SUN supplies file systems and environments to the stations through the same network).

The difference was nonetheless quite significant, and I came to believe that it was due to a better I/O architecture of the big system, traditionally put in the supermini class. Later it was confirmed that the UDP code on the 8600 had been patched and was not performing a checksum over the UDP data field. I now believe that the discrepancy was mostly due to the extra processing load imposed by calculating the checksum over all bytes to be sent through the Ethernet, in the case of the SUN system. However, a less efficient I/O architecture cannot be ruled out without a deep analysis of the hardware involved. This discussion suggests that the following statement might be true: all SUN machines would show proportionally better figures if i) their code was also patched, or if ii) the experiments were run using raw sockets (IP protocol).

Close examination of the 'packets accepted' column also reveal that the ability to receive packets safely is also related to the processing power available (again, the 8600 shows its extra advantage by not performing checksum calculations). If both ends of a communication link have similar power the percentage of accepted packets will be in the range of 80-90%, meaning that only 10-20% of the packets will need retransmission. The overhead incurred by these retransmissions can be kept at a not much greater figure using a simple acknowledgement protocol coupled with the one used in the experiments.

Throughput and transmitter machine speed look well related, as can be seen

by correlating columns 'source' and 'throughput', bearing in mind Table 2.1. There seems to be two variables in the process: i) the ability of the host to supply data ready for transmission at a fast rate, and ii) the ability of the Ethernet interface to process the data to be transmitted into network packets. With the machines used in the experiments it seems that the network limit was not reached at all.

The results close to 1 MBytes/s (close to 80% of the maximum bandwidth at UDP level) guarantee almost the full bandwidth. This is important to confirm the results of [Gon83, HM87, NB82] and allay earlier fears about achievable bandwidth.

2.4 Recommendations for the Project

It has been seen in the previous section that even a slow SUN workstation could show quite good figures for Ethernet throughput, at least if simple protocols are used. By simple is meant not very far from a UDP protocol, without the extra toll imposed by performing a checksum at this level.

The intended application (live image display) is not as sensitive to errors as a file transfer and its associated reliability, in the sense of possible harm caused. If the CRC code at the Ethernet data link level allows a faulty packet to be accepted (a very rare situation), the image can stay disrupted for some time, but it is likely to be refreshed by updating data, either by a moving scene or by the scheme used to compress the image. This is true in the case of a compression or encoding method that does not transmit 100% of the picture information, where almost certainly a slow refresh of all points of the screen will have to be regularly performed as a background task, if image fidelity is to be maintained. In any case, a disruption in the display of an image sequence is not considered to be fatal for this project and could be tolerated in rare situations.

One can imagine an architecture that relies on a separate processor to receive compressed images from an Ethernet connection, possibly using a dedicated cable to isolate the imaging processing load from the normal network and file system operations on which a discless workstation is based, so that normal processing performance is not affected by image sequence transmissions.

Processing units for these communications tasks must be fast in moving information. RISC architectures provide this feature with much less silicon area than

CISC ones and are therefore primary candidates to look at.

One example implementation would use, say, transputers, to do all functions in the add-on hardware: transmission/reception via the network, encoding/decoding and compression/decompression. The fast communication features of these devices help a great deal in solving some of the internal bandwidth problems to be encountered in devising a suitable architecture for this project.

2.5 Conclusions

This chapter established a detailed view of the desired video viewing system, presenting also important data rate figures. The standard image source adopted generates data at a rate of ≈ 6.6 Mpixels/sec, which is translated to 6.6 MBytes/sec.

Ethernet networks were not very well reputed in the academic environment for their overall throughput. The experiments reported in this chapter proved that given appropriate hardware one can obtain useful throughput figures close to the theoretical limit. A VAX 8600 managed to pump data into the network at 917 KBytes/sec, using a protocol that provided enough facilities for the intended video transmission. At this protocol level the theoretical upper bound of the Ethernet is ≈ 1.2 MBytes/sec.

A simple division of the figures cited in the two previous paragraphs indicate that compression rates of at least $5.5 : 1$ are required to transmit video through an Ethernet, as specified in this chapter. To allow room for normal network operations a good suggestion is twice that figure ($11\text{-}12 : 1$), and to allow more than one simultaneous transmissions the compression rate suggested is at least $22 : 1$.

The experiments also revealed an apparent weakness of SUN systems as regards to I/O architecture. A machine of processing power equivalent to the VAX cited above did not stand up in the ability to pump data down the Ethernet interface.

The next chapter presents a survey of image compression and coding techniques, so that appropriate techniques could be examined for possible use in the intended system.

Chapter 3

Survey of Image Coding and Compression Techniques

3.1 Introduction

This chapter aims to present a broad view of image compression and encoding techniques, going from the basic ones up to hybrid methods and analysing advantages/disadvantages as seen in the light of this project's goals.

Although such techniques have well defined scopes of action, subdividing them into separate categories is not easy, usually because their scopes often overlap with one another, giving rise to several different categorisations.

Pratt [Pra78] presents encoding and compression schemes in such a way that suggests the following subdivision:

> Frame and line techniques.
> Basic techniques.
> Predictive coding.
> Spatial coding.

whereas Netravali and Limb [NL80] prefer the following categories, which are no doubt more descriptive than the above ones:

> PCM (Pulse Coding Modulation).
> Predictive coding.
> Transform coding.
> Interpolative and extrapolative coding.

Statistical coding.
Other methods.

Jain [Jai81] makes the division in four levels, as seen below:

Zero memory methods.
Predictive coding.
Transform coding.
Hybrid coding.

It was felt that a more suitable classification was needed to properly accommodate several methods not cited by one or another of the above categorisations. None of them, for instance, mentioned newer techniques such as BTC and VQ. Therefore it was decided to follow the structure depicted in Table 3.1 to present a collection of available coding and compression techniques.

To justify such classification a few of the principles that governed its conception are given below :

- Although line and dot interlace are closely related to subsampling and interpolation, the former are considered as a kind of hard technique, i.e. assumed as design features that may not be altered by the coder design. For example, the line interlace present in standard video is taken for granted throughout this thesis. However, after the video signal is acquired, digitised and stored in the frame buffer, further subsampling and interpolation may be employed; that is why subsampling and interpolation appear as "Basic techniques".

- Under "Predictive coding" are listed those methods that are based on some form of estimate about what the next element to be coded looks like. For instance, DPCM and Deltamodulation consider an underlying trend that the next element to be coded is likely to have approximately the same value as the previously coded one.

- Transform coding should actually be classified as an image block coding technique. Not only is it important enough to be in a class of its own, but it also differs in a practical way when compared with the other image block coding techniques, as explained in the item below.

Frame and line techniques.

- Line and dot interlace.
- Frame rate reduction (frame freezing).

Basic techniques.

- PCM.
- Statistical coding.
- Run-Length coding.
- Bit-plane coding.
- Quadtree encoding.
- Subsampling and interpolation.
- Feature coding.
  - Luminance contour coding.
  - Edge coding.
  - Texture coding.

Predictive coding.

- Deltamodulation.
- DPCM.
- Interframe conditional replenishment (DPCM).
- Motion compensation.
- Quantization and Predictive Techniques.

Transform coding.

- Standard transforms.
- Adaptive transform coding.
- Variations of basic transform methods.

Image block coding.

- BTC (block truncation coding).
- VQ (vector quantization).

Table 3.1: Structuring of image compression and coding techniques.

- By "Image block coding" are meant those techniques that are based on fixed sub-blocks of the image. Their approach to code a given block may render the reconstruction block quite different in its basic grey-level shape. BTC happens to do so due to the lack of a better shape description tool, given that only two pixel values are possible in the reconstructed block, whereas VQ might present a reconstructed block with very little shape resemblance, since the coder has to select a shape description among the finite number of shape descriptions available. Transform coding on the one hand and BTC and VQ on the other hand differ in their approach in a practical way, when comparing the number of different reconstructions possible with each method: in the latter case it is quite small, whereas in the former it can be much higher.

- the remaining techniques are grouped into "Basic technique", meaning that either they are indeed considered basic in the image compression field or they do not fit any other class and do not present an importance of its own in the scope of this project.

3.2 Frame and Line Techniques

Since its beginning commercial TV has relied on line interlace to cut down the bandwidth required to transmit the signal necessary for good motion rendition on screen. Besides line interlace, other techniques used for very low transmission bandwidths are dot interlace (subsampling along a scan line) and frame rate reduction.

3.2.1 Line and Dot Interlace

3.2.1.1 Line Interlace

Figure 3.1 depicts the 2-to-1 line interlace mechanism. The use of line interlace in commercial TV allowed halving the bandwidth of noninterlaced transmission but without incurring two side effects that would normally show up.

flickering: Normal persistence phosphor used in commercial TV and in most computer screen equipment tends to flicker at refresh rates below 50-60 Hz,

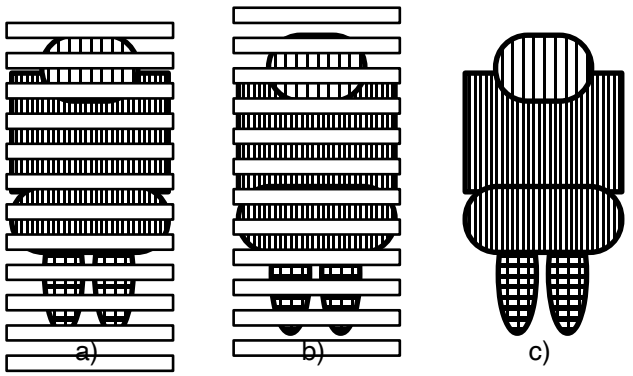


Figure 3.1: Interlaced video: two fields compose a frame.

for most observers [Jai81]. Halving the frame rate to 25-30 Hz and keeping it noninterlaced would cause objectionable flickering, but by using line interlacing (two fields making up for a full frame) the effect is greatly diminished. An explanation for this fact is that although each field (every alternate line on screen) is being refreshed at a lower rate, the visual effect of the last refreshed field is much stronger than the previous one. Since the two images overlap each other the brighter one (most recently redrawn) will be dominant, helped by the spatial integration performed by the human visual system.

motion breakup: To determine a minimum frame frequency to capture motion depends on the nature and speed of the motion. With noninterlaced video 25-30 frames/s is considered sufficient for typical scenes [Bal58, Jai81], but is noticeably worse than 50-60 frames/s, observers noticing an image breakup and jump effect. However, by using interlacing this effect is also lowered for the same reason pointed out in the previous item, since fields are now being acquired and shown at double the frame rate (though only half the number of lines), with the steps in motion (jump) being reduced to half the previous length. The effective motion rendition is closer to 50-60 frames/s than to 25-30 frames/s.

The above reasoning reveals also another problem that this project will face,

a problem that will not have an easy solution. In a complete system, as being devised, an image is acquired and coded at one station, sent through a communication channel to another station and there it is decoded and shown in a window under control of the local window manager, in an as close as possible real-time fashion.

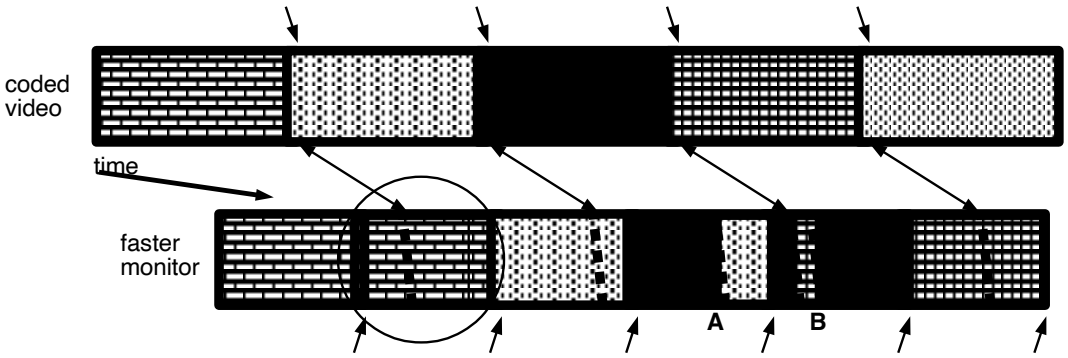
The problem arises due to frame rate differences between the standard TV that is being used as data source and the output medium, which is the station's monitor. Commercial TV uses either 25 or 30 frames/s interlaced (50 or 60 fields/s), whereas workstation monitors usually display in the range 55-78 frames/s noninterlaced.

The first thing to note is a side-effect of going from interlaced to noninterlaced display. If the chosen encoding method works on a frame basis (as opposed to on a field basis), then the resulting reconstructed image sequence will present a more blurred appearance for moving regions. One would expect at first that a more accentuated jump effect and motion breakup would result, given that motion rendition is now performed at the source frame rate (as opposed to the field rate), but in fact what shows up is a blurring effect. Figure 3.2.1.1 shows this effect visually. The blurring is the result of mixing two consecutive shots (two fields) of a sequence in a single output frame, because no longer there is time for the first field registration to fade enough so that only the current field dominates the visual appearance.

The main problem however relates to the different frame rates, on the unlikely event of the station's monitor vertical refresh rate being an integer multiple of the input frame rate (either 25 or 30 Hz).

Most modern video architectures allow access of the frame buffer by the main CPU, even when the screen is being refreshed, thanks to the availability of VRAMs (Video RAMs). VRAMs makes this much more easily attainable for the hardware designer.

Supposing that the coded input video data arrives ordered and well timed at the receiving station, Figure 3.2 shows what happens when those frequencies are not integer multiples. The explanation given in the figure shows that a frame cannot be displayed before it arrives in its entirety. Even if such delay is honoured, other effects will happen. If a frame is received and updating must start, and



The small arrows signal frame starts for both video streams. Notice that the faster output monitor will display, in a single output frame, parts of more than one frame from the input video source.

The diagonal double arrows represent an arbitrary skew, a delay caused by processing and transmission of the image information.

The different shade patterns represent which frame information is being displayed at a given moment.

The dashed lines represent important moments in the process, either the beginning of a new frame from the received video signal or the moment when discontinuity happens, i.e. the displayed image changes to another frame from the received data. Notice that the image sequence can actually go backwards, as in moments **A** and **B**.

If a horizontally moving object happens to be where the frame boundary is in the reproduction monitor the human visual system picks up the disjoint images and registers the effect, even though this happens for a short time only. The effect is not strong but research is needed to study what damage to reproduction quality is felt by observers.

In this situation faster does not mean better. In order to avoid this loss of phase, the only solution is to delay the display of the received signal at least a for a full frame period of the received signal. In other words, display of a new frame cannot be started before the complete frame arrives. In practice, this helps the designer of a video viewer, since image reconstruction takes some time anyway. The image reconstruction delay is offset by the need to delay the displaying process.

Figure 3.2: Timings for different frame rate displays.

the local station's refresh cycle is in progress, say half way through the window where the image is to be displayed, discontinuity of frames will happen. If that image region happens to have a horizontally moving object, the motion breakup will be noticed. A vertically moving object will not cause much side-effects. One can think that since the distortion only lasts for the duration of that frame, the sequence as a whole would not suffer too much and the observer would hardly notice it. Experiments performed by the author (see Section 4.5) with image sequences containing distortions revealed that bad effects are quickly noticed and remembered for a long time (as regards to quality evaluation). Further research is needed to qualify and quantify the badness for image sequences with non-synchronizing frame rates.

3.2.1.2 Higher Order Line Interlace

Higher order line interlace is also possible, with 3-to-1, 4-to-1 or even 8-to-1. Apart from motion breakup the main problem faced by line interlacing relates to flickering, since parts of the screen will only be refreshed after a longer period than usual. This can be overcome if the display has storage for the full frame and performs a normal rate refresh on screen while updating the contents at the slower rate.

If storage is not available, however, the flickering effect shows up in the form of line crawling disturbances. With 3-to-1 sequential line interlace only one every three lines is refreshed in the same field while other lines refreshed previously are fading out and are much less visible, so that the visual effect is of lines shifting slightly but continuously down (if that is the order of the refresh method). By adopting a pseudorandom scanning sequence one can minimise this effect. Notice that with 2-to-1 interlace the lines are symmetrically spaced and do not cause the crawling effect.

3.2.1.3 Dot Interlace

Interlacing can not only be employed in the vertical direction but also be extended to the horizontal direction, known as dot interlace. Applying dot and line interlace simultaneously is generally avoided due to line crawling disturbances [Pra78]. Many possible dot patterns can be devised with high order interlace [Dub85], but the more bandwidth is saved the more pronounced become the side effects.

These are patterning, a false texture appearing in the picture, and dot crawling, an optical illusion similar to line crawling.

Both effects can again be minimised by careful choice of of the dot pattern, but again the usefulness of such techniques is limited by the combined degree of phosphor and retina persistence. Provided overall persistence is enhanced (such as by having storage in the display), dot interlace can provide some useful bandwidth reduction, but motion activity must not be high and some tolerance to blurring must be accepted.

3.2.2 Frame Rate Reduction (Frame Freezing)

As seen in the previous paragraphs, frame rate reduction is limited by the combined persistence of the phosphor and the retina. By employing storage at the display one can overcome the flickering problem and some bandwidth saving may be accomplished.

One will now run into motion breakup effects since frames are being dropped from the original sequence, although without causing flickering. There is little report of specific research to determine what frame/field rate is acceptable under what circumstances. Baldwin [Bal58] performed experiments using 24 frames/s motion picture and reported 12 frames/s to be satisfactory for head-and-shoulder sequences, but less satisfactory for scenes with more motion contents.

Chen [CP84] presents a coder that uses frame freezing at 2:1. In private correspondence he says 3:1 frame freezing is the most one can do to avoid losing lip sync in head-and-shoulder sequences, but has not performed quantitative experiments to determine what frame freezing rates are acceptable or not.

In order to get a better idea of how one could count on frame reduction (frame freezing) for bandwidth reduction some experiments were really needed. It was decided to perform a set of experiments with audio visual equipment, described in Section 4.5.

Results showed that the first step one can do in frame freezing, that is, freeze every other frame (12.5 frames/s) is already too much at PAL standards (25 frames/s), yielding clearly noticeable motion breakup. 55% of subjects (6 out of 11) reported motion breakup as annoying in most sequences of the experiment described in Section 4.5. Evidence from numerical results and significant subject opinion suggests that the use of frame freezing should be restricted, if used at all. Under this view even the slightest level of frame freezing should only be considered

at rarer occasions, such as scene change and heavy load on the communications link.

Frame freezing causes motion breakup. To lessen this techniques to obtain better motion rendition have been devised, e.g. [Dub85, NR81, Ber81]. Instead of just repeating the same frame (in the case of a 2:1 freezing pattern, for instance) at the receiving station, the decoding process is delayed enough frames so that the missing frame can be reconstructed from the previous and next ones, employing basically linear interpolation, but more complex motion-compensated techniques yield better results.

3.3 Basic Techniques

This section presents an overview of image coding and compression methods which do not fit in any of the other main categories. Among them are from very early and basic techniques such as Pulse Code Modulation (PCM) through information preserving and hierarchical structuring ones, up to feature-based techniques.

3.3.1 PCM (Pulse Code Modulation)

3.3.1.1 Image Quantization

Digital image compression and coding almost always rely on the existence of a quantizer. An analog signal representing an image has to be sampled (continuous time becoming discrete) and quantized into a finite set of possible values digitally represented (continuous range of amplitude becoming a finite set of values). This process is called Pulse Code Modulation (PCM).

Quantization involves firstly a mapping of the continuous function conveying the brightness along a scan line into a finite set of values called *transition* or *decision levels*, each representing a continuous range of the input signal. This set of ranges should of course be contiguous and cover the whole dynamic range of the input signal. Secondly, *reconstruction levels* are assigned to each range and given to each sample that falls inside them.

The quantization function is more easily seen as a staircase function: if a sample of the input signal lies between two decision levels, it is quantized to a reconstruction level lying inside the range defined by these decision levels. Refer

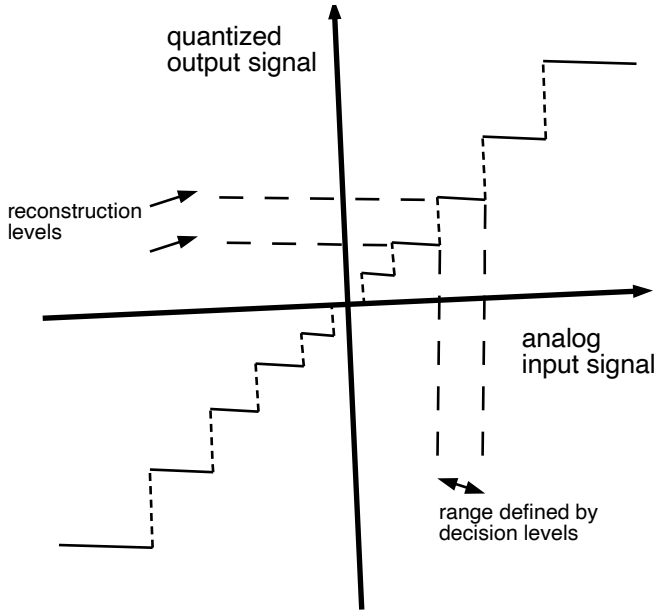


Figure 3.3: Quantization: mapping of analog to digital signal.

to Figure 3.3.

3.3.1.2 Quantization, Contouring and Dithering

The quantization process obviously introduces distortion in the digitally reproduced output, the amount of distortion depending on how accurate the quantization is in terms of amplitude resolution as well as spatial resolution (sampling), considered bidirectionally on the image plane.

For the PAL television standard usually a spatial resolution of approximately 640x512 is taken for granted. In this project I shall always be referring to a resolution of 512x512, unless otherwise noted. A square centred in the middle of the picture is being considered, with the vertical margins being discarded for practical purposes. Also, an amplitude resolution of 6-8 bits is considered necessary to allow good grey-scale rendition whilst keeping false contouring imperceptible. Five bits (32 grey levels) will not present objectionable contouring [Jai81, Pra78]. The acceptable contouring limit depends on other factors such as subjectiveness, the dynamic range of the output monitor and the contrast setting a given user prefers.

The allocation of decision and reconstruction levels can affect the resultant image quality and ultimately the number of bits needed at the quantizer to satisfy

a given quality goal (i.e. how much contouring will be allowed).

Consider the pixel brightness to be modelled by a random variable with a continuous probability density function. The optimum quantizer known as Lloyd-Max quantizer [Max60] minimises the mean square quantization error for a given number of quantization levels. The result is transition levels that lie half way between the reconstruction levels and reconstruction levels that lie at the centre of mass of the "density" in the transition intervals.

For the case of a uniform probability density function that means that both the transition and reconstruction levels are equally spaced, and the quantizer is also called a *linear quantizer*. This is not true for probability density functions following other models, such as Gaussian and Laplacian, but in general decision levels will be more tightly spaced where the probability density is more concentrated. Nonlinear quantization can also be performed by using nonlinear transformations on the sample, quantizing linearly and performing the inverse nonlinear transformation.

Contouring effects can be suppressed by adding a small amount of pseudorandom uniformly distributed noise to the brightness samples before quantization and subtracting it at the receiver, a technique called *dithering* [Rob62]. This method works because the added noise causes some pixel values to go above the original decision level and others below it, 'hesitating' about the decision level, but the average value of the quantized pixels remains about the same as the original ones.

The amount of dither to be chosen is a trade-off between being small enough to keep a reasonable spatial resolution and high enough to allow brightness to vary randomly about the decision levels. A practical measure is to let noise affect the least significant bit of the quantizer [Jai81], but observation of a number of acquired images and comparison with their filtered versions[1] showed that noise affecting the last two bits is not worse than 1-bit noise, but actually causes better image rendition, in the sense of their looking more natural. Working with 64 grey levels and that much noise rarely gives chance for contouring to appear, but under closer examination it becomes clear that this happens thanks to the intrinsic noise of the Vidicon camera in the setup (making the pixels vary within

[1] the standard gaussian convolution is adequate for this purpose, since the regions of interest are where brightness varies slowly. A 3x3 Gaussian filter is enough to remove most of the noise and leave exposed the boundary between adjacent regions. Perception, of course, depends on a number of other factors.

approximately 3 out of 64 brightness levels). Contours appear very disrupted by such high noise.

What is called 'dithering' here is a special case of a more general technique called *digital halftoning*, also known as *spatial dithering*. Digital halftoning is the method of rendering the illusion of continuous-tone (grey-scale) pictures on devices capable of producing only binary picture elements [Uli87]. This is achieved through judicious arrangement of the picture elements on an imaginary grid laid onto the display surface. Both white and blue noise based halftoning relies on a random approach, in much the same way as described in the previous paragraph. The importance of digital halftoning lies in the fact that most printed material nowadays use digital halftoning for grey-scale rendering.

PCM in itself is of little use for the purpose of data compression in the scope of this research, since part of the image acquisition process is considered fixed as a design feature. The brightness sampling will most probably be done in a standard way, which is usually a linear scale on the analog voltage provided by the camera, perhaps with some tone adjustment (but this makes the setting specific to the type of camera used). The output amplitude resolution is assumed to be 256 grey levels, whether or not a particular image acquisition device plus sampler being used is able to provide that range. This assumption about image acquisition involves most image coding applications and standardises the way the image is treated.

In summary, PCM will use 8 bits/pixel to fully represent images (a compression rate of 1:1), with 5 bits/pixel being the least acceptable (1.6:1).

3.3.2 Statistical Coding

The basis for statistical encoding lies in the well known fact that real world images contain a great deal of redundancy, which is easily shown by statistical measurements. In this case redundancy refers to the total number of bits per pixel required for PCM coding of the image minus its entropy. Here entropy is expressed in bits/pixel, and information theory guarantees that entropy is the minimum amount of data necessary to code all information in that image. This redundancy can easily be confirmed by having a quick look at a real world picture and noticing that neighbouring pixels are very likely to have the same brightness. The most obvious way to exploit this redundancy is to code the difference between pixels, using a technique called DPCM (see Section 3.4.2), thus allowing less data

to carry the same quantity of information.

There are statistical techniques that allow coding a real world image with a rate in bits/pixel close to the theoretical minimum (entropy) for that particular image (the significant point here is the rate being anything *less than that of straight PCM coding*), with all its original details. This is called *lossless information coding*, or *information preserving coding*, which accounts for the fact of statistical coding being used in general data compression where exact data reconstruction is required.

These techniques basically work by subdividing the string of data in small packets called *source messages* and finding out the probability of occurrence of each message. The ones with highest probabilities are assigned *codewords* of smaller sizes. In image applications suitable messages to be encoded are the pixel differences along a scan line (DPCM), instead of the brightness of individual pixels. This already reduces a great deal of redundancy normally present in an image. Statistical coding can also be used to code with some information loss, for example, by assigning codewords to a range of pixel differences [Pra78].

Among the most common techniques are Shannon-Fano [SW49], Huffman [Huf52], and Lempel-Ziv [ZL77]. The latter two will achieve performance close to the limit given by the entropy of the image, more often than the former, and appear in popular data compression utilities like *compress* for Unix, *PKARC* for the IBM PC and *StuffIt!* for the Apple Macintosh [LH87].

For the purposes of this thesis statistical coding is not generally useful since it is good only for lossless coding, which yields nothing close to the compression rates needed. Exact reproduction is not required and the trade-off for reproduction quality allows much greater compression rates. It could possibly be used, however, to compress the data stream resulting from some other technique, although generally at this stage the input data (the output from the previous technique) has already enough entropy to render statistical coding useless[2].

Typical compression rates to be expected are in the range of 4 to 6 bits/pixel (1.33:1 to 2:1), obtained with *compress* on some images appearing in this thesis. Adaptive statistical techniques may achieve up to 2.5 bits/pixel (3.2:1).

[2]An interesting technique is used by the Unix *compress* utility. It tests the output stream length to know whether it resulted shorter or longer than the original. If longer, it discards and replaces the generated stream by the original stream [UNI85].

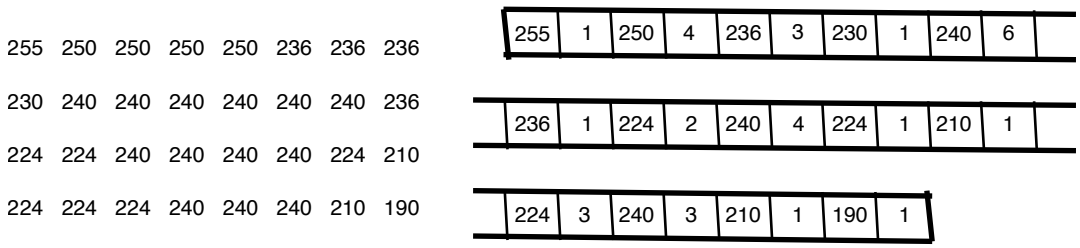


Figure 3.4: Run-length coding scheme.

3.3.3 Run-Length Coding

Run coding is a simple coding technique that explores the redundancy along a scan line by finding sequences of contiguous pixels with the same brightness. It can also be employed as a non-information preserving technique by ignoring small changes in the brightness, but it is most often employed as a lossless coding technique. It can be found in just about every host platform (at least as an intermediate image format, as is plain PCM coding) and is employed for many image formats due to the ease with which some compression is achieved at very low computational cost.

For each contiguous sequence (called *run*) either the brightness or the amplitude of the difference in brightness from the previous run is registered, together with information about where the run ends. If the location information is specified relative to the previous end then the method is called run-length coding. Figure 3.4 illustrates a typical run-length coding system.

To further compress the resultant string of data a maximum run length is imposed, so that fewer bits will be needed most of the time. When a run does exceed it, the solution is to insert more than one run to represent it. The overload with extra runs is more than compensated by the saving provided by fewer bits used in small runs, but this will in the end depend on the image statistics, which are in turn dictated by the quantization chosen and the noise level of the acquisition system. Pratt [Pra78] presents an analysis of the average number of runs versus probability of edge occurrence (i.e. brightness change), for limited length runs.

Notice that the technique can also be extended to the two-dimensional case.

This thesis is interested in images quantized with 256 levels (8 bits/pixel), which decreases the technique's compression abilities as compared with shallower images. Image noise will then be the determinant factor in deciding upon its utility. Also, the coding structure does not make itself attractive for exploiting one of the most important sources of redundancy in image sequences: the frame-to-frame redundancy. The view at this stage of the research is that it could be used as a support technique to work with some other one.

The example of Figure 3.4 is a bit optimistic. More typical values for data rates range from 4 bits/pixel to over 8 bits/pixel (obtained from some images appearing in this thesis), depending mostly on camera noise.

3.3.4 Bit-Plane Coding

Grey scale images are usually represented as a matrix of integer values giving the brightness associated with each pixel. Another way of representing the image is to consider the number of bits required for the pixel brightness (quantization) and subdivide the matrix into a heap of planes 1-bit thick, so that in effect one has several monochrome images to code, where methods like run-length coding are well suited.

The first thing to notice is that planes corresponding to the most significant bits of the brightness values have little activity and are very efficiently coded by run-length coding or similar techniques. Unfortunately the least significant planes are almost entirely constituted of noise, which overloads most coding techniques, particularly run-length coding, rendering the use of bit-plane coding not much useful. Compression figures in the range of only 2:1 have been reported for grey-level images [SB66, SH69].

Decomposition in bit-planes can be better run-length encoded if one gets rid of the noisy least significant planes. One trick that achieves this without loss of information is the use of Gray codes, instead of normal binary codes, to represent the brightness amplitude. Gray coding is known for its feature of always having a single bit transition in the code word when the value represented is changed by one unit, wherever it is in the range of allowable values.

Kawaguchi [KEM83] worked with 4-bit deep images and reported a complexity value for a Gray coded image half of that for a binary coded version of the same

image. Even better results were given by processing the image with the Depth-First encoder, but this is a non information preserving technique.

The range of overall compression rates achieved with bit-plane encoding is not good enough to be considered for this project. Figures in the range of only 2:1 (4 bits/pixel) or lower compression rates have been reported [SB66, SH69].

3.3.5 Quadtree Encoding

Quadtrees are hierarchical data structures based on recursive decomposition of space. There are many variations on the basic theme, but in order to narrow the field and summarise matters for the purpose of this research some reasonable assumptions will be made. The decomposition process will be limited to a regular decomposition, that is, into equal parts at each level (regular polygons). The quadtrees will represent region data, thus being more properly called *region quadtrees*. This decomposition will be based on the successive subdivision of an image array into four-equal sized quadrants (blocks), and the process may or may not be recursively applied, according to the pixel values in the array [Sam84].

Figure 3.5 shows the basic mechanism of a quadtree. Part *a* shows a region to be coded, part *b* its corresponding bitmap, part *c* shows the block decomposition and part *d* a tree of degree 4 (each node can have 4 children). The decomposition is as follows: if the array or first block has pixels of different values (0 or 1) then it is subdivided, and its children are subdivided further until all subblocks have a single value inside them. Elements in the tree are its root, representing the whole array, *leaf nodes*, which have no children, and *nonleaf nodes*, intermediate in the structure. The resolution of the decomposition is the number of times the decomposition process is applied, in this case 3, and the blocks resulting from a decomposition are labelled NW, NE, SW, SE, as appropriate. The scheme is easily extended to grey-level images: an image composed of different brightness quadrants is decomposed and so on successively till obtaining a block constituted of a single brightness level.

Quadtrees are attractive because of their hierarchical structure, making it easy to implement split and merge operations and allowing, among other things, some sort of resolution independent geometrical operations. However, quadtrees are interesting in the context of research for this thesis because of the possibility of compression by hiding redundancies in the encoding process. If a largeish block of an image, say 8x8 square, has the same brightness all over, it could be

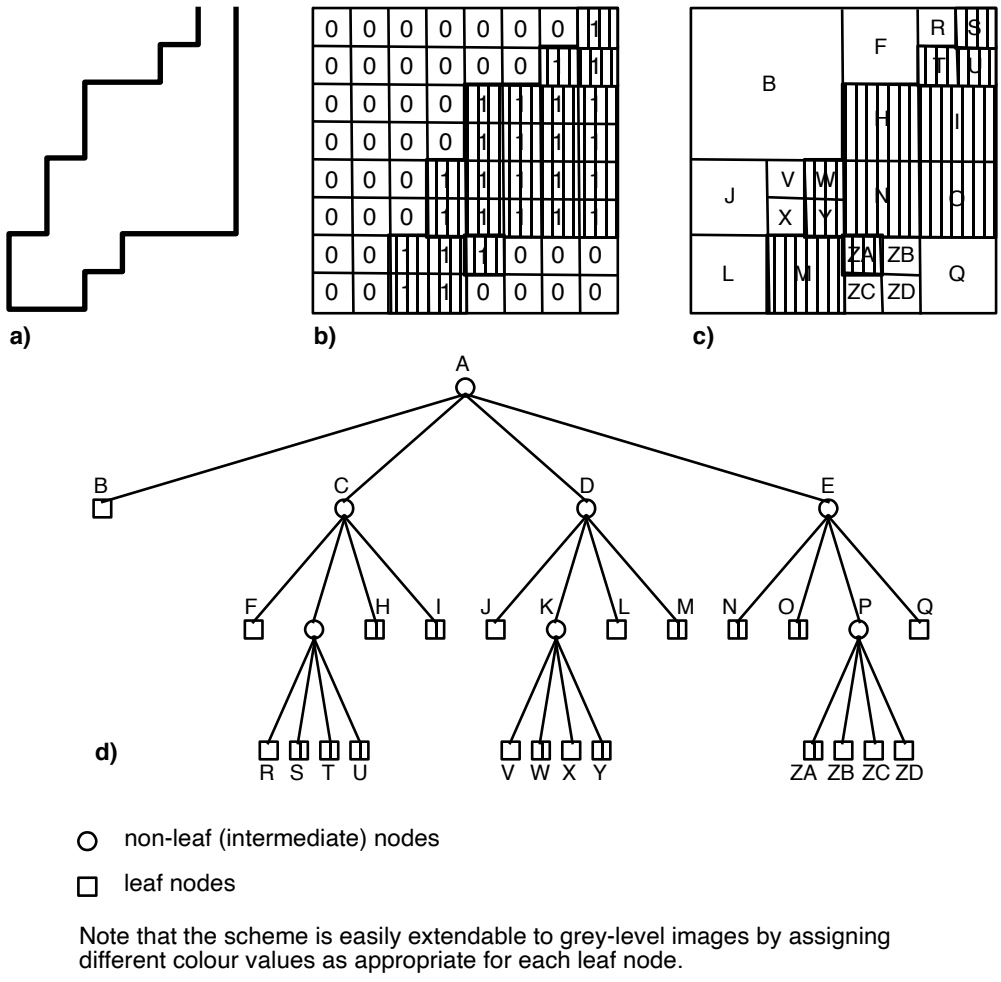


Figure 3.5: Basic mechanism of a quadtree.

coded by a quadtree as a single leaf node of level 3 (counted upwards, with level 0 corresponding to a single pixel), with an associated brightness.

Gargantini [Gar82] proposed a more efficient way of representing quadtrees that avoided the use of pointers normally used in tree-like structures and made an integer number represent each possible node in the tree. The binary representation of this number provides the key to localise the correspondent block of pixels in the image through the use of the Morton numbering sequence [Mor66]. By following this sequence in a 2-dimensional array one gets the nodes ordered exactly as if traversing a quadtree in its standard representation (see Section 4.6.6 for more details).

Unfortunately there are few reports in the literature on the compression figures that may be obtained when encoding real-world images with quadtrees. Lauzon *et al* [LMKG85] cite statistics for some images, but the only one derived from real-world scenes seems to be the well known "Baboon" face image. This is a 512x512 image at 4 bits/pixel, which is shallow in comparison to the images used for the research in this thesis. The latter are at least 6 bits/pixel, preferably 8 bits/pixel. This image was coded with 46,789 leaves, out of a maximum 262,144 for all single pixel leaves. On top of this 5.6:1 reduction there must be taken into account the number of bits necessary to code a leaf to get to the final compression rate. Obviously a deeper image (in the sense of more bits/pixel) will cause higher fragmentation and the number of leaves should increase, thereby worsening the quadtree encoding's compressing capabilities. Section 4.6.1 presents quadtrees more thoroughly and reports on experiments performed with real world images and their usefulness in image processing.

3.3.6 Subsampling

Subsampling is perhaps the most obvious way of achieving image compression. From a square of size 2x2 pixels one can take a single sample to represent the square, yielding a 4:1 reduction in data volume. In this case resolution is reduced by a factor of 2, yielding an image size of 256x256 instead of the original 512x512. Actually, subsampling consists of reducing the effective sampling frequency (at video acquisition time). In the context of this thesis, however, images are assumed already acquired at a predefined resolution of 512x512. Subsampling can then be applied over this digitised image.

Some image detail is inevitably lost when subsampling, and the process can

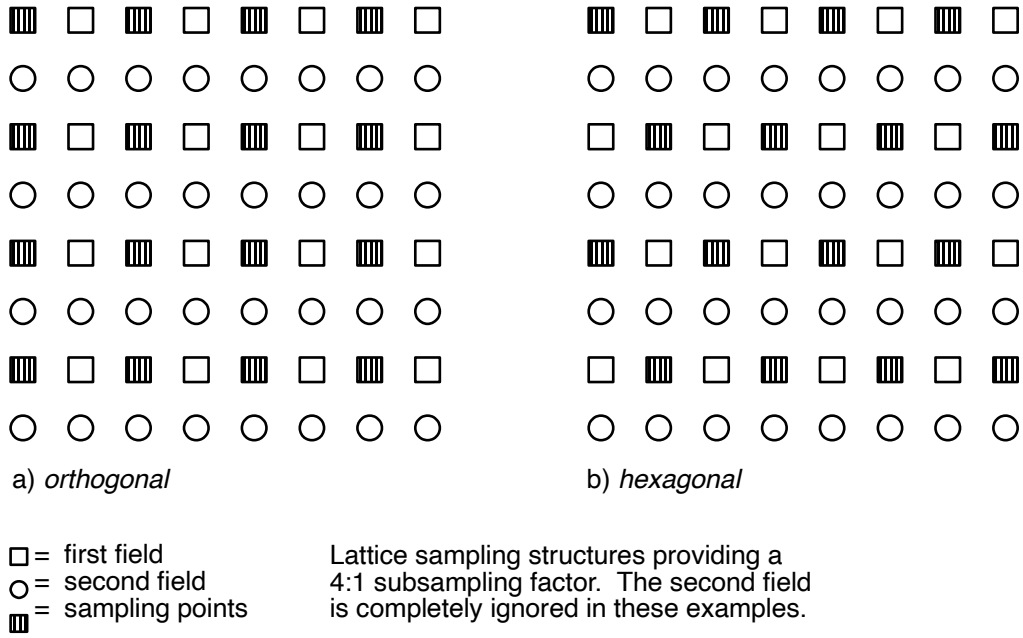


Figure 3.6: Orthogonal and hexagonal sampling patterns.

cause unwanted visual effects due to aliasing. Aliasing is an effect caused by sampling at less than twice the Nyquist rate (twice the highest frequency component of the original brightness function). Mathematically, aliasing happens by the folding-over of replicated spectra on top of the original spectrum, which one wishes to isolate to reconstruct the original function. As a result components of high frequency energy from the folded signal are positioned on low frequency values. Aliasing appears visually as low frequency artifacts or patterns on an image.

Prefiltering (smoothing) will usually avoid aliasing by removing the high frequency components responsible for the aliasing artifacts [Dub85]. An alternative way to avoid aliasing is to choose nonorthogonal sampling patterns like an hexagonal lattice, for example, in which alias components do not overlap with the baseband components and therefore can be removed by post-filtering [SP85, Dub85]. Sketches of both orthogonal and hexagonal lattice sampling structures are shown in Figure 3.6, both providing a 4:1 data reduction.

An image reconstructed from a subsampled image will appear jagged due to the lower resolution, and the jaggedness will increase as subsampling is increased.

A companion technique, interpolation (see Section 3.3.7), helps remove jaggedness and renders a more pleasing image for the observer, if not one of more quality (in the strict sense of a signal to noise ratio measurement).

Subsampling so far has only been discussed in the two-dimensional sense. It can also be applied three-dimensionally by including time (i.e. a sequence of frames) in the process. However, this extension falls into the scope of other techniques, such as frame replenishment and frame freezing, and so is not being considered here.

In summary, subsampling (together with interpolation on reconstruction) can sometimes be used to reduce image data. Obviously the image information loss becomes too high at higher compression rates and the resulting image may not be considered useful. Subsampling patterns of 2:1 and 4:1 seem to be the only useful ones, but this highly depends on the interpolation function used.

3.3.7 Interpolation

Reconstruction from a subsampled picture needs interpolation so that higher resolution is restored. Interpolation can demand a great deal of computation, it all depends on how sophisticated the interpolation function is.

The sampling process in the frequency domain causes the spectrum of the image function being sampled to become periodic. The spectrum of the original function is replicated to infinity, separated by distances on the frequency axis equal to the inverse of the sampling interval.

Interpolation with the ideal function $sin(x)/x$ provides an exact reconstruction of the original function, provided the sampling is done at least at the Nyquist rate. The replicated spectra are removed, leaving the original function. In practice, however, this ideal function is not implementable in an imaging system, due to the negative values that the function assumes. However, the ideal interpolation process can be approximated by several ways. These methods can be classified according to the order of the polynomials used:

zero-order The interpolation is done by convolving a pulse function with the array of brightness samples. In the case of images, the action consists only of propagating the sampled grey level value to the corresponding neighbouring pixels whose values one wishes to fill. When subsampling, a simple approach takes the sampled brightness value as the value to be propagated

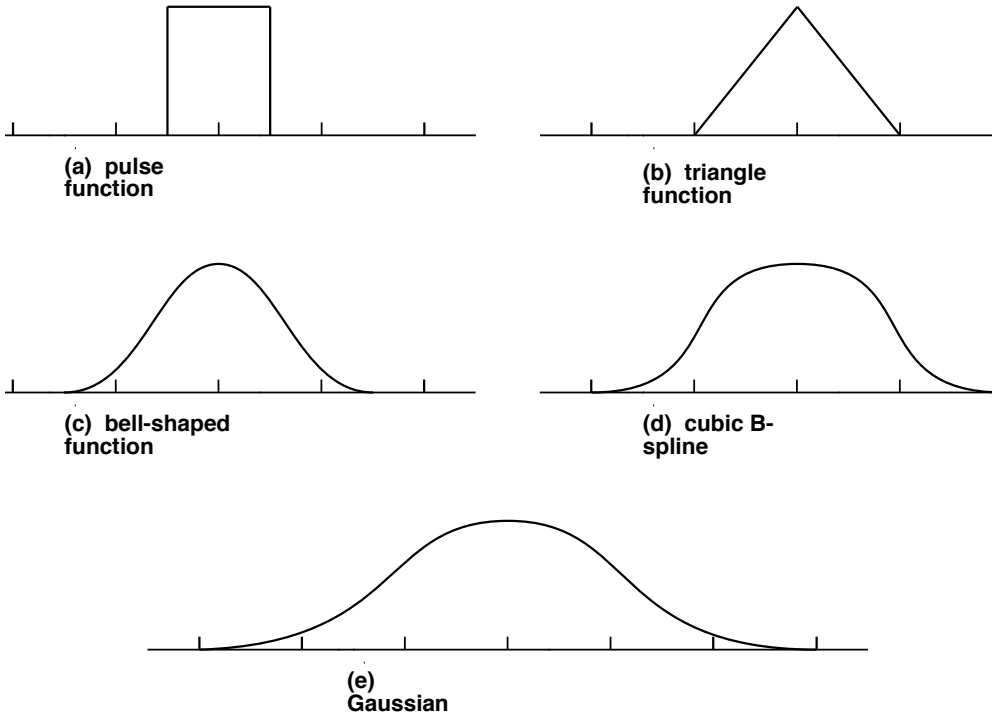


Figure 3.7: Several interpolation functions.

(sample deletion); however, to avoid aliasing it is good practice to use not an individual pixel value, but the average brightness of the region to which the subsampling applies. Pratt [Pra78] reports good antialiasing results by using this technique. The basic compression scheme developed in this thesis (BCS, Section 5) uses the latter approach. Zero-order interpolation is by far the easiest interpolation method, but results are poor to say the least. See Figure 3.7.a shows the shape of the interpolation function (pulse) and Figure 3.8.b an example of zero-order interpolation.

first-order Linear interpolation is accomplished by the use of a triangle function. It should be noted that the triangle function is obtained by convolving a pulse with itself (Figure 3.8). In image applications, this is translated by convolving a cone-shaped kernel with the image, as in Figure 3.9. The result is a much better rendition of what the original image was, as compared to a straight zero-order interpolation. Figure 3.10 shows the preferred method

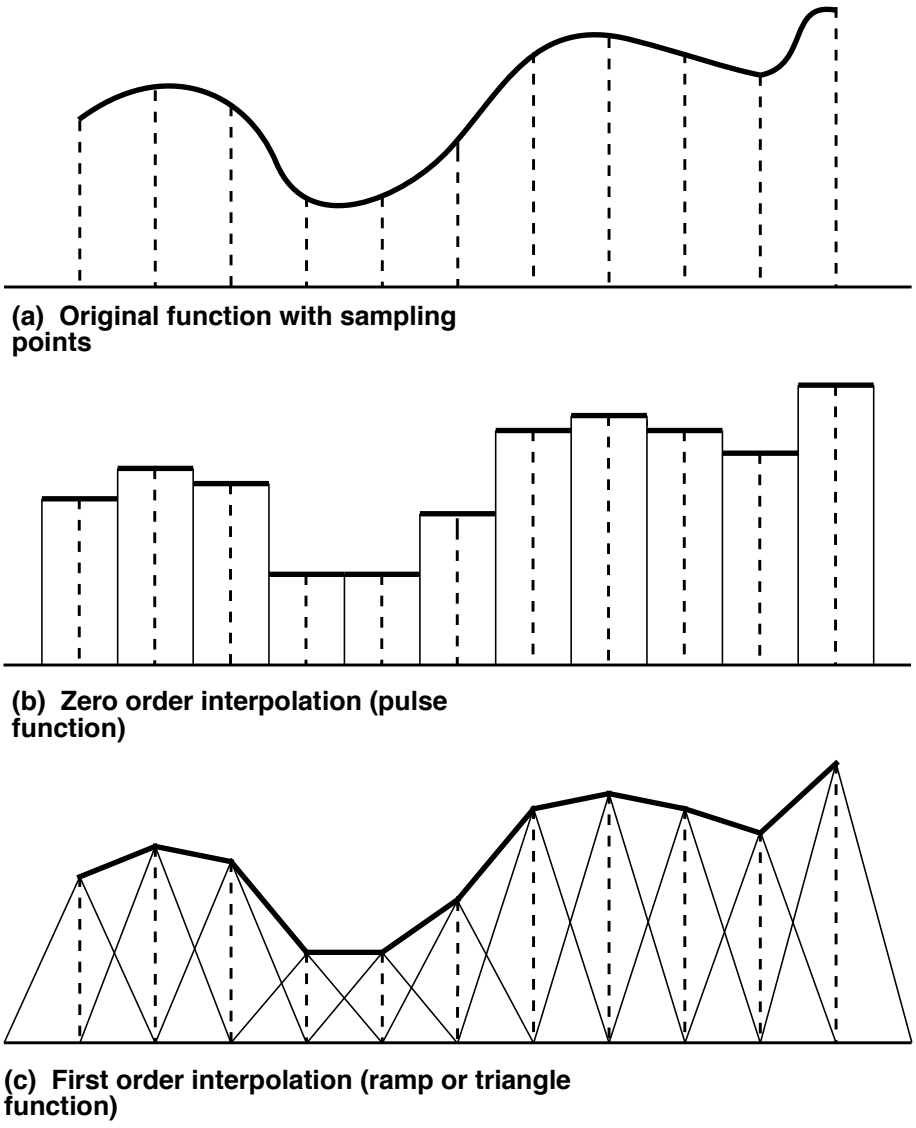


Figure 3.8: Zero and first-order interpolation.

0	0	0	0	0	0	0
0	1	4	5	4	1	0
0	4	8	10	8	4	0
0	5	10	15	10	5	0
0	4	8	10	8	4	0
0	1	4	5	4	1	0
0	0	0	0	0	0	0

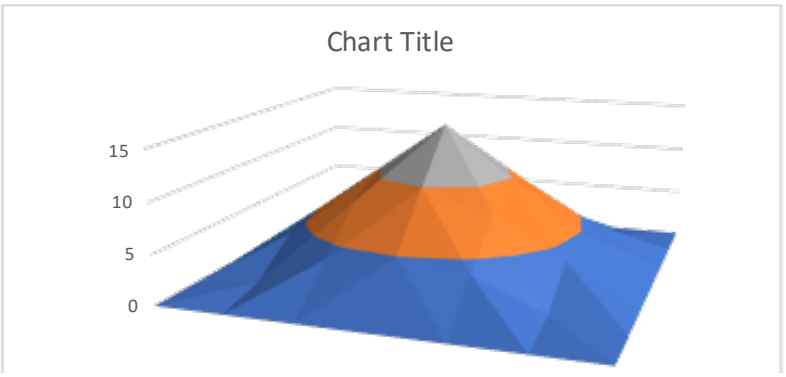


Figure 3.9: Cone-shaped kernel for image applications.

when extending linear interpolation to the bi-dimensional case, called bilinear interpolation. The main disadvantage of linear interpolation comes at the finer detail level, since there are discontinuities in the derivative of the generated surface.

bell-shaped Convolving a triangle function with the pulse function yields a bell-shaped function (quadratic curve). It provides a smoother interpolation, but also requires more computation. See Figure 3.7.c.

cubic B-spline By convolving the bell-shaped function again with the pulse function one obtains a third order polynomial. The cubic B-spline is often cited due to its inherent properties of smoothness at its extremities. See Figure 3.7.d.

The sequence described above converges to a Gaussian-shaped function (see Figure 3.7.e), giving smoother results, but each step in this chain means that more computation is involved. In image applications a cone (triangle) interpolation is approximated with a 3x3 kernel, a bell-shaped one requires at least a 5x5 kernel and a bicubic spline surface at least a 7x7 kernel. Clearly the computation involved increases exponentially with the size of the kernel, that is, with the sophistication of the interpolation function.

In general, subsampling coupled with interpolation is suitable for a certain class of applications only, due to the loss of detail incurred. 2:1 and 4:1 compression rates achieve acceptable image quality, and 16:1 might be used under certain situations.

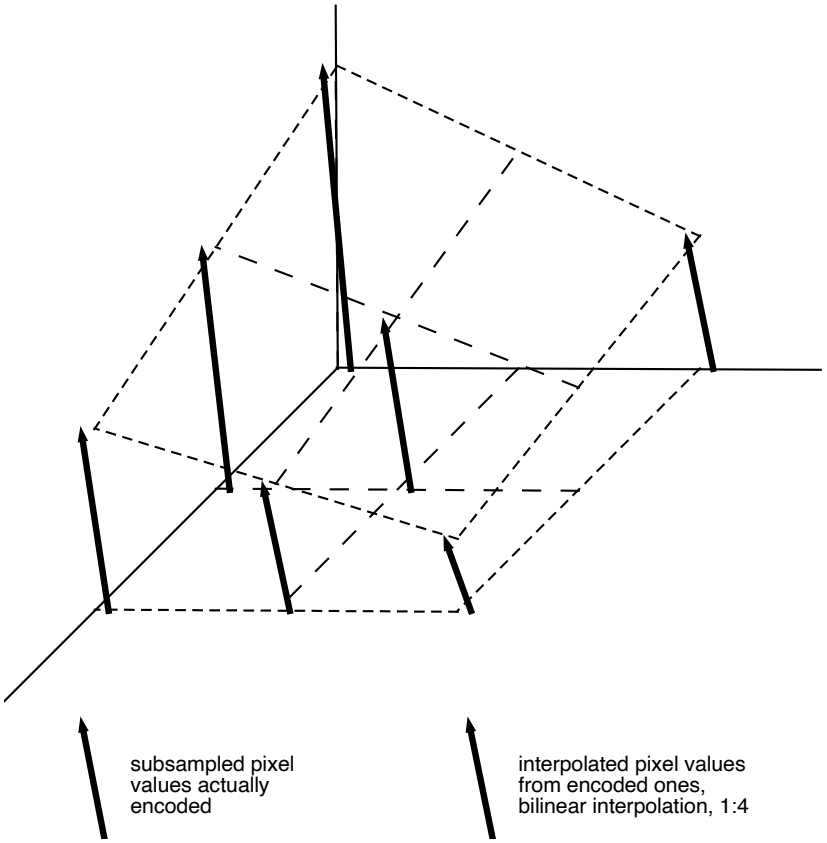


Figure 3.10: Bilinear interpolation.

3.3.8 Feature Coding

The idea behind feature coding is to depart from a pixel-based approach to coding and to attempt to mimic the way a human observer's visual system works. Rather than by taking into account individual pixels, our visual system searches for image features that distinguish themselves from an otherwise plain background, such as edges (high gradients in brightness) and textures (subregions delimited by edges, usually containing repetitive patterns). Combining these features tends to form recognizable objects which can be compared with the vast number of similar entities stored in the observer's memory, allowing the observer to work out an interpretation of the scene.

The motivation behind feature coding stems from the fact that the end destination of all the image compression and processing effort is the visual system. It seems therefore logical to search for coding methods based on pictorial descriptions of the image. The main techniques studied so far can be grouped as luminance contour coding, edge coding and texture coding.

3.3.8.1 Luminance Contour Coding

This technique considers the image as being formed by layers of planes each of which is a constant grey level, akin to contour-based topographic maps. What is coded are the boundaries or 'contours' of grey levels thus formed.

Real world images tend to include noise which disrupts most contours. This is good for image quality, but bad for coding the contours. In order to make any use of contour coding good filtering must be applied to get rid of noise and still keep most of the image detail, a task well suited to the use of nonlinear filters such as the sigma filter [Lee83].

To code the boundaries the suggested technique is chain coding: contiguous pixels of a boundary are coded one after the other by means of a simple directional code, three bits being needed to properly identify the correct direction.

The technique is most effective for binary images (2 grey levels) and in general does not suit deep grey level images, since there are a lot of planes to code. Computational complexity of the method is high and as the reported performance gives 2:1 or lower compression rates (4 bits/pixel or more), it is of no interest for the research in this thesis.

3.3.8.2 Edge Coding

Edge coding attempts to treat the image signal separately as high and low frequency components. The rationale for this is that if one can, after compression and decompression, reproduce edges well then the resultant image generally appears to be sharp and of good quality [NL80]. Tests have also shown that the human visual system is highly sensitive to edge positional inaccuracies but quite compliant with large amplitude differences [SHT72]. This helps with coding the high frequency portion of the image, i.e. the edges, more like a binary image, where efficient coding is more likely attainable. The low frequency information is coded in a more relaxed way since it carries much less information [YS77].

Reports give good compression rates for grey-level images, in the range of 0.5 to 1 bits/pixel, but processing load is high due to complexity and number of steps needed, further complicated by susceptibility of edge detection mechanisms to noise. These steps are:

- edges are obtained through gradient or Laplacian filters,
- a suitable binary coding method is employed to carry the edge information, augmented with some edge amplitude information,
- a low pass filter is used to extract the low frequency components,
- and another suitable method is used to code the low frequency components, typically subsampling or transform coding.

A recent technique based on edge coding, *directional decomposition based coding* [KIK85], ensures edge preservation in the best possible way, accomplishing this by use of directional filters in several directions. Several other tricks are used in the coding process to achieve compression rates in the high tens to one range, that is, data rates as low as 0.1 bits/pixel. However, due to cost-effective real-time requirements, edge coding is not of use in this thesis.

3.3.8.3 Texture Coding

Texture coding is similar to edge coding. Edges are detected so that the image can be segmented into regions of reasonably smooth texture. The shape of the grey

level distribution[3] in those regions can be coded, for instance, by smooth two-dimensional polynomial functions, usually of low order (0, 1 or 2). Performance of each polynomial is measured by the mean square error. Of the polynomials considered the lowest cost (in terms of bits/pixel) approximation, whose error is less than a given threshold, is chosen to represent the region. Granularity[4] is usually removed in a preprocessing operation to ease the segmentation procedure, but a pseudo-random signal is added to the image in the end of the process to restore a natural feel to the image. Compression rates go up to several tens to one [KIK85], corresponding to data rates in the 0.15 to 0.5 bits/pixel range.

As with all feature-based coding techniques, cost-effective real-time implementation is hard to achieve.

3.4 Predictive Coding

Techniques to be examined in this section have in common the coding of a given pixel as a prediction based on the past history of already coded pixels. The actual information to be coded is the difference between the current pixel and the predicted value, properly quantized, and then sent to the communications channel.

3.4.1 Deltamodulation

Deltamodulation is the simplest incarnation of predictive coding, dating back to 1952 [Jag52]. A band limited video signal is fed to a comparator which determines the polarity of a pulse generator whose output is integrated. If the input signal is higher than the integrator output at the sampling moment the generated pulse is positive, otherwise it is negative. At the receiver side the pulses are collected and integrated to reconstruct the video signal, conveniently smoothed by a low-pass filter. Figure 3.11 shows a diagram of a deltamodulation coder and how it works. It also points out the main problems it has, namely slope overload and granularity at constant brightness regions: a trade-off must be chosen between larger pulse steps (less slope overload but increased granularity due to quantization error) and smaller ones (less granularity but more slope overload). A low-pass filter at reconstruction always helps to decrease granularity.

[3] this shape can be thought of as a 3D plot.

[4] *salt and pepper.* noise that occurs in low gradient regions, see Figure 3.11.

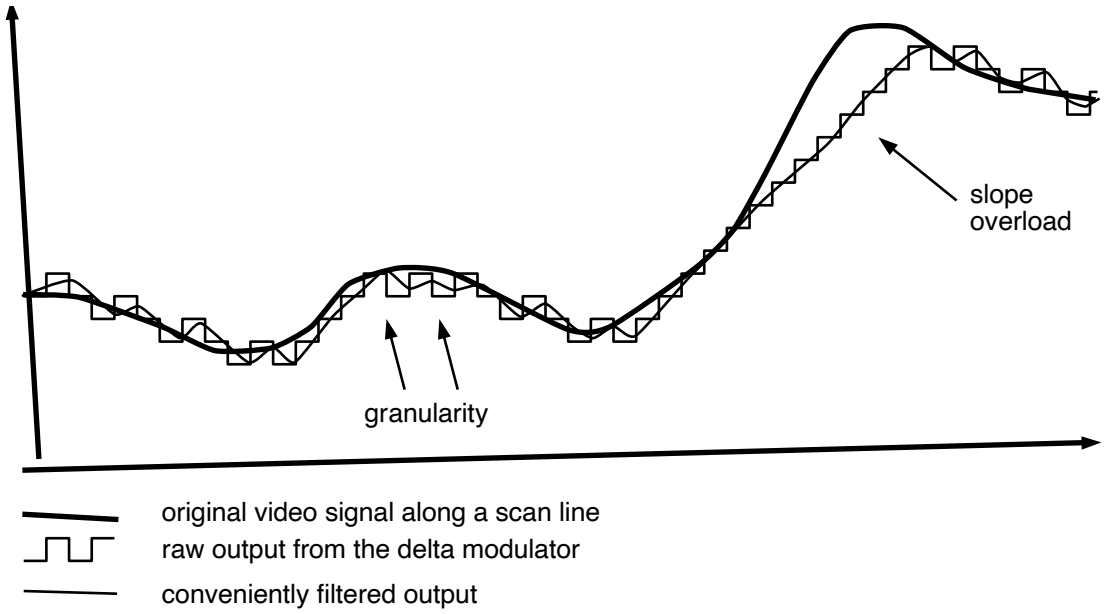


Figure 3.11: Deltamodulation.

This technique appeared very early in television transmission history due to its suitability for analog implementations. Departing from its simplicity, extensions have been proposed such as the *tristate deltamodulation* [PCB76] and others [SGS71]. The tristate DM has three levels, for "level", "rise" and "fall", and the output can be coded either by a straight binary code, or by variable length Huffman or run-length code. Most pixels are likely to stay "level", allowing the latter two methods to fare better than a binary code: performance is reported down to the 1 bit/pixel level, but with very poor quality. To get better quality it is necessary to sample the image at a quicker rate, with the side-effect of lowering the compression rate. A different sampling rate is outside the scope of this study.

The extensions cited above, however, already characterise the method as a special case of DPCM, to be examined in the next section.

3.4.2 DPCM

A simple Differential Pulse Code Modulation (DPCM) works by predicting the current pixel value based on the previously sampled value along a scan line; the difference between the two values is then properly quantized and coded for transmission. The difference signal is usually quantized to eight levels and coded with

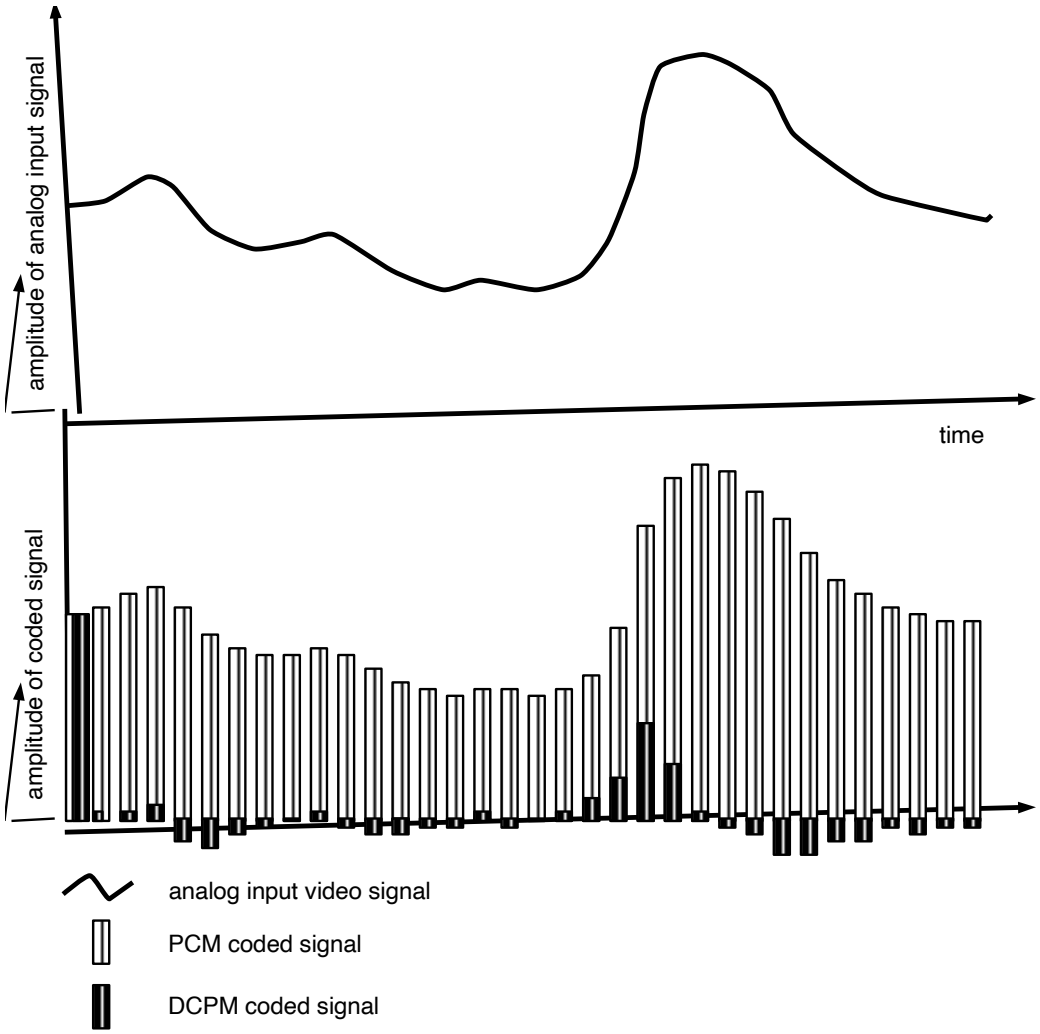


Figure 3.12: Comparison of PCM and DPCM signal amplitudes to be coded.

a 3-bit code, yielding a good quality image at a quite easily accomplished compression rate of 3 bits/pixel (the reconstruction process works symmetrically). Figure 3.12 shows a comparison of the amplitude of the signals to be coded by both PCM and DPCM in a typical video signal. DPCM shares the basic mechanism of deltamodulation and is bound to the same constraints deltamodulation is, namely slope overload and granularity.

Given a fixed rate in bits per pixel, one can get better results for the image quality by using a well-designed quantizer to make the most of the few difference levels available. It is generally considered that a 3 bits/pixel DCPM coder in this situation performs equivalently to a 6-bit PCM coder, with just moderate

effects on high gradient edges. Furthermore, since the difference values are likely to stay at the smaller quantized levels, a statistical coding applied to the stream of difference values accomplishes some extra compression. Simulations indicate figures around 2.5 bits/pixel [Cho71].

Predictors of a variety of forms can be devised for DPCM coding, and can be extended to the general two-dimensional case. Taking a linear combination of previous pixel values both horizontally and vertically results in slightly better prediction, which can be exploited by a statistical coder at the end [Har52].

In general it is said that a predictor that uses n previously scanned pixels is an nth-order predictor. Experiments have shown that gains in image quality (i.e. less coding error) quickly stabilise around the level of a third-order predictor, meaning that there is no point in using more than about 3 previous values for prediction.

The importance of DPCM, both for image compression in general and for the aims of this project, relies on two main factors:

- it is a very simple technique to implement, requiring very little computational power per pixel.
- It suits very well as a supporting technique for other simple or more complex compression methods.

In summary, although the technique does not perform wonders by itself, it is generally taken for granted as either a front-end or an intermediate step in other composite compression methods. Data rates to be expected, as cited above, are down to 3 or even 2.5 bits/pixel (compression rates of 2.67:1 and 3.2:1, respectively).

3.4.3 Interframe Conditional Replenishment

Conditional replenishment tries to remove temporal redundancy, that is, the redundancy between two contiguous frames in an image sequence [Mou69]. Simple detection of moving areas is performed: the difference signal between the brightness values at the same pixel position in both frames is compared with a predetermined threshold. A difference signal greater than that threshold is quantized and coded for transmission. At reconstruction, stationary area pixels are copied

from the previous frame (remaining unchanged) and moving area ones are replenished with the difference signal, i.e. the difference signal is added to the previous frame's pixel value.

Since this coding scheme produces a data rate dependent on the amount of moving regions in the scene, and transmission channels usually require a steady rate for better efficiency, a buffer is needed before the transmission process. So is a buffer control policy, so that buffer overflow is avoided. The easiest way to avoid overflow is to temporarily raise the threshold used for movement detection until the buffer level comes back to a comfortable level.

A variation on the basic idea is called *conditional block replenishment* and works quite similarly, the difference being that instead of working with single pixels over two frames, both the movement detection and the updating mechanism work with blocks of image of predetermined size and shape. The most practical shape is the square, at sizes typically from 4x4 to 16x16 pixels.

Experiments performed in the course of this thesis and reported in Chapter 5, Figure 5.8, revealed that significant compression rates can be achieved with the above method alone, ranging from 2:1 to 5:1 (50% to 20%). Notice though, that the real figures are really better than the ones cited above. The measurements refer only to the conditional part of the method (movement detection), with the replenishment part being performed with straight PCM coding. For normal scenes the lower figure is likely to occur. For high movement content scenes the higher figure is more likely.

By adopting a predefined block size one can capitalise on the available structuring to add other compression techniques to the basic conditional block replenishment. Besides its already significant compression rate (which depends on the amount of movement in the scene), another point in favour of this technique is that it demands little computational power. As mentioned previously conditional block replenishment needs roughly one arithmetic operation per pixel per frame, which is very cheap indeed when compared to most other techniques.

3.4.4 Motion Compensation

The term *motion compensation* can be applied not only to predictive coding but also to transform and interpolative interframe coding. I will, however, present a summary of how it is applied to predictive coding only (i.e. DPCM), thereby covering the salient features of motion compensation.

Moving regions in a scene cause havoc when one tries to compress an image sequence by removing temporal redundancy on a frame-to-frame basis. A logical path to follow is then to try and compensate for the motion in these regions. Movement in general can have many components, but due to the computational task involved only the translational component has been considered for real-time implementation so far. By estimating a displacement vector for an object moving between two successive frames, a motion-compensated predictor can find a prediction that more closely resembles the original data for that region. An error signal (resulting from comparing the original and predicted objects) is then more easily coded than the original object, due to the amplitudes involved being smaller in average. The advantage comes from the fact that a higher compression rate may be used and still return distortion figures (end image quality) at a par with a DPCM scheme working at a higher data rate.

The basic mechanism for motion-compensated coding works as follows: an image region in the current frame is coded by finding a similarly shaped region in the previous frame. The best match in the previous frame is then subtracted in a pixel basis from the original region in the current frame. The error signal thus generated is coded, together with the displacement vector between both regions.

The review by Musmann *et al* [MPG85] presents in detail the displacement estimation algorithms that have already been studied. They are classified into two main categories:

1. *recursive algorithms*, usually associated with backward motion estimation, where the displacement vector is estimated by recursive methods: starting with a first estimate, an improved estimate is produced by attempting to minimise a criterion function based on the gradient of the frame difference in relation to the displacement. The process is repeated till a minimum for the frame difference is found. Different methods provide fewer number of iterations.

2. *block-matching algorithms*, usually associated with forward motion estimation, where the displacement vector is estimated basically by trial and error over a predefined search region and then transmitted. A certain comparison criterion is used to find the best match. Simplifying assumptions are used by the various methods to speed up the search operation.

Compression figures for each of the two approaches above are not mentioned

separately since both of them yield basically similar performance. As a general guide, motion-compensated DPCM gives roughly twice the compression rates of standard DPCM, with figures ranging from 0.5 to 1 bits/pixel. A more detailed explanation of both approaches is given in the next two sections.

3.4.4.1 Recursive Algorithms

The original algorithm by Netravali *et al* [NR79] attempts to find the displacement by updating the estimates at each iteration by a term proportional to the gradient of the frame difference in relation to the displacement. The evaluation of the gradient requires interpolation to obtain luminance values for nonintegral displacements (between pixel positions). Netravali *et al* also propose simplified versions to overcome the above difficulties. The algorithm requires a considerable number of iterations to converge.

Other algorithms have been developed to estimate the displacement more quickly by substituting variables for the constant used by Netravali *et al* in the gradient term. Among these algorithms are one that uses a Newton-Raphson method [Ber82], one by Cafforio *et al* [CR83] and one by Bergmann [Ber84]. Of these, Bergmann's showed faster convergence in the recursive estimation process with a collection of test sequences [Ber84]. In general recursive algorithms are complex, requiring a great deal of computation.

3.4.4.2 Block-Matching Algorithms

In block-matching algorithms the idea is to attempt to find the best match by searching over a predefined search region. Displacement estimation is done by measuring a cross-correlation function. A block of pixels in the current frame is taken and correlated with a block of pixels in the previous frame. The search area is centred around the original block. If the block is of size 7x7 pixels and the maximum displacement the algorithm is able to track is set to $dm = 10$ pixels, the resulting search area is a square of 27x27 pixels. It is clear that, whatever the computational cost of the cross-correlation function, the total computational cost is way too excessive: the number of cross-correlation calculations necessary to cover the search area is $(2dm + 1)^2$, in this case 441.

To simplify the matching process Jain *et al* [JJ81] used a simple mean-square error measure, others have used the mean of the absolute frame difference and so on. The searching process was also much speeded up by intelligent search

methods like the 2D-logarithmic search [JJ81], Koga's search [KIH+81] and the conjugate direction search [SR84].

All these methods are founded on the assumption that the matching criterion increases (i.e. distortion worsens) monotonically as the search moves away from the direction of minimum distortion. At each step the minimum distortion direction is sought by checking at a number of basic possible directions away from the starting point and trying to cover the whole 360 degrees around it. To detect local matching criterion minima (i.e. local best matches but not absolute best matches over the whole search area) so that they can be avoided, the searching process cannot be speeded up by fast methods such as the ones described below. The success of these methods with real data suggest that local minima are rare and monotonicity is a good trade-off assumption: the few times one is found and the very best match is missed, little harm is done to overall performance.

The 2D-logarithmic search takes into account the central point of a cross-shaped search strategy and goes about by starting with a longer distance between search points. When the minimum is either at the center or at the border of the search area the distance is reduced.

Koga's search looks at eight search points around the origin (four directions of a cross-shaped search and four corners) separated by a longer distance still (covering approximately half the search area). In the second step the distance is reduced and so on till the best match is found. In a search area whose maximum search displacement is 6 pixels the best match is found at the third step.

The conjugate direction search takes on a different approach. It looks first at the horizontal direction only to determine the way to go for the minimum distortion, till it is found, and then looking at the vertical direction only, till the definitive displacement is found. That way each step in the search needs only two distortion calculations, instead of four for the previous search methods.

Musmann *et al* make a comparison of the above methods and indicate that Koga's presents the least number of sequential steps in the search, whereas the conjugate direction search method needs the least number of distortion calculations. The conclusion is that with sequential processing the conjugate direction search is faster, but with parallel processing available the first two methods are potentially faster. Koga's method is better for higher number of processors since it can make use of eight calculations simultaneously. Srinivasan *et al* [SR84] also found out that the distortion criterion makes little influence on the outcome of

the search, therefore further savings in processing load can be made by using a simple mean of absolute difference instead of quadratic methods, since almost no multiplications or divisions are required.

3.4.5 Quantization and Predictive Techniques

Basic to image acquisition is the quantization process, by which an analog signal coming from a video camera is digitised to a fixed number of levels (typically 256). To achieve image compression, however, several techniques subject the image information to further quantization to reduce the number of levels available, and therefore the number of bits necessary to code that information.

Max [Max60] studied optimum quantization based on statistical features, and many coders have benefitted from this work. An optimum quantizer for a DPCM system allocates more difference levels around the value zero, since that is the region where most of the differences will occur and where a better quantization will contribute most for a better reconstruction. Also, if variable-length coding is being used, those are the differences that should receive the smallest code words, since they occur most often.

Most of the works with optimum quantizers have, however, considered previous pixel DPCM coding only. Little consideration has been given to a more general 2D case and to interframe prediction. Among existing variations on quantization are: i) the quantization process may also be adaptive to local image statistics, to match more closely the required levels of image fidelity; and ii) the quantizer design might be based not on statistical features but on psychovisual criteria, derived from new understanding about the workings of the human visual system.

For the purposes of this research, though, it is felt that the study of more elaborate quantization methods will return very little benefit overall, as compared to the extra complexity and processing load usually involved. The preferred action is therefore to stick to standard Max quantization whenever quantization is required.

3.5 Transform Coding

3.5.1 Introduction

Transform coding of images consists in processing an image by a suitable mathematical transform and quantizing and coding the resulting coefficients for transmission. The concept was developed first by applying the Fourier transform and noticing that for real world images most of these coefficients are of relatively low magnitude, and more importantly, that the energy distribution through these coefficients is much more suitable for coding [Pra78, GW77]. In fact, the energy tends to be clustered into a small number of coefficients, usually in one corner of the two-dimensional array of coefficients, as a result of the pixel-to-pixel correlation present in natural images. Since the way image compression basically works is by trying to remove redundancy, it is clear that such a transform process already does much of the work needed. Many transform techniques can be used in transform coding today as a result of people studying how to get better compression and quality with the least calculation, and some choice is now available.

The transform process can be interpreted as a decomposition of the image data into a generalized two-dimensional spectrum. Each spectral component in the transform domain corresponds to the amount of energy of the spectral function within the original image, in much the same way as a function is decomposed into its series expansion, in an analogy to the one-dimensional case. The transform operation produces an array of coefficients the same size of the original image array. Basically the upper left corner coefficient corresponds to the average brightness (DC level) of the original image. Further elements of the coefficient array taken horizontally (down) or vertically (to the right) correspond to higher frequency horizontal or vertical components, respectively, of the original image.

Compression is achieved by dropping a number of the less significant coefficients and by grossly quantizing the more significant ones. For practical computational reasons the transform is not taken over the whole image: the image is first subdivided in small blocks of suitable size, normally in the range 4x4 to 32x32 pixels, and then the transformation is applied to each block independently.

3.5.2 Types of Transforms

There are several transforms able to achieve compression as described above. These can be classified into sinusoidal and non-sinusoidal unitary transforms:

Sinusoidal transforms : Karhunen-Loeve, Fourier, cosine, sine.

Non-sinusoidal transforms Hadamard, Haar, Slant (square-wave transforms).

The Karhunen-Loeve transform (KL) is optimal in the sense of packing the maximum energy into a few coefficients of the transform. Though obviously desirable for image compression purposes, it is generally difficult to calculate and no fast algorithm is available for it (see Section 3.5.3). The difficulty arises from the fact that the transform requires statistical knowledge of the image source. This knowledge is not easy to obtain and may vary according to the source material. Therefore in practice one has to use instead a transform that possesses a fast algorithm, like any of the remaining ones cited above.

The transform process can be seen as an operation involving the original block of pixels $F(j,k)$, which for the sake of simplicity is assumed to be a square of $M \times M = N$ pixels with $j,k = 0,1,2,\ldots,M-1$, a transform kernel denoted by A, and a transformed image block denoted by $\mathcal{F}(u,v)$, equally of size $M \times M = N$ pixels with $u,v = 0,1,2,\ldots,M-1$ (B denotes the inverse transform kernel). The forward transform is then

$$\mathcal{F}(u,v) = \sum_{j=0}^{M-1}\sum_{k=0}^{M-1} F(j,k)A(j,k;u,v)$$

and the inverse transform (to reconstruct the image) is

$$F(j,k) = \sum_{u=0}^{M-1}\sum_{v=0}^{M-1} \mathcal{F}(u,v)B(j,k;u,v)$$

Following the model above, here is a chart with the definitions of some popular transforms:

Fourier:

$$\mathcal{F}(u,v) = \frac{1}{M}\sum_{j=0}^{M-1}\sum_{k=0}^{M-1} F(j,k)\exp\left\{\frac{-2\pi i}{M}(uj+vk)\right\}$$

Cosine:

$$\mathcal{F}(u,v) = \frac{1}{2M^3} \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} F(j,k) \left[\cos\left(2j+1\right) u\pi\right] \left[\cos\left(2k+1\right) v\pi\right]$$

for u, v = 1, 2, ..., N - 1, and

$$\mathcal{F}(0,0) = \frac{1}{M} \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} F(j,k)$$

Hadamard:

$$\mathcal{F}(u,v) = \frac{1}{M} \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} F(j,k)(-1)^{p(j,k,u,v)}$$

where

$$M = 2^m$$

$$p(j,k,u,v) = \sum_{i=0}^{m-1} (u_i j_i + v_i k_i)$$

u_i, v_i, j_i and k_i are the bit states of the binary representations of u, v, j and k, respectively. The bit states for $u = 11$, for example, are $u_3 = 1, u_2 = 0, u_1 = 1$ and $u_0 = 1$.

3.5.3 Comparison of Some Transforms

The KL transform (or a non-fast version of the Fourier transform) needs N^2 multiplications to be calculated; if the blocks are of size, say, $N = 1024$, i.e. 32x32 pixels, that means over a million operations (1 Mops) for each block. For an image of 512x512 pixels there are 256 blocks to be processed, totalling 256 Mops for a single frame, showing very clearly the unsuitability of these transforms for real-time operation.

The next best transform, with distortion and compression rates very close to KL's, is the cosine transform (see below for some references). The cosine transform, however, has a fast algorithm for its calculation. The fast algorithms take advantage of common intermediate terms in the whole calculation, as well as other algebraic tricks, to reduce the number of calculations by orders of magnitude. The fast cosine transform needs, instead of N^2 multiplications, a much more feasible $3N/2(\log_2 N - 1) + 2$ additions and $N \log_2 N - (3N/2) + 4$ multiplications. For $N = 1024$ this represents, instead of 1 Mops, just over 22,500 ops [WH84]. Even

so, for images of 512x512 pixels, using a block size of 16x16 pixels, being shown at 25 frames/s (real time for the PAL TV System), the amount of computation necessary would be:

$$22500\ ops \cdot 256\ \frac{blocks}{frame} \cdot 25\ \frac{frames}{second} = 144\ \frac{Mops}{sec}$$

The above measure represents in fact 144 MFLOPS (million floating point operations per second).

One of the goals of this project was to achieve a certain amount of compression, utilising less computational power than a typical transform approach, preferably by using off-the-shelf components as opposed to dedicated VLSI designs. At the time this is being written (early 1990) state-of-the-art general purpose processors are still delivering 10 MFLOPS at best so it is clear that transform-based compression cannot be used with this approach, unless large-scale parallelism[5] is implemented. The use of general purpose processors is at odds with the desire that a future implementation would fit in a small physical space; what is expected is something like an add-on board to a workstation, and not a rack larger than the workstation itself.

Single chip implementations of Fourier transforms have recently been announced, such as the IMA A100 Cascade Signal Processor from Inmos [A1089]. This would allow the transform approach to be a viable alternative. These new introductions are claimed to perform transform operations in small blocks (8x8, 16x16) at standard frame rate for images of standard TV size (640x512).

Hadamard is another interesting transform, for its coefficients are all unity, only the sign of the components changes, and therefore only additions are necessary. This considerably simplifies the calculations, at the expense of higher distortion.

The performance of several transforms with first-order Markov image data has been widely publicised [Pra78, NL80]. Although it is difficult to extrapolate performance with real images from the published data, the results are of some interest: covering block sizes from 4x4 to 128x128 pixels and adopting the same strategy for coefficient dropping and quantization (i.e. the same coding bit rate), the KL and cosine transforms show almost identical behaviour, presenting the least distortion overall. For block sizes of 8x8 and 16x16 other transforms

[5] at least a few tens of general purpose processors.

(Fourier, Hadamard) show about twice the above mentioned distortion, measured as MSE (Mean Square Error) in percentage points.

Netravali *et al* [NL80] report that simulations on real pictures show improvements with increasing block size, for a given bit rate, but very little beyond 16x16 pixels. However, other reports on subjective quality with real images show no significant improvement for blocks larger than 4x4 pixels.

3.5.4 Selection of Coefficients

As discussed in Section 3.5.1, compression with transform coding is achieved when part of the coefficients are dropped or the number of quantization levels for a coefficient is reduced [Pra78]. There are basically two strategies to select samples from the transform coefficients to be coded: *threshold sampling* and *zonal sampling*.

With threshold sampling samples that are larger than a specified threshold are selected for coding. A minor inconvenience is that this generates a variable data rate: it is not known beforehand how many bits will be needed to code a particular block of image, further complicating the coding of the resultant stream of data. With a fixed bit allocation the number of bits required is known and makes for an easier design.

Zonal sampling is more widely used and the sample selection is performed according to geometric zones in the array of coefficients. The coefficients that most contribute to the overall energy carried by the picture are the low-frequency components, which are quantized according to their influence in the final image quality (lower frequency components receiving more levels)[6]. Variations on the above scheme include different strategies for zonal coding and, instead of statistical coding, the use of run-length coding to code the stream of bits from the coefficients.

Figure 3.13 shows a typical bit allocation for a 16x16 block cosine transform at ≈ 1 bit/pixel: the top left corner contains the average brightness for the given block of image and so receives good quantization (7 bits). Other coefficients in the neighbourhood of the top left corner also receive more bits for quantization; they are responsible for low frequency components taken bidirectionally from the image block: the ones to the right for the x direction and the ones to the bottom

[6] and levels whose occurrence is more likely are assigned smaller codewords, in case further statistical coding is applied to the resultant stream of data.

7	6	5	4	3	3	2	2	2	1	1	1	1	1	0	0
6	5	4	4	3	3	2	2	1	1	1	1	1	0	0	0
5	4	4	3	3	2	2	2	1	1	1	1	1	0	0	0
4	4	3	3	3	2	2	2	1	1	1	1	1	0	0	0
3	3	3	3	2	2	2	1	1	1	1	1	0	0	0	0
3	3	2	2	2	2	2	1	1	1	1	1	0	0	0	0
2	2	2	2	2	2	1	1	1	1	1	0	0	0	0	0
2	2	2	2	1	1	1	1	1	1	1	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

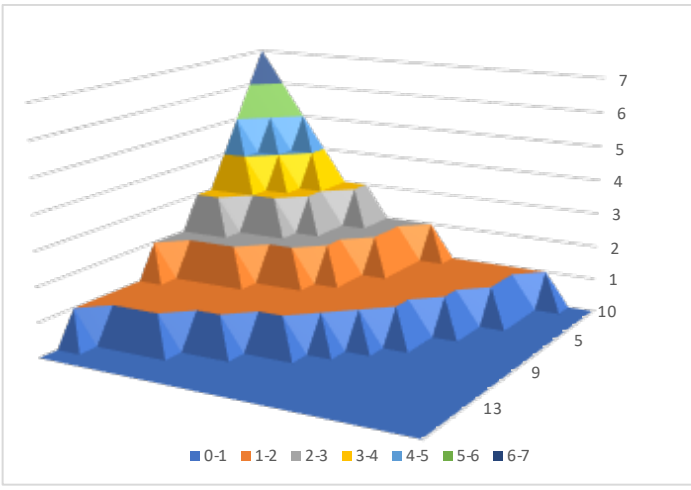


Figure 3.13: Typical bit allocation for a cosine transform (spatial coding).

for the y direction. See also how many high frequency components (towards the bottom right corner) are usually discarded (assigned 0 bits). Although high frequency components are indeed necessary in the perception of good quality, their overall weight energy-wise is very low when compared to the image energy carried by the lower frequency components, and so they are allocated fewer bits.

3.5.5 Adaptive Transform

The concept of adaptivity can be applied to transform coding in many of its possible forms. In principle the term "adaptivity" is related to changing parameters of the coding process according to how local image statistics change inside a frame, but it is easily extended to cover changes in a sequence of images that include movement. Items that can be made adaptive are:

- sample selection, be it by zonal or threshold coding.
- quantization levels of selected samples from the coefficient array.
- bit allocation for each coded sample.

Adaptivity improves coding efficiency, usually in the range of 20-50 percent, but as always the increase in coder complexity must be taken into account. Some research in the area of adaptivity can be found in [TW71, Gim75, NPM77, CS77].

3.5.6 Variations on Transform Coding

Several variations on the basic transform coding are possible. One of the more obvious extensions is three-dimensional coding, i.e. the incorporation of several frames to form 3D blocks of size, say, 8x8x8 pixels. The problems with this are (a) the large amount of memory needed to store sufficient frames to apply the transform coding and its inverse, and (b) the added computational overheads involved in the 3D case due to larger blocks.

Another variation is called hybrid coding, which combines transform coding with predictive coding techniques. The usual first step is to use transform coding and then apply DPCM techniques, possibly with adaptivity. It has been seen in Section 3.5.3 that larger blocks are better at removing redundancy, giving better quality at the same compression rate, but they are computationally more expensive. On the other hand, by adopting small blocks and using DPCM techniques and adaptivity, the overall processing load falls below that of a larger block transform, whilst returning better quality than a pure small block transform.

Transform coding may be adaptive and combined with other techniques, resulting in a broad range of data rates to be expected, going from 3 bits/pixel all the way up to 0.1 bits/pixel.

3.5.7 Conclusions

The figures presented in Section 3.5.3 suggest that the computational load involved with transform coding is too high for real-time implementation. The necessary hardware would make the basic concept of the Video Viewer not feasible, as it was laid down on Sections 1.1.1 and 2.2.

For this reason this research does not consider the possibility of using transform coding in the compression scheme sought. The applicability of transform coding to the Video Viewer concept could change in the near future though. With the release of chips that perform the FFT or the DCT transforms in real-time, the design of a real-time compression system becomes a lot easier.

3.6 Image Block Coding

As seen in Section 3.1, it was chosen to group Block Truncation Coding (BTC) and Vector Quantization (VQ) separately from other techniques.

These techniques have in common not the way an image block is described, but rather the departure from the way other block techniques usually work. Neither technique derives its block image description "directly" from the array of pixels, having a small, finite number of different possible reconstructions. What is meant by the previous statement can be explained as follows:

- BTC coding in fact directly derives an array of binary values from an array of pixel brightness, but there are only two possible values that can be assumed on reconstruction. Since the reconstructed image block can depart quite far from the original block, particularly as block size increases, it was chosen to label BTC as "not directly derived".

- VQ generates a finite collection of possible image block descriptions. Thus, it is possible that a reconstructed block presents features in the image not in the original block. VQ was also labelled as not being "directly derived".

Transform coding, on the other hand, is another technique that works with blocks of images, but the number of possible reconstructions is not limited as with BTC and VQ. This allows the coded information to more closely resemble the original image, and so transform coding is considered as "directly derived".

It might be worthwhile at this point to introduce the notation that is mentioned in the next sections. Imagine that to show a block of image in an illustration one uses, instead of halftoned brightness, a three-dimensional plot. The pixel array constitutes a plane (two dimensions) and the pixel brightness the third dimension. This plot describes a three-dimensional surface, which thereafter I shall call the *shape* of the image block. This notation will be useful in evaluating the ability of a compression technique to best describe an image block.

3.6.1 BTC (Block Truncation Coding)

Block Truncation Coding (BTC) is a relatively new technique, first published in 1977 [DM79]. It uses a two-level (one-bit) nonparametric quantizer to describe the shape of a block, adapting to the local properties of the image at a given block. The quantizer can be chosen to satisfy any desired criteria; one that preserves local statistics measures (first sample moments) gives very good quality with a rate range of 1 to 2 bits/pixel [DM79].

Taking for example an image block of $n \times n$ pixels, some characteristics of this image are taken, namely the average grey level, the sum of squares and the variance. Let $m = n^2$ and let $X_1, X_2, \ldots, X_m$ be the grey values of the pixels in a block. The first and second sample moments and the sample variance are given as follows:

$$\overline{X} = \frac{1}{m} \sum_{i=1}^{m} X_i$$

$$\overline{X^2} = \frac{1}{m} \sum_{i=1}^{m} {X_i}^2$$

$$\overline{\sigma^2} = \overline{X^2} - \overline{X}^2$$

A suitable threshold, X_{th}, and two output levels, a and b, are found so that:

$$\text{output} = \begin{cases} b & \text{if } X_i \geq X_{th} \\ a & \text{if } X_i < X_{th} \end{cases}$$

$$\text{for } i = 1, 2, \ldots, m.$$

To more easily visualise the algorithm one can use a quantizer that preserves the first two sample moments. First, a suitable threshold level is chosen such as $X_{th} = \overline{X}$. Then, let q equals the number of X_i's greater than the threshold X_{th}. To preserve the first two moments the following equations have to be solved,

$$\overline{X} = \frac{(m-q)a + qb}{m}$$

$$\overline{X^2} = \frac{(m-q)a^2 + qb^2}{m},$$

giving

$$a = \overline{X} - \overline{\sigma}\sqrt{\frac{q}{m-q}}$$

$$b = \overline{X} + \overline{\sigma}\sqrt{\frac{m-q}{q}}.$$

A block is then described by $\overline{X}$, $\overline{\sigma}$ and an $n \times n$ bitmap indicating which pixels are above or below the threshold. A simple implementation will assign 8 bits each to the average and to the variance, plus a bitmap consisting of n^2 bits. With 4×4 blocks that means a final rate of 2 bits/pixel. Figure 3.14 shows the result of a block of pixel values being encoded and reconstructed using second

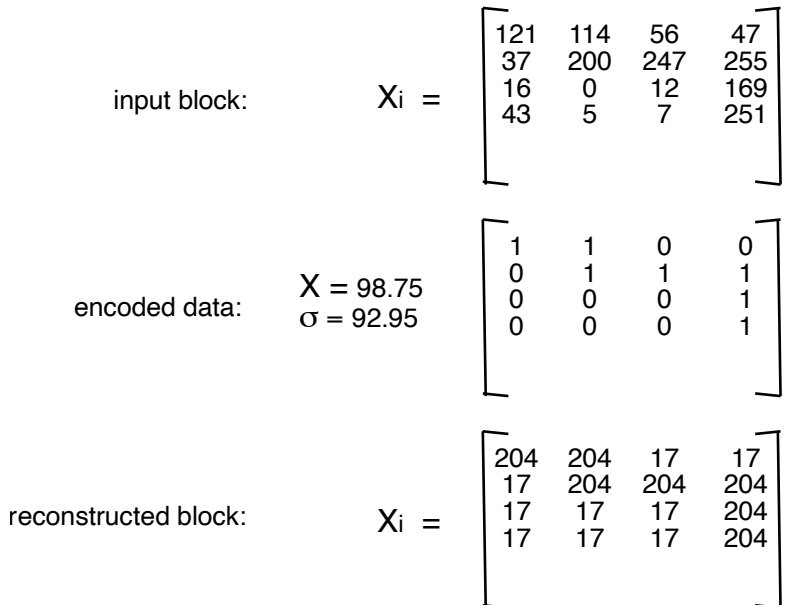


Figure 3.14: Example of BTC coding.

moment preserving BTC.

The results reported by Delp [DM79] referred to a quantizer that preserved the first 3 sample moments, allowing the threshold level to be a variable. Better performance is achieved, which leaves room for taking a few bits out of the encoding of the two parameters. In fact, a joint encoding of variance and average using 10 bits caused little degradation to the overall quality, achieving a data rate of 1.63 bits/pixel.

Computation complexity is now somewhat higher, since there is a third coefficient to be calculated for every block according to:

$$q = \frac{m}{2}\left[1 + A\sqrt{\frac{1}{A^2+4}}\right]$$

where

$$A = \frac{3\overline{X}\overline{X^2} - \overline{X^3}2(\overline{X})^3}{\overline{\sigma}^3}$$

$$\text{for } \overline{\sigma} \neq 0$$

Further work on the BTC encoding scheme reported rates down to 1.0 bits/pixel with good quality.

The results reported in this paper show good overall performance for the

moment preserving quantizer. Although a quantizer dedicated to minimising the Mean Square Error presented slightly better figures (on the order of one dB in the SNR figures), the visual impression favoured the moment preserving quantizer, probably due to its contrast enhancing side-effect.

Among the limitations of BTC coding are, due to its inherent non-parametric quantizer, the small range of block sizes that can yield fairly good image quality. All examples cited in related papers deal with block sizes up to 4×4 pixels, sometimes with elongated rectangular blocks of smaller area.

BTC is also employed in conjunction with other methods to achieve higher compression ratios and better quality. Usually this is done with transform coding, whose coefficients are then coded with either DPCM, VQ or BTC. However, these composite methods require too much processing power and therefore are not being considered here.

In summary, BTC coding can give reasonably good quality at rates in the 1-2 bits/pixel range. However, this is only true for smallish blocks (at most 4×4 pixels); for better compression rates blocks must be larger. BTC could be of good use as part of a composite scheme or as an intermediate stage with large blocks, before achieving full image quality.

3.6.2 VQ (Vector Quantization)

3.6.2.1 Introduction

One of the first processes involved in encoding images is quantization. The first device to make use of the video signal is the analog-to-digital converter (ADC), which produces digitised samples of the signal. The video information undergoes a transformation inside the camera's sensor and is output in analog form. However, in the ADC information has to occupy predefined finite levels. A sub-range of analog values are mapped to a single digital value, for the entire analog range.

The process above is called scalar quantization, since a unidimensional value is being mapped, but it could also be generalized to encode vectors of values (multidimensional quantization). Multidimensional quantizers have been studied by, among others, Gersho [Ger82], Nasrabadi and King [NK88]. Linde *et al* [LBG80] laid down the basic algorithm for the design of optimal vector quantizers, generalising Lloyd's basic algorithm for optimal PCM (scalar) quantizer design [Llo82].

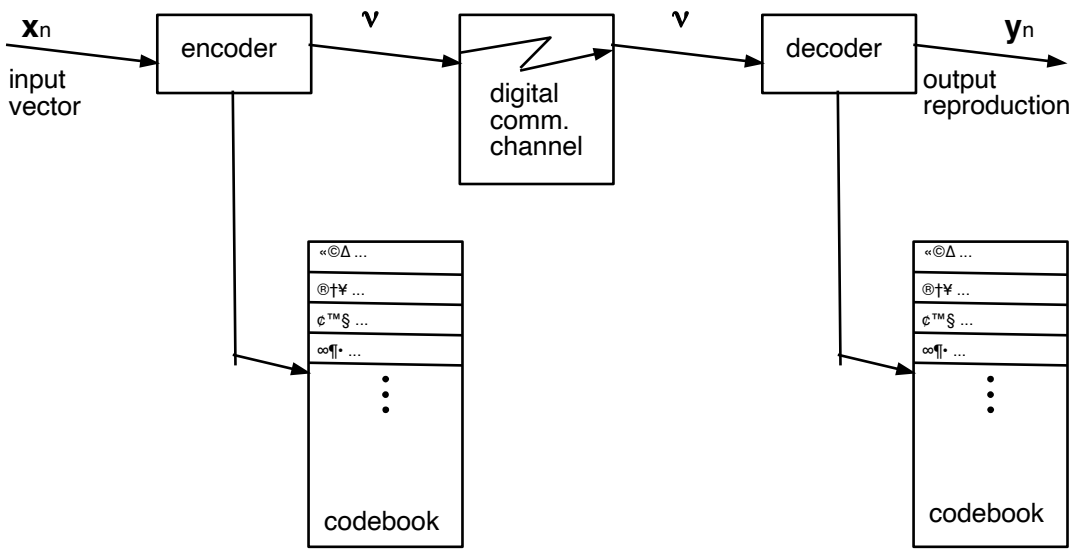


Figure 3.15: Basic vector quantization.

3.6.2.2 Basic Vector Quantization

A diagram describing vector quantization encoding is shown in Figure 3.15 [Gra84, NK88]. Note that in this diagram it is assumed the use of a noiseless channel. Real channels are usually noisy, but coupling a good error correction coding system yields a "clean" system, assured by information theory. In the scope of this project the channel is supposed to be "clean", as provided by the lower-level hardware and software communication systems.

Given a channel symbol set M, a k-dimensional memoryless vector quantizer (VQ) consists of two mapping functions: an encoder α which assigns each input vector $X = (x_0, x_1, \ldots, x_{k-1})$ a channel symbol $\nu = \alpha(X)$ in M, and a decoder β assigning to each channel symbol ν in M a value in a reproduction alphabet $\hat{Y}$.

The channel symbol set is often assumed to be a set of 2^R binary numbers $(0 \ldots 2^R - 1)$, for convenient binary handling. The reproduction alphabet is not necessarily the same as the input vector space (it may even be of a different dimension), but often is. If M has $m = 2^R$ elements then R is called the *rate* of the quantizer in bits per vector and $r = R/k$ is the rate in bits per sample (elements of input vectors).

Concluding the presentation of the notation commonly used in vector quantization, the reproduction alphabet will be referred as *reproduction codebook*, or even *codebook* of the quantizer. Each entry in the codebook, mapped by a channel

symbol, may be called *codeword* or *template*. The term *memoryless* refers to the lack of any feedback in the coding process as regards to past input vectors.

The operation of a simple VQ applied to imaging can be described in plain words as follows: i) the image array is partitioned into two-dimensional arrays (vectors) of a convenient size; ii) each of these image vectors is compared to a set of templates or patterns (usually stored in ROM); iii) a channel symbol (a key into the set) identifying the best match is then transmitted over the communication channel. At reception the decoder looks up the given channel symbol in the codebook (same set of templates as in the encoder) and reconstructs the image based on the template retrieved.

3.6.2.3 Distortion Measure

The idea of a vector quantization scheme is to produce a good reproduction sequence at a given rate R. Obviously there are limitations regarding the ratio of the total information content the input vectors can provide. As with all encoding schemes that are not of the information-preserving type, some sort of distortion measure is needed to achieve good image reconstruction. The distortion measure is needed to assess the coding performance and for the coder to run.

This measure could be defined as $d(X, Y)$, meaning the penalty involved in reproducing an input vector X as a reproduction vector Y. An average distortion $Ed(X, Y)$ for a single reproduction process is not so important as the long term sample average, i.e. taking the number of samples to infinity, as in

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} d(X_i, Y_i)$$

If the vector process is stationary and ergodic then the limit exists and equals an expectation $E(d(X, Y))$.

The trouble with this approach is that for real sources of data (speech and image, for instance) the vector process may be neither stationary nor ergodic. Gray [Gra84] treats this issue more thoroughly and points out that the practical approach of using long training sequences does allow one to design nearly optimal codes. If the code is designed on a *sufficiently long* training sequence and then used on future data produced by the same source, then one can expect a performance roughly equal to that obtained on training data. The most commonly

used distortion measure is the *squared error distortion*:

$$d(X,Y) = \sum_{i=0}^{k-1}(X_i - Y_i)^2,$$

3.6.2.4 Properties of Optimal Quantizers

The goal of vector quantization is ultimately to minimise the average distortion of the reconstructed data (here, an image array), given a reproduction alphabet $\hat{Y}$ and a memoryless decoder. The best encoder is the one that selects a minimum distortion codeword y in $\hat{Y}$. Likewise, given an encoder, the best decoder is the one that assigns to each channel symbol ν in M a vector which is the generalised centroid (or centre of gravity) of a collection of input vectors. These are the vectors that happened to be mapped into ν, and so the chosen reproduction vector gives the minimum distortion for the collection.

3.6.2.5 LBG Algorithm

Linde, Buzo and Gray [LBG80] extended Lloyd's original optimal design algorithm [Llo82] intended for a scalar random variable to the vectorial case. Starting with a training sequence and an initial codebook for the decoder, the algorithm (known as the LBG algorithm) is as follows:

step 1 - encode the training sequence into a sequence of channel symbols using the minimum distortion rule chosen for the decoder. Check the average distortion at the reproduction; if small enough quit.

step 2 - replace each codeword (corresponding to a channel symbol ν) of the current codebook by the centroid of all training vectors which mapped into that particular ν in the previous step. Go to step 1.

On the subject of optimality, it should be noted that this method does not guarantee optimal codebooks all the time. What it does is to yield a locally optimum codebook. Better codebooks could be found, for instance, by starting with better initial codebooks, or by introducing perturbations in the coding process. This would perhaps allow the coder to overcome a local maximum that was barring the minimisation, thus allowing the average distortion to reach a lower minimum. Experimentation is recommended to achieve good codebooks: it is worth trying with several initial codebooks.

3.6.2.6 Initial Codebook Generation

Fundamental to the VQ design algorithm is the existence of an initial codebook to start with. One can see two basic methods of arranging such initial codebook: either to start with a full codebook initialised with some values or to start with a smaller codebook and recursively build a full one. The simplest method starts with a full codebook and is as follows:

Random codes - by which some vectors from the training sequence are taken to initialise the codebook, usually the first vectors in the sequence, or better than that, widely spaced ones in the sequence. This should avoid local correlations that could badly affect the initial phases of the design and make it stick to a poor local minimum as regards to the distortion measure.

A method that starts with dimensionally smaller codewords is:

Product codes - the basic idea is to start with several codebooks of small dimensions and take the product codebook, that is, the set of all possible concatenations of codewords, taken one of each codebook. The resulting dimension is the sum of the dimensions of the original codebooks. Probably the simplest way of achieving this is by starting with a scalar codebook and apply this rule k times in succession to yield a k-dimensional codebook. After the first run the resultant product code is the original codebook, after the second it will consist of 2-dimensional vectors and so on. The final set of product codes is then shrunk (by discarding some of the codewords) to fit the codebook size originally planned.

Another method starts with small codebooks of the same dimension and generates more codewords till the planned size is reached:

Splitting - starting with the optimum codebook of rate 0, i.e. a single vector which is the centroid of the training sequence, one splits this codeword in two through adding a slight perturbation error, therefore ending up with y and $y + \epsilon$. Running the training sequence with these two codewords allows one to find a pair of centroids for the training sequence, giving a good codebook of rate 1. Repeating the process for these two codewords and so on will yield a codebook of the required size. Obviously this method is more suitable for codebooks of sizes that are multiple of two (2^R).

3.6.2.7 Problems of Standard Searching in VQ

During operation a standard VQ will take an input vector, find the best match for it among the codewords and select the one with minimum distortion according to the specified rule. This is done by calculating the distortion between the input vector and each of the codewords in the whole codebook,

A quick examination for the case of image processing applications reveals that this operation can be a quite expensive one. As a rough idea of the task involved, it is noted that codebooks are typically 64 to 1024 codewords long. Each input vector requires typically from 8 up to 64 square integer differences and integer sums to be compared with a single codeword. Moreover, those figures apply to a single block being vector quantized, and the number of blocks in an image is typically 4096 (a 512x512 image with 8x8 block size, giving 64 sums and differences for each comparison). The task of vector quantizing following the standard search method poses a large obstacle for any tentative real-time implementation.

Several alternative methods have been proposed to reduce the search effort required for each vector, with an eye on real-time applications, as well as ways of achieving better performance at a given codebook size. The following sections examine some of them.

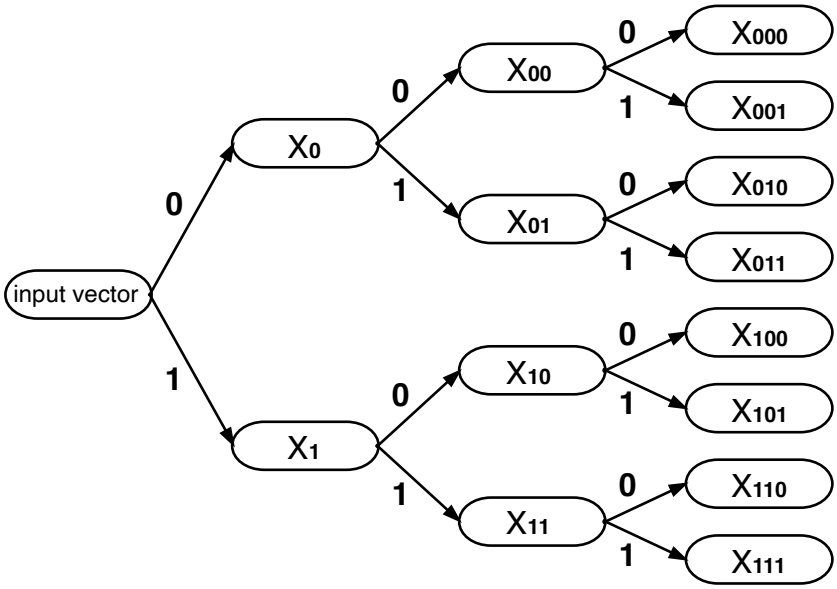
3.6.3 Variations of VQ Methods

3.6.3.1 Tree-Searched VQ

The splitting approach to generating the initial codebook gives a hint on how to extend the idea to coding. A binary tree will be considered here to simplify the analysis, but this is not a limiting factor. In fact, higher order trees tend to give better performance with a slight increase in processing load.

Suppose one builds an initial codebook for a vector quantizer by using the splitting technique described previously in section 3.6.2.6. Here, instead of simply incrementing the *rate* of the quantizer at every step (and doubling the number of codewords in the codebook), one also builds a tree keeping the old centroid values at each rate. This process is portrayed in Figure 3.16.

When operating, no longer has the vector quantizer to search the whole codebook for the best match for the input vector. Instead, the quantizer starts by



Example: encoder of rate 3

X... = centroids at each rate

0
1
= each chosen centroid adds its own index to the string which yields the final code at the end. Decision between the two centroids at each level is done according to a minimum distortion rule, such as the mean square error (MSE).

X_{ijk} = $i\,j\,k$ is the code (channel symbol), and $X_{i\,j\,k}$, the centroids at the last level, are the actual codewords available for reconstruction

Figure 3.16: Tree-searched VQ.

comparing the input vector with the two centroids or codewords (since the example assumes a binary tree) found as children of the root centroid. The root centroid is not represented in Figure 3.16. After deciding on one of them, the respective one-bit index is taken and added to the full index string being built. The process repeats for the next levels and each index bit is added in turn added to the string, which in the end will be the channel symbol transmitted [BGGM80].

An interesting point in the above algorithm is that by design each codebook available at each node is good for all input vectors mapped into that node. There are some points worth comparing between an ordinary full-search and a tree-search VQ:

- they both yield reproduction codebooks of the same size, i.e. 2^R, where R is the rate in bits per vector. Note that the intermediate centroids do not take part in the set of possible reproduction vectors.
- storage requirements are doubled due to the intermediate codebooks.
- the tree-search encoder is not optimal (the full-search one is), since not all reproduction vectors are checked for a best match, but it still gives performance close to the original full-search encoder.
- *(most important)* the tree-search VQ's efficiency can be orders of magnitude better than full-search VQ's, although with some performance penalty as regarding the quality of the reproduction.

Let us consider an example to present some comparison figures, bearing in mind the significant amount of computation involved in a single vector distortion measure (this, of course, depends on the size of the vectors). For the case of a rate 10 encoder (1024 codewords), a full-search encoder requires 1024 distortion measures for each input vector, whereas the tree-search encoder needs 20, a factor of 50 times faster.

Another trade-off can be set to achieve better performance at the expense of not so swift searches. This is done by using non-binary trees at the intermediate levels. For the example above one could use, for instance, a sequence of 3 - 3 - 4 bits per encoder, resulting in a tree of depth 3 (2 intermediate codebooks). This requires $2^3+2^3+2^4 = 32$ distortion measures, still well over an order of magnitude faster than the full-search encoder, but with very close performance.

Clearly, real-time requirements restrict practical VQ compression methods to tree-searched ones. The computational cost of full-search encoders are far from the original target requirements of this thesis.

3.6.3.2 Multistep VQ

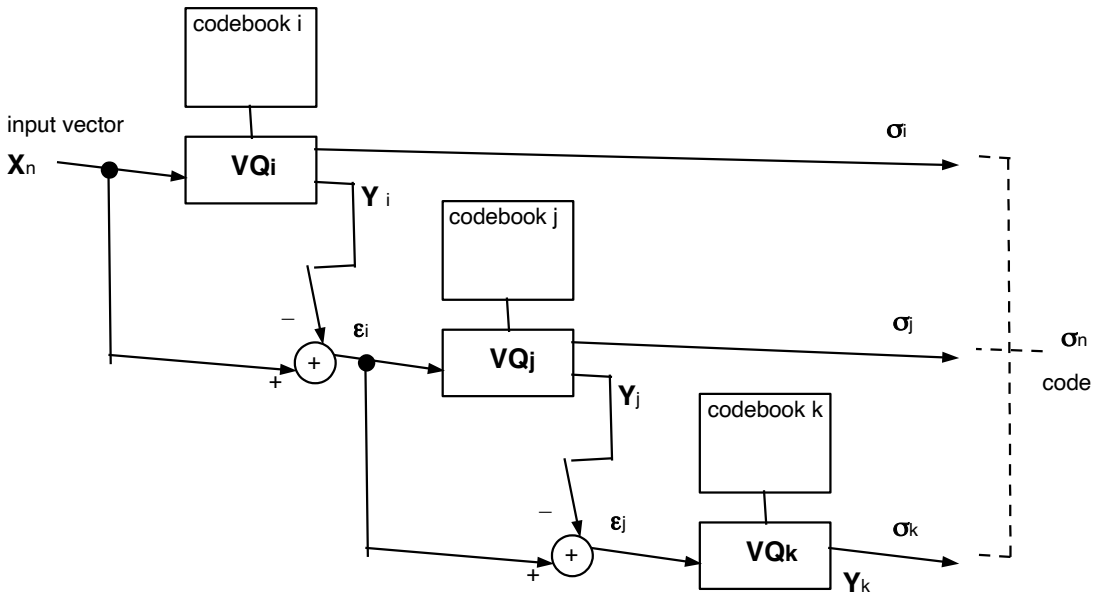
A variation of the tree-search encoder, intended towards reducing memory storage requirements, is called *Multistep VQ* [JG82]. It is depicted in Figure 3.17. Considering again the assumption of a binary structure, the first level of the coder is built as in the case of the tree-searched VQ. To build the second level, however, the procedure differs: the training sequence is coded with the first level coder (two codewords only to choose) and error signals are generated for each input vector. This error training sequence is then used to generate another codebook of two codewords, specifically to code the error signals. A new level codebook is generated by doing the same with the error signals generated from the previous error signal codebook, and so on.

The result is a single codebook at each level of the tree (two codewords only in the case of a binary tree), instead of a codebook at each node of each level of the tree, as in the tree-searched coder. To reconstruct an input vector a codeword is selected at the first level, then a codeword chosen from the second level is added to the first one and so on. The input vector signal is then rebuilt in stages. The number of different possible reconstructions is again 2^R. The search effort is about the same as in the tree-searched VQ, but storage requirements are halved: a rate R coder keeps $2R$ codewords in memory, whereas the tree-searched coder needs 2^{R+1} codewords.

Although the number of possible reconstructed vectors is still the same as the tree-searched VQ's, this coder is more limited and may not reproduce vectors as well as the tree-searched one. However, the multistage coder still boasts the basic features of a VQ, and also has significant advantages regarding storage space.

3.6.3.3 Mean/Shape or Separating Mean VQ

This coder first removes the sample mean of the input vector before coding it [BG83]. The separated mean is normally scalar quantized and the vector quantizer is entirely devoted to the shape to be described. This makes better use of the VQ ability to code vectors (shapes) rather than values. The term "mean" refers to the average of the brightness values in the vector.



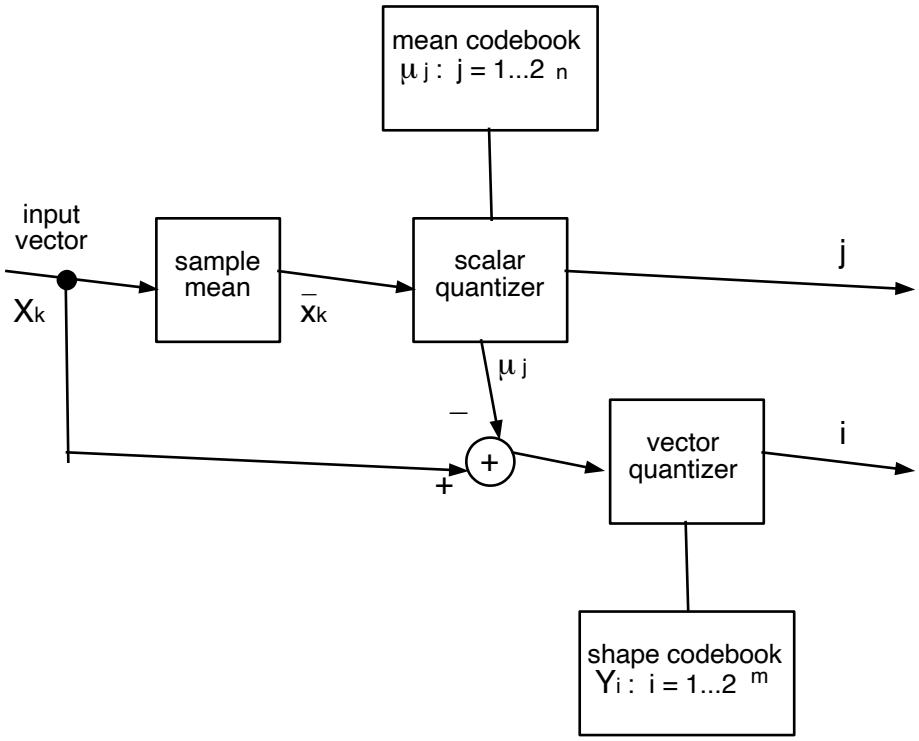
Example of a Multistage VQ with 3 stages: The first codebook i is designed to give a rough description of the block's shape. An error vector is generated subtracting the coded word from the input vector. This error vector is passed on to another encoder more adapted to small differences, whose output is then fed to a final encoder. The final channel symbol (σ) is assembled from the three individual ones. Reconstruction follows a similar path .

Figure 3.17: Multistage VQ.

The idea behind this scheme is the fact that a small rectangular block of an image has usually a strong low frequency brightness component. The average background brightness value is likely to vary little and slowly along the block. Since the same holds true for the corresponding blocks between two contiguous frames in a video sequence, this can also be applied to interframe coding.

The scalar quantization of the sample mean can also introduce error due to a possibly required coarse quantization. The Mean/Residual vector quantizer (M/RVQ) subtracts instead the quantized mean from the input vector [BG82]. The quantization error is then incorporated into the vector to be quantized, as seen in Figure 3.18.

One disadvantage of the Separating Mean VQ is that the coder is no longer optimal, i.e. it does not select the best possible reconstruction as a full-search coder does.



Mean/Residual (Separating Mean) VQ: the sample mean is more easily and better quantized using a scalar quantizer. By removing the large mean component from the input vector the vector quantizer yields a better shape description than by quantizing the original input vector.

Figure 3.18: Mean/Residual VQ.

3.6.3.4 Gain/Shape VQ

In this quantizer two separate codes are employed, as with the Separating Mean VQ. But instead of removing the sample mean, now a gain factor is extracted from the original input vector, in fact normalising it [BGGM80, SG84].

As an advantage over the Separating Mean VQ, the encoder is still optimal. The codebook of the product Gain/Shape, however, may not be optimal due to the restrictions imposed by the product form.

3.6.3.5 Classified VQ

If very low bit rates are being sought, several problems may show up at the reproduction image, such as edge distortion. One way of dealing with this problem is that presented by Gersho and Ramamurthi [GR82], called Classified VQ (CVQ). An input vector is first classified into several categories according to its features, such as having smooth shades, shades with moderate gradient, horizontal/vertical edges, diagonal edges and so on.

The above classes may be further subdivided to correct any weakness noted in the quality at reproduction. Each class has its own separate codebook tailored to it, therefore being more able to reproduce particular features of the image. The search requirements are also reduced by restricting the search to the appropriate codebooks only instead of looking up all of them.

The authors reported a great improvement in image quality, particularly by increasing the size of the edge codebooks. Although search requirements are reduced as compared to a single large codebook, the total computational requirements may be quite high. This is due mainly to the gradient edge operators needed to classify the edges, but also to housekeeping routines needed to manage the overall complexity.

3.6.3.6 Feedback and Adaptive VQ

Several alternative schemes make use of memory in the quantization process, for which they are generally called *feedback* or *adaptive quantizers* according to their basic *modus operandi*. The term 'memory' here refers to the use of past input vectors to choose a codebook among a collection, to best suit the local input characteristics. For such scheme to work the decoder has to keep track of which codebook is being used at the moment, and this is accomplished in two ways:

- the codebook selection procedure followed by the encoder may depend only on past encoder outputs, so that the decoder can also follow the encoder steps (*Feedback Vector Quantization*).

- The decoder is informed by the encoder of which codebook is selected via a low-rate side channel in the communications link (*Adaptive Vector Quantization*).

The continuous change of codebooks as seen in the encoding of an image, can also be viewed as a finite state machine (FSVQ), where each state stands for a particular quantizer and its associated codebook. A state transition function defines how the next state of the machine is found.

Many variations have been presented on the basic theme in the recent past, but the development of VQ which relies on past information is still at an early stage. What follows is a list of several proposals on this subject, noticing that it is hard to classify each one into separate categories. Each approach makes use of features found in other schemes.

- Hang and Woods [HW85] presented a VQ where each input vector consisted of the current pixel, plus previous inputs as the remaining elements of the vector. This is used as a sort of mapping to predict the next sample. A look up table is used to predict the next codeword and an error signal generated.

- Murakami *et al.* [MAIY84] used vector quantization to code the interframe difference signal of areas detected as moving regions.

- traditional techniques of motion compensation can also be incorporated into basic VQ.

- Some methods work by code replenishment: in the coder presented by Goldberg and Sun [GS86] small codebooks are built and transmitted to match local characteristics of the image to be coded. There is however a certain redundancy among the different codebooks of the complete image. A better idea [GY85] makes use of adaptivity by updating a subset of a single, larger codebook. During coding, distortion is measured and if found above a specified threshold, the input vector is added to the codebook.

- The possibility of having vectors represent variable size blocks is tackled with a hierarchical VQ [Nas85]. A quadtree keeps the hierarchical structure,

which is transmitted via a side channel. Vaisey and Gersho [VG87] use a standard split-merging segmentation technique to partition the image.

3.6.3.7 Composite VQ Schemes

Besides using vector quantization as a compression scheme on its own, VQ is often combined with some other method to achieve a higher compression rate.

VQ can be used with transform coding by coding the coefficient matrix resulting from the transformation, or even by coding the frame difference signal [ML86, AK86, NK83]. Other hybrid methods usually found in image coding can also be applied with vector quantization, such as adaptive transform [HR85, STA+86] and Block Truncation Coding (BTC) [UR87].

3.6.4 Performance of VQ

VQ and transform coding work by treating small blocks of image as separate entities to be coded. VQ gives quite good compression rates, but in general better results are obtained with transform coding, particularly the cosine transform.

Memoryless VQ seems to be usable in the range of 0.5–1.5 bits/pixel. As a reference for the figures that can be obtained, the Separating Mean VQ gave acceptable image quality at 0.5 bits/pixel [Gra84]. Details were as follows: the mean sample was DPCM coded and a 6-bit codebook was used with 4x3 subblocks of image. It is clear from the example in the paper that the overall distortion is quite high, with a significant "blockiness" effect. Larger codebooks should certainly improve quality, but Adaptive VQ presents more significant differences, at the expense of a more complex implementation.

3.7 Conclusions

This chapter presented a survey of available image compression and coding techniques, evaluating their advantages and weak points under the assumptions stated in the beginning of this thesis. Rough figures were given for compression rates achievable, together with comments on resulting image quality and computational power required.

Chapter 4

Experiments with Real World Images

4.1 Introduction

This chapter presents first a description of the equipment used to do the image compression experiments for most of this research, and some relevant details about the image acquisition process are discussed.

Next, results of first experiments with real world image acquisition are presented, together with a discussion of some basic points regarding image compression.

Then, frame freezing (dropping frames from the sequence) is investigated for possible use in a composite compression scheme. The use of frame freezing often results in image quality impairment. An experiment with evaluation by subjects was necessary to assess the acceptability of frame freezing, and the conclusions are reported.

Finally, this chapter contains a closer examination of hierarchical coding, in particular quadtree coding. Further investigation is dedicated to quadtree coding to assess its potential for the compression scheme sought.

4.2 Description of the Image Processing Equipment

The equipment used throughout this research was developed by the Wolfson Unit in the Medical Biophysics Department at the University of Manchester. It is an evolution of an earlier design employing a 16-bit computer as host, which was rather slow and limited for serious image processing work. This earlier equipment was used for the experiments described in Section 4.7.3, where its details are presented.

The new equipment is a module that plugs into the VME bus of a SUN-3 workstation. The SUN-3 provides the usual assortment of tools for software development as well as a clean 32-bit machine, with regard to its memory architecture. This new system does not present the memory and other system resource limitations imposed by the earlier system, making image processing tasks a lot easier to implement. The image processing module contains video acquisition and display circuits coupled to a frame buffer and a bit-slice processor with an address processor to carry out common image processing tasks more effectively than a general purpose CPU. The video input accepts standard PAL TV cameras supplying an interlaced video signal, and outputs to a standard PAL monitor, also interlaced. The frame store can be fitted with a variable amount of memory, usually 4 MBytes, which is enough for storing 16 frames at 512x512 pixels or 64 frames at 256x256 pixels. Data can be transferred from the SUN host to the frame buffer and back again by using special library calls and a dedicated UNIX kernel.

Although the library is comprehensive for normal image processing tasks, it has very little use in the course of this research, since there are no tools for image compression studies. Only the frame store, used as a storage device for captured frames, and its ability to acquire and display frames in real time (40 ms/frame) are actually of interest here. After acquisition images are transferred to the host computer (SUN 3/160) to be processed by an application program. Transfer rates from the frame buffer to the host and back are rather slow (about 60 KBytes/sec). This precludes the ability to remove, in real time, acquired frames from the frame buffer so that more can be acquired in order to digitise long image sequences. However, the system is of use for short image sequences.

The analog video signal coming from the camera goes to an A/D converter to

be digitised and stored in the frame buffer. The circuit provides real-time acquisition as well as real-time display, which allows visual testing of processed image sequences. The design employs Vidicon cameras which give moderate Signal-to-Noise ratios (commonly referred to as SNR), as will be seen from practical measures in a later section (4.1). The image noise from this combined camera/acquisition system can be seen by visual inspection to change pixel brightness up to the fourth least significant bit[1] (a change of $\pm$ 8 levels out of 256). Probably for this reason the A/D converters used in the design have only 6 bits resolution, since the two least significant bits would have been concealed by noise anyway. This somewhat limits the extent of part of the results obtained in this research.

4.3 Real World Image Acquisition and Analysis

This section discusses a few topics that are often taken for granted in video teleconferencing and image sequence compression in general, but that may present rather curious conclusions for the designer of systems in these fields.

4.3.1 Grey-Level Contouring

In Section 3.3.1.2 it was seen that the number of grey levels considered necessary for non-objectionable contouring is between 32 and 64 (i.e. 5-6 bits/pixel). However, video equipment does not allow as simple a solution because there are some extra variables that affect observers' judgement of the abovementioned threshold.

Observations performed by the author suggested that reducing the number of quantization levels may produce some unexpected results. For most real world images, decreasing the number of bits per pixel from 8 down to 1 shows a relatively sharp fall in image quality at 3-2 bits/pixel. This point can vary according to the image characteristics (contrast and brightness), as well as with the corresponding settings of the monitor used. Some comments about this feature follow below (see also Figure 4.1:

- at 1 bit/pixel (2 grey levels) image quality is bad in the sense of the amount of image information, but the effect produced ("posterisation[2]") is very common as a special effect in printed material (Figure 4.1d. This makes the

[1] all values are stored as if having an 8-bit scale.

[2] in the printing field the term is applied for reduction to other numbers of grey levels as well.

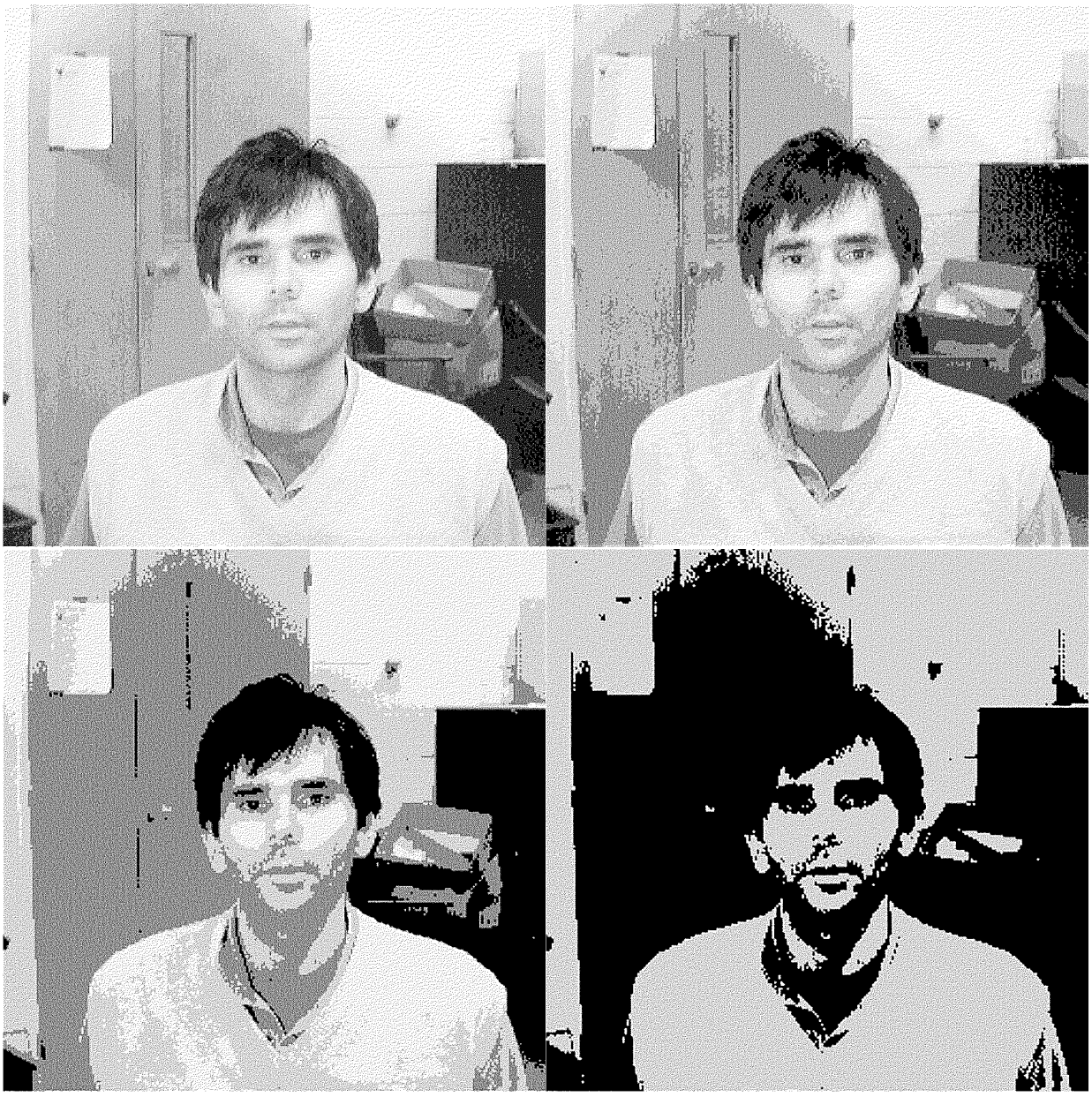


Figure 4.1: Contouring effects caused by coarse quantization at: a) 4 bits/pixel (top left); b) 3 bits/pixel (top right); c) 2 bits/pixel (bottom left); d) 1 bit/pixel (bottom right).

appearance of the image look less unexpected.

- at 2 bits/pixel (4 grey levels) a lot of detail is added to the image, so it no longer resembles a posterised image. Neither does the image appear normal, since there is insufficient brightness detail. The availability of several brightness levels allows the appearance of bands of similar brightness on the image. The shapes created by these bands usually do not correspond to the normal shapes expected out of the scene by an observer, as can be seen in Figure 4.1c.

- at 3 bits/pixel (8 grey levels), the above description may also occur, depending on the image characteristics. The example of Figure 4.1b does not present this effect, partly because of the limitations of the image rendition on paper.

- at 4 bits/pixel (16 grey levels) the banding effect is still noticeable but the main shapes of the image dominate the overall appearance (Figure 4.1a). The image now looks normal, but with some visible distortion.

It was also found that 16 grey levels gives a reasonable picture quality for critical situations, i.e. when transmission bandwidth gets tight. Contouring is visible but it still allows the observer to perceive the picture contents with no unexpected effects. 32 grey levels was considered acceptable according to more stringent requirements: usually very little contouring is visible. At 64 grey levels, the maximum the equipment could render, no noticeable contouring could be detected. The effect of contrast settings is discussed in Section 4.3.3.

4.3.2 The Presence of Noise

The camera used in these experiments was a Bosch camera fitted with a Vidicon-type sensor[3]. It was found that real world images captured with this setup presented considerable random noise (salt and pepper like) which is at first taken to be harmful to image quality. Closer examination revealed that in fact the majority of the images benefitted from the presence of this noise. The images in Figure 4.1, particularly 'b', show that the bandings are severely disrupted, contributing for a less marked appearance of the contouring effect.

[3]the complete specification is camera Bosch, type TYK 9A.

The video capturing equipment was tested to measure the magnitude of the noise embedded in the acquisition process. The measurements were done supplying an even brightness background with diffused light. The camera-digitiser set limits image acquisition to 64 grey levels, but throughout this thesis the acquired values are immediately converted to the 0-255 levels range and processed that way. Noise was measured at different base brightness levels (16, 128 and 240) and spatial positions in the field of vision of the camera-lens combination. Measurements were done over a neighbourhood of 64 pixels at the chosen spots. Overall there was little variation on the measured values regarding both spatial position and luminance levels. These values can be summarised as follows:

$$\begin{aligned} \text{Range of variances:} \quad & 5.7 - 6.4 \\ \text{Range of standard deviations:} \quad & 2.39 - 2.53 \end{aligned} \tag{4.1}$$

For the sake of simplicity the value to be considered for the standard deviation thereafter is 2.5. This knowledge will be employed now and then in future sections.

When compared to the brightness resolution of the acquisition process (a 6-bit digitiser giving a resolution of 4 levels out of 256), it becomes clear that the camera's inherent noise will influence the acquired brightness values. With a standard deviation of 2.5, a particular noise sample has a good likelihood of crossing the threshold of the 6-bit digitiser's least significant bit. On the border of two bands some pixels from one brightness level will assume the level of the other band, and vice-versa.

Figure 4.2 shows the image of Figure 4.1 after being filtered using the Sigma filter, which is discussed in Section 4.4.5. Since this filter is quite efficient in removing low-level noise, the result is that band edges no longer benefit from the disrupting effect of noise. These edges then become quite apparent due to the edge enhancing characteristic of the human visual system. To assess the difference between the original image and the noise-free image, compare Figure 4.1a with Figure 4.2c, both at 4 bits/pixel. Or yet, compare Figure 4.1b and Figure 4.2d, both at 3 bits/pixel.

The design of a video teleconferencing or similar system therefore must take into account the existence of noise either generated by the camera itself or by the cabling necessary for the connections, which may degrade the signal as distance

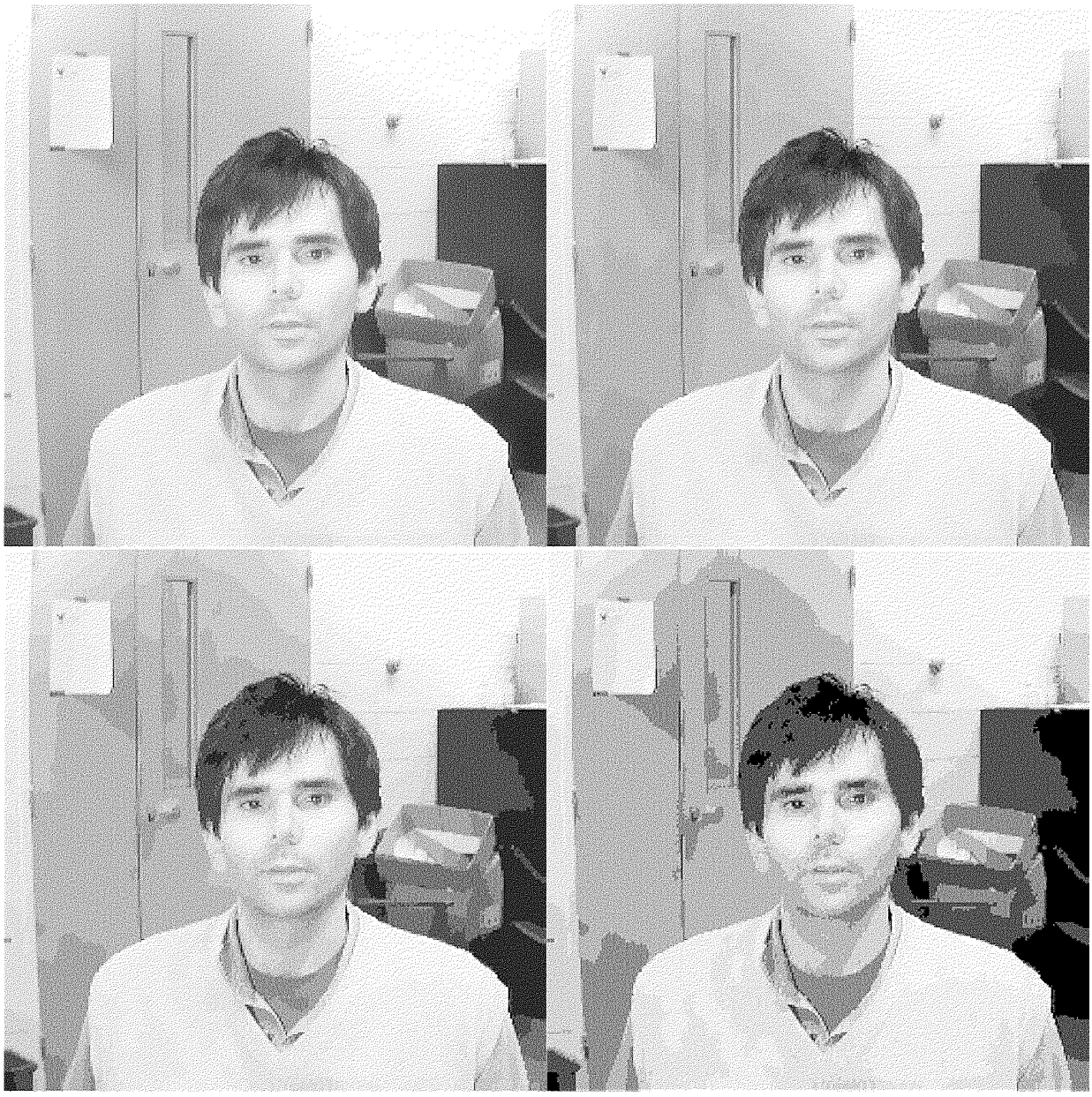


Figure 4.2: An image processed by the Sigma filter with a threshold of 8: a) at 8 bits/pixel (top left); b) 5 bits/pixel (top right); c) 4 bits/pixel (bottom left); d) 3 bits/pixel (bottom right).

grows. The availability of cameras with better Signal-to-Noise Ratio (SNR) figures will certainly benefit image compression systems, in terms of compression efficiency, but the end visual quality may not present considerable changes. Noise filtering is discussed in Section 4.4.

4.3.2.1 Noise Measure Definition

This is an opportune moment to introduce the definition of Signal-to-Noise Ratio (SNR), as used throughout this thesis. This measurement criterion is used to report the magnitude of noise and, more importantly, as a distortion measure.

SNR will be used as one of the means of comparing processed images with their originals. Other criteria must be taken into account though, since this measurement alone does not always translate the image quality perceived by the human visual system. Other alternative criteria exist, each one claimed to better detect a certain range of distortion types. SNR is however the most commonly used measure, and so was adopted here.

The convention followed in this thesis is as follows:

$$SNR = 10 \log_{10} \frac{(255)^2}{e_{ms}^2}$$

where e_{ms}^2 is the mean-square error, calculated over all pixels. For example, a uniformly distributed noise with values 1, 0, -1, 0 (out of the 256 possible values in a typical image) will give a Signal-to-Noise Ratio (SNR) of 51.1 dB.

4.3.3 Influence of Contrast and Brightness Preferences

The visibility of false contouring is greatly diminished by the presence of noise in the acquisition process, as has been seen in Section 4.3.2. Other parameters that affect the overall visibility of contouring are the brightness and contrast settings of the monitor where the image is being displayed. These settings depend a great deal on personal preferences, but other things contribute to leave these parameters open to circumstantial variations.

Awareness of this unpredictability is important because certain limits of acceptability in image quality may depend on these parameters. For example, it is common to find in the literature the suggestion of images quantized with 5-6 bits/pixel as being of reasonable quality. Depending on the settings of the monitor, a user may consider this suggestion as too good or not good enough, and so

these suggestions must be taken cautiously.

A user may want to change the contrast and brightness settings of the monitor, depending on the factors related below:

- the analog dynamic range of the camera. The user normally establishes a focus of attention on a certain region of the picture. If the contrast of the this image region is dull, the user may consider increasing the contrast setting to get a better view. As a consequence both the contouring effect and the noise produced in the acquisition process are enhanced. If, on one hand, the image is quantized with fewer bits/pixel, contouring may become obtrusive. If, on the other hand, the noise amplitude is high enough to overcome the contouring effects, the noise itself may become objectionable[4].

- environmental lighting conditions can also affect the choice of contrast and brightness settings. This further contributes to widen the range of variations affecting the image quality.

An important conclusion can be drawn from the above explanation and from Section 4.3.1 onwards, affecting the choices made for critical situations of image quality. If the design of the final compression algorithm is to cover most of the conditions described above, the limits must be pushed towards better quality. In the case of quantization levels, 6 bits/pixel must be considered the minimum acceptable for most situations. However, if the final compression scheme turns out to require noise prefiltering, a minimum of 7 bits/pixel is recommended. Such a number of grey levels makes the banding effect virtually invisible, even with high contrast settings. Fewer bits per pixel should only be considered as temporary measures, when bandwidth restrictions in the communications channel force drastic measures.

[4]However, workstation monitors usually have more limited ranges for the contrast and brightness controls than general purpose monitors. This may be due to the need of preserving good image quality when black and white text is being displayed. The internal functioning of monitors is outside the scope of this thesis.

4.4 Noise Filtering

4.4.1 Introduction

Although the presence of low-level noise may actually be beneficial for visual quality (see Section 4.3.2 above), the efficiency of all coding techniques suffers from it. Noise prefiltering (smoothing) is therefore an important technique to be used in image compression systems, subject to constraints of image quality and processing load. The following subsections discuss several filtering techniques and their applicability to this thesis. Conclusions regarding recommended action are presented in Section 4.4.6.

4.4.2 Basic Filtering Techniques

One filtering technique commonly used in single image processing (multiple frame acquisition of a still picture) is time-oriented frame averaging: since noise manifests itself not only from neighbouring pixels but also from the same pixel taken from two consecutive frames, several frames are acquired and averaged in a pixel by pixel basis. Statistical theory guarantees that noise will be reduced with the square root of the number of frames averaged. Time-oriented frame averaging is very effective for still noisy images and takes on a more important role for very low SNR pictures (image reconstruction from poor originals). For image sequence coding, though, the more frames that are averaged, the more motion blur is exacerbated, causing severe degradation of image sharpness. Overall the cost becomes too high, in exchange for very little improvement in the noise figures.

Among spatially oriented filters there are mainly two categories, recursive and non-recursive ones, also known as IIR and FIR filters, respectively. Recursive filters are, for their own nature, very processing-intensive, which mostly rules them out for real-time utilization. It must be recalled that noise prefiltering is supposed to be an auxiliary technique in the complete compression scheme, and not the most demanding one.

Among non-recursive filters some have been examined to compare their processing loads and picture quality effects. The more immediate options are linear filters, starting with a simple rectangular function and going up to more elaborate shapes like the Gaussian. The former calculates the average pixel value over the area overlapped by the rectangle, and the latter behaves as an averaging measure

that emphasises the more central pixel values in the calculation. These filters are applied by convolving the image with a kernel describing the particular filter function[5].

4.4.3 Non-Linear Filters

The result of applying the linear filters mentioned above to a real world image is usually disappointing. On one hand, the underlying noise is not completely removed and still upsets the coding scheme that lies ahead, as part of a complete compression scheme. On the other hand, as the filter area is increased to remove more noise, details of the image start to be smeared out. By the time the filter complexity (and the size of the convolution kernel) manages to clean the noise down to more reasonable levels, most observers will miss the crispness of the original image.

The solution giving a better compromise lies in using non-linear filters, which can be tailored to the particular noise levels most likely to occur with the given video source. Adaptivity to local image conditions also helps in attaining better overall performance.

The first example of a non-linear filter is the ubiquitous median filter. Its action is not to average the values in the region as happens with the mean (normal average) filter. Rather, all values in the region are sorted by their amplitudes and the value that lies in the middle of the sequence is chosen as the filtered value. This does not sound like a very logical way of doing filtering, but in fact its effect is particularly good regarding edge preservation. The median filter is particularly useful to remove high amplitude, spiky noise, preserving the crispness of the original image remarkably well [CY83]. The processing load is one of the lowest among filters, making it an attractive option for noise prefiltering. However, it is not particularly effective with the application this thesis is dealing with: it is not suited to smooth low-level noise.

The median filter is quite limited by its simplicity and cannot cope well with features that are smaller than its operational window, and several other filters have appeared to address this issue. Most of them show the same basic strategy, which is to be more selective with the values to take when actually doing the filtering calculations. They also add what is indispensable for removing low-level noise, i.e. averaging. Here are some examples:

[5]another application of the convolution process can be seen in Section 3.3.7.

K-nearest neighbour average: this filter picks up only K pixels from the population defined by the filter window. The pixels selected are the ones whose brightness values are closest to that of the centre point of the given window. A smaller K will lead to less noise attenuation but better detail preserving characteristics [DR78].

FMH – FIR–Median Hybrid: this basically employs median filters, subdividing the original filter window in smaller regions. The filter is then able to preserve smaller features than a normal median filter of same window size is, while still doing a reasonably good noise removal. Another problem with large window median filters is the processing required to sort the values so that the median value can be chosen; the FMH filter decreases this load considerably with the subdivision strategy [NHN87].

Sigma filter: this follows the basic strategy of the K-nearest filter, but is not restricted by a fixed number of values over which to average. Rather, it picks up the values that are within a 2-sigma range of the centre point value. Sigma here refers to the standard deviation of the noise, so that this range restricts the collection to values of brightness similar to the centre point. Nearly 70% of the noise occurrences fall in this range. Of course, this range can be changed to suit particular image characteristics.

4.4.4 Performance of Some Non-Linear Filters

Chin and Yeh [CY83] compared several filters with respect to noise removing and detail-preserving abilities, and the K-nearest filter came top for all round results. The Sigma filter performs better than the K-nearest, as can be seen in [Lee83]. For the analysis that follows, it should be borne in mind that the interest for this project is on images with low amplitude noise (high SNR figures).

In the first paper above it is seen that the K-nearest filter only performs reasonably well in areas of low spatial activity. This actually refers to the basic ability of noise cleaning, which is the end purpose being sought. Since the K-nearest filter does not stand out among the other filters in this aspect, the reader might then ask why it ranked on top overall. Well, the other filters in the comparison perform better in noise removal, but they do so at the expense of poor edge-preserving results. The K-nearest filter does the best job in this key area, and this translates very well into visual image quality.

It is also easy to see why it does not perform well for low spatial activity areas (low-level noise smoothing), and why the simple averaging technique performed best in that test. The simple averaging filter benefits from having the total number of pixels to evaluate the average. In contrast, the K-nearest filter is, by design, limited to only K pixel values out of the total number of pixels.

4.4.5 Sigma Filter

The Sigma filter overcomes the particular weakness of the K-nearest filter, discussed in the previous paragraph, by not limiting the averaging process to a fixed number of pixels. The limitation now applies to whether or not the pixel brightness values are within the predefined threshold range. When applied to high spatial activity this change does not come into play and so does not alter the performance of the filter in relation to the K-nearest filter (it might be slightly better in some cases though). However, when applied to low spatial activity areas it almost generalises to the simple averaging filter, which is recognised as the best filter for this particular case. For example, if all pixel values are within the predefined threshold range, they will all take part in the averaging process. In fact, the Sigma filter can perform a little bit better than the simple average filter in some cases, since it excludes occasional "out of the curve" pixel values (spike noise). These values are blindly swallowed by the simple averaging filter and therefore contribute to an undesired, slight change in the resultant average brightness.

The processing load of the Sigma filter is roughly comparable to the simple averaging filter, since the basic arithmetic operations to be performed are the same. The basic task is to sum a collection of 8-bit integer values and to divide the result by a small integer (averaging). Some extra processing is introduced by the logic that finds out which pixel values to add to the sum, and the proper incrementing of the divisor. The K-nearest filter has less values to sum, but it also has to sort the list of brightness values from the filter window. Other filters looked at by Chin and Yeh are generally more computationally expensive than the above two filters. The median filter might be the least expensive one, by a slight margin. Figure 4.3 shows a sample picture processed with a normal average filter and a Sigma filter, together with some statistics about their smoothness.

Figure 4.3: Comparison of the average filter and the sigma filter: a) original (top left); b) average filter, 3x3 window (top right); c) average filter, 7x7 window (bottom left); d) sigma filter, 7x7 window, threshold = 8 (bottom right).

4.4.6 Conclusions

It has been found that the Sigma filter provides good filtering characteristics at reasonable computational cost. Other more complex filters, particularly recursive ones, are able to give better results, but at a much greater expense in processing power. Even with a filter as simple as the Sigma filter, noise removal requires a dedicated processing unit to achieve its goal in real time.

From Section 2.2.4.1 it is known that the data supply alone needs approximately 6.6 MBytes/s (512×512 image). The Sigma filter with a 3x3 window processes 9 pixel values for each of those pixels arriving at 6.6 MBytes/s, resulting in a few tens of integer operations for every pixel. This translates to something in the range of 12-24 MOPS (Mega Operations Per Second).

It becomes clear then that if noise prefiltering is judged necessary, a pipeline arrangement is the only way to insert such feature in the whole compression scheme. Features required from the architecture would be, apart from real-time capability, the availability of memory space to hold at least a full frame for filtering purposes. Also important is memory bandwidth to negotiate the high volume of data flow from video acquisition, through prefiltering, to the actual compression units.

4.5 Experiments with Frame-Freezing

Frame rate reduction by frame freezing is the technique of skipping frames from the original image sequence and repeating previous frames to take the place of the missing ones. This way the screen is still refreshed at the standard frame rate.

As pointed out in Section 3.2.2, experimental data about its acceptability for viewers is very rare in the literature. In order to consider its employment in the compression scheme being sought some hard facts were needed on which to base the research.

An experiment was then devised and set up, counting with the help of some volunteers. The goal of this experiment was to find out whether image sequences subject to frame freezing would still be taken as normal sequences, by the observers. Another goal was to try and quantify any differences the observers might notice. It is understood that scoring is intrinsically a subjective issue, and that the magnitude of any differences in scores are only meaningful to some extent;

however, these differences might be able to give a rough quantitative idea of the relative quality of several frame freezing patterns. This eventually led me to decide on a parametric test.

4.5.1 Equipment Setup

4.5.1.1 Recording

It was decided to employ U-Matic recorders to avoid the recording quality interfering with the judgement of the image sequences. These recorders give a vertical resolution close to the 512 lines that have been assumed throughout this research. Overall these recorders boast better image quality than home video recorders. A Video Editing Table greatly eased the repetitive tasks of rewinding tapes to appropriate position and synchronizing time bases. This allowed achieving clean image source switching, as well as control of the sound track.

The Time Base Corrector is a device whose primary purpose in video editing is to correct small time differences that arise between two different video sources. These differences can normally make video editing a messy job. The unit that was used had the ability to freeze either a frame or a field in its memory, and this was how frame freezing was achieved. Normally it is controlled manually from a front panel switch, but there was also a TTL interface at the back, very convenient for computer control.

The TTL signal was converted by an external circuit to RS 232 levels, and then fed to a status line of a serial interface of an Atari ST microcomputer. Since the Atari also generates a standard PAL video signal, it was found that the computer's time base precision matched very well that of the U-Matic recorders. No external frame synchronization was felt necessary. Any phase errors that might occur would rarely cause a field to be slipped and this event would not be noticeable.

The Atari's internal time base was again used to count frames for the different freezing strategies to be tested. A system call was used to generate pseudorandom numbers, conveniently modified to look like a Poisson distribution.

4.5.1.2 Playback

A large screen TV set was used to show the sequences to the judges, who were seated at a comfortable viewing distance. Proper care was taken to avoid disturbing reflections on the screen, since a normal working environment was assumed for the test. No other activity was going on in the office at the time of the tests. Sound volume was set to a comfortable level, and the sound track was as in the original sequence. This research assumes that sound is being taken for granted, its delivery quality being guaranteed by some other system.

4.5.2 Details of the Experiment

4.5.2.1 Description of the Experiment

The experiment devised employed two basic image sequences, each about 30 seconds long. The first one was basically a head-and-shoulders scene and the second one presented a little more movement, since the camera pans during a few seconds to change the focus of attention.

Two strategies were pursued for doing frame freezing, and applied to both basic sequences: one in which the frames are frozen at a fixed rate (*FF*) and one in which the freezing pattern is made random by following a Poisson distribution (*PF*). The original sequence is indicated by *ORIG*. It was felt necessary to help judges in marking, by establishing parameters on which to judge other sequences. It must be emphasised that all modified sequences lasted for the same time as the original one, and showed the same short take.

In the notation used from now on, a number following the strategy name means the period (in number of frames) each frame of the modified sequence is displayed on reproduction. In other words, the time each frame lasts on screen. For example, FF1 means fixed freezing at 1 frame period, which corresponds to no freezing at all (original sequence). All frames from the original sequence are shown, each lasting one frame period. FF2 means fixed freezing at 2 frame periods, which is the minimal fixed freezing pattern. Only every other frame from the original sequence is shown, each lasting two frame periods.

In the case of the random pattern (Poisson distribution), the number appended means the average display duration of each frame. For example, PF1.7 means that in average a frame lasts on screen for 1.7 frame periods. It should be noted that the Poisson distribution is actually calculated as if the average

duration were 0.7. The whole distribution is then shifted by one (to 1.7) as the smallest possible freezing value is 1 (no freezing at all).

For the sake of statistical methods each modified sequence is called a *treatment*. The treatments chosen for the test were ORIG, FF2, FF3, FF4, PF1.5, PF1.7, PF2.0 and PF2.5.

4.5.2.2 Experimental Procedure

For each of the two basic sequences a test set was assembled consisting of one instance of each treatment. It was decided that the original sequence (treatment ORIG) should always be shown first, and that the other ones should follow it randomly sorted. The random order, on the one hand, avoided any bias that I might have shown through a particular ordering choice. On the other hand, the arrangement made things harder for the judges when making up their minds about the marks. This was the reason to decide on displaying the original sequence first. The solution helped them to more easily work out a marking structure in mind, as confirmed in a post-interview.

It was suggested to participants that they were actually making use of such visual facilities to communicate with people in a working environment. Participants were then required to give scores to each of the short sequences shown, according to how acceptable they found the sequences. The range of scores allowed was from 0 to 10 in integer steps, with 10 supposed to be the highest mark. After each short sequence (about 30 seconds long) a short pause was given to allow the participant to make up his/her mind and write the mark down.

Each basic sequence generated a set of 8 sequences (including the original). Before actually starting the practical sessions it was decided to allow another display for re-assessment marks. After the first pass over the set of sequences, the process was repeated and the participant was required to give another set of marks. Thus, a possible change of mind about the marking of a particular sequence could take place, based on the previous markings (the previous markings were not allowed to be modified). The total number of judges was 11.

4.5.3 Results

An ANOVA (Analysis of Variance) was performed to detect whether the different treatments caused significant differences to appear in the participants' marks

[SC80]. Following this, a comparison of means was made to find out whether these differences were significant or not [AR72]. Notice in the tables to follow that the class "treatments" includes the original sequence.

4.5.3.1 Analysis of Variances

Table 4.1 shows an analysis of variances table (ANOVA) aimed at identifying whether the frame-freezing processed sequences (treatments) were really different, one from another, to the eyes of the judges.

The tables indicate the F-test number found and the thresholds for significant and highly significant differences (5% and 1%, respectively), for easy comparison. An arrow pointing to the left of the 5% column means that there were no significant differences; pointing between the 5% and the 1% column means a 5% level significant difference; finally, pointing to the right hand side of the 1% column means a 1% level highly significant difference.

Since significant differences were found between the treatments a comparison of means was allowed, for which a Duncan's new multiple range test was chosen, as described in the next section.

4.5.3.2 Comparison of Means (Duncan's New Multiple Range Test)

Table 4.2 compares the average markings given by the judges to each sequence treatment. For an explanation of the notation used, refer back to Section 4.5.2.1.

The bars under the figures represent several distances whose values were found to be important for this research. Each distance must be checked against the checker table on top, where **p** stands for the number of values covered by the particular distance chosen, and **Rp** stands for the threshold difference. The **Rp** values are commonly accepted as significance levels, under statistical theory. Both 5% (significant) and 1% (highly significant) levels are shown.

4.5.4 Comments

Findings from the experiment will be discussed in the next sections, together with useful conclusions for carrying on with the project. Attention is called to some rather interesting results concerning the difference between fixed and variable-rate frame-freezing.

ANOVA - first round

source of variation	df	sum of squares	mean square	F	significance threshold (5%)	(1%)
sequences	1	13.64 13.64	9.72	5.32	11.26	
treatments	7	358.81	51.26	36.51	3.50	6.19
judges	160	310.36	1.94	1.38	2.97	4.93
error 8	11.23 1.40					
total 176	694.04					

results

judges: no significant differences in their scores
sequences: there are significant differences (5 % level)
treatments: there are highly significant differences (1 % level)

ANOVA - re-assessment

source of variation	df	sum of squares	mean square	F	significance threshold (5%)	(1%)
sequences	1	1.64 1.64	2.27	5.32	11.26	
treatments	7	482.36	68.91	95.58	3.50	6.19
judges	160	277.45	1.73	2.40	2.97	4.93
error 8	5.77 0.72					
total 176	767.22					

results

judges: no significant differences in their scores
sequences: no significant differences (5 % level)
treatments: highly significant differences (1 % level)

Table 4.1: ANOVA for the frame-freezing experiment.

```
shortest significant ranges - first round
p     2     3     4     5
Rp (5%)     1.16  1.21  1.24  1.26
Rp (1%)     1.70  1.76  1.81  1.84

comparison of means - first round
treatment   ORIG  FF2   PS1.5 PS1.7 FF3   PS2.0 PS2.5 FF4
mean  9.04  6.95  6.09  5.73  5.64  5.23  4.36  4.36


shortest significant ranges - re-assessment
p     2     3     4     5
Rp (5%)     0.83  0.87  0.89  0.90
Rp (1%)     1.22  1.26  1.30  1.32

comparison of means - re-assessment
treatment   ORIG  FF2   PS1.5 PS1.7 FF3   PS2.0 PS2.5 FF4
mean  9.18  7.18  6.36  5.55  5.41  4.77  4.00  3.86
```

Table 4.2: Comparison of Means (Duncan) for the frame-freezing experiment.

The decision to do a re-assessment session for each judge turned out to be a good measure. The intent was that the participants could reposition the marks among the treatments in a more consistent way, after having learned what the best and the worst performers looked like. More details in Section 4.5.4.2.

4.5.4.1 Performance of Judges

By looking at the *judges* row in the ANOVA tables (first round and re-assessment), Table 4.1, we find that the judges were quite uniform in their opinions: the F-test reveals that there are no significant differences (5% level) among their scores. The re-assessment session did not change much.

Some of the subjects were not Computer Science related ($\approx$ 30%), and the remaining ones had different interests in the field of Computer Science. The test was considered to cover a representative sample from the population of possible users of such video facilities. Since no significant differences were found among the judges (see the arrows at the 'judges' line, both first round and re-assessment), they seemed pretty much in agreement about their opinions.

Overall they reported little difficulty in making their decisions, under 20% of them saying it was an issue.

4.5.4.2 Sequences

The *F*-test on variations caused by the sequences showed, on the first round, that there were significant differences between the two basic sequences (line 'sequences', first round, Table 4.1). In fact, one of them was entirely a head-and-shoulders sequence, whereas the other contained also a zooming/panning take. This could account for a possibly worse result when freezing frames, but not by much. The entirely head-and-shoulders sequence set was always shown before the corresponding set for the scene with zooming/panning. It is worth remembering that at this point of the test the subjects did not have an idea of what the range of 'goodness' and 'badness' in image quality would be.

It can be seen that the introduction of the re-assessment session was worth the effort. There were no longer significant differences between the scores of the two basic sequences, which was what I really was expecting to happen (line 'sequences', re-assessment, Table 4.1).

4.5.4.3 Treatments

In both cases (first round and re-assessment) I obtained highly significant differences, with the re-assessment session only reinforcing this fact. Since differences exist, Duncan's new multiple range test could be applied to compare the treatment means.

4.5.4.4 Comparison of Means

The first thing to notice is that no one treatment could be considered in place of the original sequence, i.e. all treatments caused recognizable decrease in the quality of the image sequences. Before preparing the tests and actually seeing how they looked like, I was hoping that at least the less damaging treatment could be accepted in place of the original sequence. Soon after the first tests I had changed my mind, and the analysis of variances confirmed this.

Among the treatments there was no significant difference between two consecutive treatments when their scores were arranged in order of magnitude. This led me to conclude that the range of parameters chosen was right, meaning that a full range of quality was covered.

Table 4.2 was examined to check whether there was a treatment that could be taken for the original sequence (i.e. no significant differences in their average

scores). FF2 had the next best score in the test, apart from ORIG itself, and so was chosen for the comparison. The figures to look at are those indicated by the bar underneath **ORIG** and **FF2**.

The difference in the first round is 2.09, which is larger than the value under Rp (1%) for p = 2 (number of means in the interval being studied), 1.70. This latter figure is looked up in the shortest significant ranges table on top of the averaged means. Therefore, this difference is considered a highly significant one. In the re-assessment the difference falls to 2.00 but the threshold for highly significant differences falls even more to 1.22. This reaffirms that no one treatment gives an image quality as acceptable as the original one. The fall in these values agree with the assumption that judges would give more consistent marks in a re-assessment session.

FF2 and **PS1.5** look indistinguishable in the first round but in the re-assessment the difference is just about significant at the 5% level. Strictly speaking this difference is not significant, since the value of 0.82 is just below the threshold of 0.83, but the exact definition of the 5% level is conventional only, and not a rule. It is then assumed that FF2 produces better quality sequences than PS1.5.

FF2 and **FF3**: judges already put them apart at the 5% level in the first round with a difference of 1.31 against a threshold of 1.24. In re-assessment the difference went up to 1.77 against a threshold of 1.30 at the 1% level.

Summarising the points seen so far, not only was none of the treatments a match for the original sequence, but also i) FF3 was recognisably worse than FF2, and ii) PS1.7 and FF3 were roughly similar, since their differences were only 0.09 and 0.14 (first round and re-assessment, respectively).

Other comparisons showing similar results (5% level in the first round and 1% level in the re-assessment) were: i) FF4 was found worse than FF3; ii) PS2.0 was found worse than FF2; and iii) PS 2.5 was found worse than PS1.5. Also, FF4 and PS2.5 were found similar.

4.5.4.5 Fixed Versus Poisson-Distributed Freezing Patterns

By looking at the means in Table 4.2, re-assessment, one is struck by the difference in quality indicated by the scores obtained by the two strategies. By taking treatment names with similar numbers appended one is comparing similar savings in data volume: equivalent amount of frames are skipped.

Notice the difference between FF2, 7.18, and PS2.0 with only 4.77 . The most

likely explanation for this behaviour is as follows. First let us recall that PS2.0 means a Poisson distribution of mean 1.0 to whose variates an offset of 1.0 is added. Most of the variates will certainly be 0, 1 and 2, with larger values only occasionally showing up. But, in a time scope of several seconds it is highly likely to find these larger values appearing fairly regularly, say, a few seconds apart. Variates generated by the pseudorandom process are 'consumed' quite quickly in a per second-basis in these video sequences (averaging from 10 to 16 variates per second).

For a person judging a sequence it does not matter too much that most of the frames ran smoothly. Rather, a few frames frozen for too long, once in a while, causes a very bad impression. These impressions do last for a long time, whereas impressions from the smoother parts are just not taken into account. The reason for this is that when an image sequence runs smoothly it does just what they are supposed to, as far as observers expect from standard TV sequences.

Therefore, the most important conclusion from this experiment is that if a variable freezing pattern is unavoidable, great care should be taken to not allow any frame to be frozen for too long. Also, if field freezing is being used instead of a full frame, some of the jerkiness can be attenuated. This can be achieved by conveniently splitting the freezing period between the two fields.

4.5.4.6 Judges Impressions on Lip Tracking and Distress

Not all subjects commented on lip tracking, i.e. the matching of sound and mouth movement. Some 36% said it was an important issue, but for some others it did not matter. Usually the best treatments were considered acceptable.

What surprised me was that the distress caused by overall jerkiness was much more disturbing than lip tracking, for all subjects. Previously I had considered both aspects to be equally important.

4.5.5 Conclusions

For 36% of the subjects none of the treated sequences could be tolerated for long periods of time, with the best ones acceptable for short periods; for another 36% the best ones were acceptable for long periods of use. Some subjects also mentioned the case of sequences with a fair amount of movement (from the sequence with zooming/panning) as being more distressing than scenes without

much spatial activity.

In summary, it is concluded that frame freezing does cause severe harm to the acceptability of the image sequences, therefore suggesting keeping as close to the normal refresh rate as possible. Also, if frame freezing at a variable rate is unavoidable, the experiment suggests quite strongly to prevent individual freezing periods to last too long. Long freezing periods could be avoided, for instance, by triggering an emergency procedure if the delay reaches a specified threshold. The system would then rush partial updating information, to allow reconstruction of a new frame, even though one of not good quality.

Whatever the freezing pattern to be used, further thought should be given to try and attenuate jerkiness. For example, a suitable technique might be some sort of filtering that blurs the main features of the scene.

4.6 Hierarchical Approach to Image Compression

Since the beginning of this project an interest has been developed on hierarchical representation and coding. A hierarchical representation is desirable for its ability to deal with partial information content. For example, part of the information conveyed by a structure describing an image can be used to get a less detailed view of the whole contents of the image.

In the case of image representation 'less detail' may be translated in different possible ways. Less detail could mean, for example, fewer high frequency components (a low-pass filtered image), less brightness information per pixel (a more coarsely quantized image) or lower resolution (less number of pixels per unit area of image).

The latter approach above is particularly interesting since the main goal of this project is to reduce the amount of data shifted to transmit image information. Halving the image resolution cuts the amount of data by four. Of course, the resultant image quality is worse than the original, but such recourse could be useful for temporary situations. Low transmission bandwidth situations occur quite often in communications channels. This way, in those critical moments the image quality could suffer, but under low traffic conditions the system could still transmit high-resolution images.

In this framework the one particular hierarchical representation that stands

out is the quadtree. A brief description has already been seen earlier in Chapter 3.3.5 and a more detailed examination follows in the next sections.

4.6.1 Quadtree Encoding

The quadtree is a hierarchical data structure that has become important in computer graphics, image processing and related fields [Sam84]. It is based on recursive decomposition, being able to focus on interesting subsets of the data and, as other hierarchical data structures do, presents an inherent clarity and ease of implementation.

The decomposition of space into smaller areas has many applications and can be applied to regions, curves, surfaces and volumes, but in this case they are useful to represent data, or more precisely colour. The decomposition may be into equal parts (regular polygons) on each level, or even into different parts, according to some input data. I shall however be dealing with regular decomposition employing squares. It comes as no surprise due to the shape of the working area, i.e. the square left from cutting the edges off the original rectangular TV image.

The decomposition goes on as many times as the image resolution determines. The resolution can vary according to some input data or be fixed beforehand. For this project the main concern is with fixed resolution situations, though it would be possible to have this changed under special situations. For example, when the communications channel is under severe demand the image resolution could be temporarily changed. The transmission process could analyse the image at one or two levels above the nominal resolution, therefore generating much less data into the communications channel. Undoubtedly, the result would be a poorer quality picture, but the sequence could kept with the same degree of liveliness (same frame rate).

4.6.2 Representation Example

Let us see how this decomposition works on the region represented by the contour on Figure 3.5 (top left), Page 66. The complete image array is shown in Figure 3.5 (top middle) with pixels marked as ‘1’ representing the inside of the region and ‘0’ its outside. In Figure 3.5 (top right) the array is subdivided into four equal-sized quadrants (squares). If a quadrant does not consist entirely of 1’s or entirely of 0’s it is further subdivided into subquadrants until it does so. As previously said,

how far the subdivision can go depends on the image resolution, or even on the minimum cell size (which may be different from one).

A tree of degree 4 (each nonleaf node having four sons) is then built (Figure 3.5, bottom), each child node representing one of the quadrants labelled NW, NE, SW and SE according to its position. Leaf nodes, or terminal nodes, are the ones that do not need further subdivision.

A leaf node is said to be *black/white* if its represented area is entirely *inside/outside* the target region (1's/0's in Figure 3.5, top middle). All nonleaf nodes are said to be *grey*.

4.6.3 Grey-Scale Image Representation

A *binary* image has only black or white elements. If shades of grey are being represented the image is said to be a *grey-scale* image. This project is primarily concerned with grey-scale images because a binary image cannot represent a real world image with the necessary quality. This is not to be confused with the use of bit patterns simulating grey levels. Dithering techniques allow a grey-scale image to be conveyed on a binary screen, though severely compromising in image quality. Anyway, the image representation still has to carry grey level information.

To represent grey level information, grey level values are associated to each leaf node. Alternatively, it is also possible to distribute the image colour information as several bit planes; the grey-scale image array is then seen as a stack of bit planes, each one a binary image. Corresponding quadtrees are then built for each binary image [KEM83].

4.6.4 Standard Quadtrees

Although quadtrees present a highly structured encoding method that is useful for searching and set operations, their data structures are very storage-consuming. Each non-terminal node has to make room for pointers to four children (usually 32 bits each) and each leaf needs room for the colour code (8 bits typically). A gross estimate for the total number of nonterminal nodes is about the same as the number of leaves. The analysis by Lauzon [LMKG85] show that by trimming pointers to the strictly needed length the average record length stays at more than 50 bits. For other encoding schemes this number drops below 30 and can go as low as a few bits, besides storing only leaf nodes. This makes a quadtree in

its standard representation highly unattractive for the purposes of this project.

4.6.5 Linear Quadtrees

In the light of the work by Gargantini [Gar82] several alternative schemes have been proposed that are based on the elimination of pointers from the encoded quadtree. The linear quadtree proposed by Gargantini gives representation codes that already have embedded the path from the root to the node. This is done by assigning the values 0, 1, 2 and 3 to the quadrants NW, NE, SW and SE, respectively.

To code a leaf (only leaf nodes are coded), the tree is descended from the root down to the specified leaf, coding each node according to the rule in the previous paragraph. The left most digit corresponds to the highest level nodes (first subdivision of the image array). For example, leaf X in Figure 3.5 is coded as 212. The encoding of the complete quadtree is done by listing the leaf nodes from left to right. Due to the fact that some leaves are of higher levels than the lowest possible, a fifth value is added to the set to describe this situation. 23102X would mean the code for a leaf of order 1 (a 4x4 square) in a tree six levels deep. The original proposal is concerned with binary pictures and stores only black pixels, but the concept can easily be extended to a grey-scale image by associating a colour value and storing all leaves.

4.6.6 Proposed Encoding Scheme

The encoding system proposed for the experiment in Section 4.7 works by building a kind of linear quadtree for two consecutive frames in an image sequence. A third, partial quadtree is built by working out the leaves that changed between the two previous quadtrees. The difference list can then be used to reconstruct the second frame at a remote station, given the previous frame.

There are several encoding schemes with good compression ratios, but the problem of choosing a given scheme is always one of trade-off between space efficiency and time efficiency (i.e. computational power needed). In this case it was felt that perhaps time efficiency might be more important than space efficiency; also, the completeness of an explicit quadtree (one that describes all the leaves) could contribute to speed up the comparison of both trees to build a difference list.

y ↓ \ x →	0	1	2	3	4	5	6	7	8
0	0	1	4	5	16	17	20	21	64
1	2	3	6	7	18	19	22	23	66
2	8	9	12	13	24	25	28	29	72
3	10	11	14	15	26	27	30	31	74
4	32	33	36	37	48	49	52	53	96
5	34	35	38	39	50	51	54	55	98
6	40	41	44	45	56	57	60	61	104
7	42	43	46	47	58	59	62	63	106
8	128	129	132	133	144	145	148	149	192

Figure 4.4: Morton sequence numbering for the first 64 cells in a digital image.

The suggested encoding scheme is derived from Gargantini's, but the value assigned to the positional code can only be chosen from 0 to 3, as suggested by Lauzon. The level information is left out of this part of the code. It is of great advantage if the individual bits within a number can be readily accessed, as when programming in the C language: conversions between Morton numbers and row and column co-ordinates are really straightforward. Certain high level languages though must use modulo and division operations to accomplish the conversion, as happened with this research (Pascal was the language used)[6]. The sequence of values for the complete codes forms the Morton numbers, as can be seen in Figure 4.6.6.

Incidentally, there is an added bonus for using such encoding: if the quadtree is to be kept in its full form (i.e. not compressed), it will be represented as a list of records sorted by the code, which is quite good for making fast comparisons.

[6]to speed up processing, the conversion was later done by look-up tables.

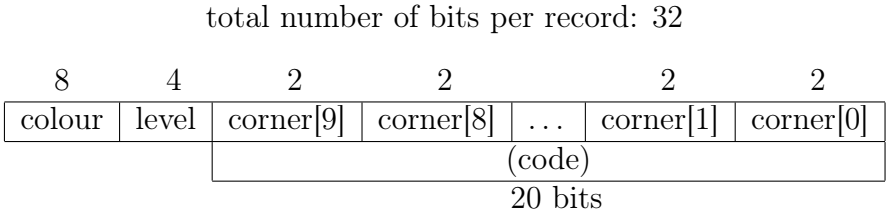


Figure 4.5: Proposed record format.

The format adopted for the encoding is shown in Figure 4.5. Each instance of the array `corner[]` can assume one of the codes 0–3. *Colour* carries the grey level value and *level* the level of the leaf.

This format has provisions for a more generic usage, coping with quadtrees of up to 10 levels deep (1024 x 1024 pixels) and up to 256 colours or shades of grey. It was intentionally devised to match a full 32-bit word, allowing efficient memory utilisation by fast processors.

An immediate improvement for the coding of the whole quadtree is to omit from the list leaves that repeat the grey level value of a previous leaf. Since the list is sorted by the code, it is quite easy to work out the missing leaves to reconstruct the image.

4.7 Experiments with Quadtree Encoding

4.7.1 Overview of a Quadtree-Based Encoding System

4.7.1.1 Video Acquisition

Figure 4.6 shows the configuration of such system. The video signal comes from the camera to an A/D converter, capable of sustaining about 20 MHz @ 8 bits of precision to match the incoming signal at the desired resolution, according to Section 2.2.4.1 The video acquisition hardware should provide in theory enough room for 4 Mbytes of data. Each quadtree in its explicit form occupies 1 Mbyte in the worst case, and there are three of them to be maintained: two frames of a sequence and the difference list; there are also the original images, each taking 256 Kbytes. To reduce memory requirements only one of the frames could be kept in quadtree form, with the second one to be built on top of the first one. The difference list would also be built in the process and immediately sent to the link. This way memory requirements would drop to 2 Mbytes.

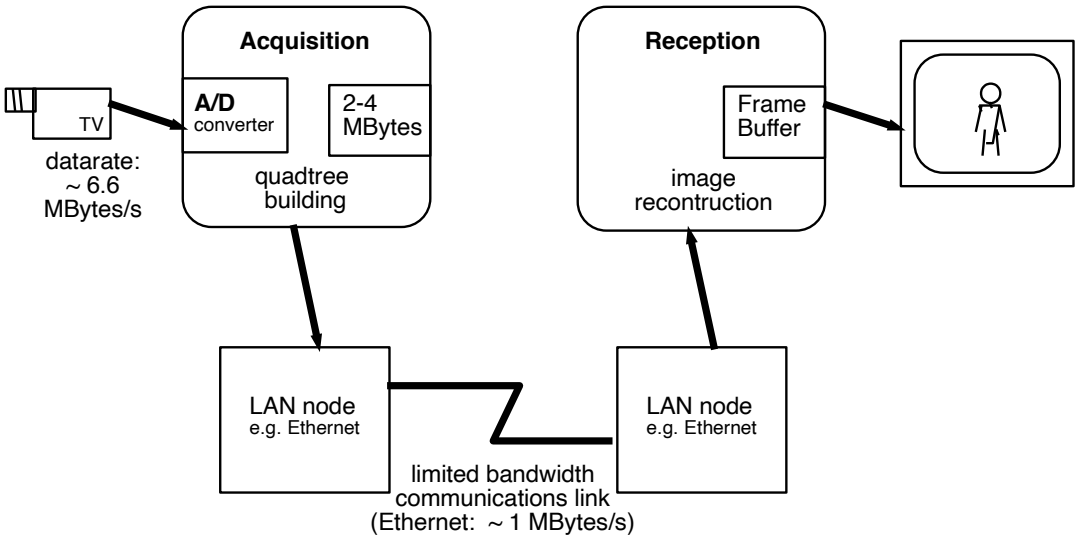


Figure 4.6: Configuration of a quadtree-based encoding system.

4.7.1.2 The Transmission Protocol

Next step in the chain is to send the difference list over the communications channel to the destination station. A protocol has to be established to handle the traffic of difference lists, in order to keep valid operations along the session. The sending rate might be dependent on the complexity of the images (i.e. a measure of the percentage area of the images represented with low level leaves) and of their difference list. The protocol will have to assure that the frame for which a new difference list is being built is the same as the one the destination station is assumed to have.

4.7.1.3 The Receiving End

Upon reception of a difference list, the receiving station must reconstruct the frame following the one it already has in the frame buffer. It does so by updating the leaves that changed, taken from the difference list.

The implementation details of the destination station determines whether the reception will contribute as a bottleneck to the overall performance. It is strongly desirable to handle the live image as a window on the station's screen, just like any other window (resizing, moving, hiding etc). The desktop metaphor of a windowed system can then be maintained. As a consequence, an implementation

making use of direct video memory access, and not integrated with the window manager, is not of interest here.

One can see two alternative implementations for the receiving end: i) the image is reconstructed from a newly arrived difference list on an intermediate buffer, followed by a repaint operation by the window manager (i.e. the transfer of the image from a temporary buffer to the frame buffer, masked by the current window lay-out); ii) the decoding processor could take over the video controller based on information sent by the window manager, diverting the video controller's output and inserting its own. This is similar to the way a video editing device called *genlock* works.

The latter alternative above would be easier to implement than having the video controller switch between source buffers "on-the-fly". In the case of switching frame buffers, the screen tiling information must be supplied by the window manager. The Intel 82720 chip [int85] does this sort of management, but to exploit the high bandwidth features of video RAMs (VRAM) those memory regions are required to be in the same physical memory module. This poses extra problems for an implementation due the synchronous nature of VRAMs.

4.7.2 Quadtree Statistics

4.7.2.1 Available Data

There is very little literature, if any at all, on statistics of real world scenes based on quadtree decomposition. The only references containing useful data that I managed to locate were Kawaguchi *et al.* [KEM83] and Lauzon *et al.* [LMKG85]. However, the sort of images to which the data refer are of specialised subjects and do not match the sort of images this thesis is dealing with.

Kawaguchi's paper shows a woman's face digitised at a resolution of 256 x 256, with 4 bits per pixel. The problem here is the way chosen to represent the image: it is actually a forest of binary quadtrees, each one being responsible for one bit-plane. One can get some clues about an equivalent grey-level quadtree by looking at the number obtained for the least significant bit-plane: the paper says that there are 45 % as many primitives as pixels.

Lauzon *et al.* show some statistics for a set of images comparing different encoding techniques. Table 4.7.2.1 cites a few of them that are about 512 x 512 images with 4 bits/pixel. However, there is little indication that those images

Image	Leaves	% of maximum	Runs	Leaves per run	Pixels per leaf
Geographic data, slope	15,301	5.84	7,912	1.93	17.13
Geographic data, drainage	21,415	8.17	12,842	1.67	12.24
"Baboon" (mandrill face)	46,789	17.85	27,709	1.69	5.60
Topographic band map	73,102	27.89	36,967	1.98	3.59
Analytical hill shading	137,587	52.49	83,162	1.65	1.91

Table 4.3: Quadtree statistics of a few images [LMKG85].

and the ones of interest for this project bear any similarity, in terms of embedded noise.

The numbers from Table 4.7.2.1 are to be considered in the following context: the proposed encoding model uses 32 bits to encode a leaf, meaning that in the worst case four times as much data is needed to encode a picture (this worst case refers to having the number of leaves equal to the number of pixels). Instead of being compressed, the image data is expanded, which is not useful at all. The break-even point happens when the number of leaves is 25 % of the number of pixels.

Experiments were then planned to check out initial expectations that i) a typical image does not present many more leaves than this break-even point, and that ii) in a typical image sequence the number of different leaves between two consecutive frames is a fraction of the number of leaves of these frames. The latter hypothesis would mean that a difference list would contain substantially less data than the original image array.

The data from Table 4.7.2.1 seem to go along with the above assumptions, with just one of the pictures presenting too many leaves. However, only one of those images could be considered to resemble a real world scene (the "baboon"), if at all. To make things worse, all the images reported are only 4-bits deep. Since this thesis is working with a wider range of grey shades (256 grey levels, 8 bits/pixel), these data may have no usefulness. The figure from Kawaguchi's paper, ≈ 45 % of primitives/leaves, does reveal some worries about the feasibility of the whole system.

4.7.3 Quadtree Experiments

Since no data available seemed to properly cover the type of images with which this project was concerned, it was then decided to perform a set of experiments on a proper set of images before going any further.

4.7.3.1 Implementing an Image Sequence Analysis System

To implement an analysis system I had to rely on equipment outside the department, consisting of a camera–frame buffer–minicomputer combination, able to store up to 4 frames of 512 x 512 images, with up to 6 bits/pixel. Unfortunately this equipment imposed severe limitations on my original intentions:

- reasonable software library undermined by the availability of only Pascal as a high level language. C would be much more efficient thanks to bit-field addressing and readily accessible unions.
- limited system memory of 64 Kwords. This meant that frame data storage should rely entirely upon the four subsets of the frame buffer.
- 16-bit microcoded engine lacking enough processing power and ability to handle 32-bit integers. Much of the complexity of the program derives from the latter point.
- no significant mass storage available and incompatible floppy disc format. Data could not be acquired and transferred to a suitable environment for later processing.
- maximum acquisition rate of one frame every 140 ms, which is three and a half frames of standard TV. I was therefore unable to study the actual process required, i.e. a stream of contiguous frames of a sequence.

4.7.3.2 Performance of the Analysis Program

The program described below was the first step towards a larger program written to provide a suitable environment and tools for image sequence analysis.

Besides its original running host, the program was also running on a VAX 8600 to allow better support (editing and debugging facilities), simulating the other system with fake data and not taking advantages of the 32-bit architecture. The

execution times obtained were actually very bad for an actual implementation: 13-16 minutes on the imaging equipment and 40-50 seconds on the VAX 8600, to compute quadtrees for two frames and the difference list.

These figures are clearly nonsense for a process that is supposed to run about as fast as the time of a few frames of standard TV, say 40-120 ms. However, it should be pointed out that the overhead in simulating the weak points of the external system were incredibly high. It is reckoned that an implementation in C could mean an execution time one or two orders of magnitude faster. Also, a real implementation does not need to build two quadtrees to get one difference list (the next difference lists in a sequence require building one quadtree only).

4.7.4 Results and Comments

4.7.4.1 First Impressions

The first surprising fact was that dealing with real world equipment presented characteristics that could change the directions of the whole project. The knowledge acquired from the first experiments made clear that this 6 bits/pixel equipment had a rather high noise level in the acquisition process. Preliminary estimates showed noise oscillations of up to ±2 grey levels out of 64 [7]. This number does not sound very high, but quadtree images turned out incredibly fragmented due to this noise.

When a smooth area of the image is being encoded one would expect that a good proportion of the leaves will fall into higher level categories, resulting in a good compression rate. Unfortunately it was observed that higher level leaves rarely occur due to oscillations in pixel brightness corresponding to noise. Examining acquired data revealed also that many sequences of similar brightness value occur, but do not end up being merged into a higher level leaf. Either the length of the sequences was too short (2 or 3 leaves), or the a sequence of 4 or more leaves did not match the leaf boundaries in the quadtree structures. Just anticipating the results shown in the tables that follow, most pictures were returning 90-95 % of the maximum number of leaves, instead of the 20-40 % that was hoped for. Se also Section 4.7.4.2 for detailed comments on data obtained.

The high number of leaves is not in itself a problem, as long as the process is stable and does not change along time. What was found, though, was that

[7] refer to Chart 4.1 on page 114.

the high number of leaves generated by noise actually helped to generate further shades of grey, in much the same way as dithering is used to simulate shades of grey on binary output devices. Also found was the fact that the noise was not present only in the spatial sense, but also in the time sense: the patterns on screen changed from frame to frame. As a result the quadtree difference lists were always much longer than expected, making a system based on this technique very unlikely to give useful compression results.

4.7.4.2 Table of Statistics

Table 4.4 presents a summary of several scenes upon which the comments are based. All sequences were shot at the maximum fire rate of 3.5 times the standard frame period (or 7 times the field period, 140 ms). The table lists: i) a number describing the scene, a pair of numbers for the frames chosen from the sequence and the number of bits/pixel taken into account; ii) the number of first level leaves (i.e. of size 1 pixel) for both frames being compared; iii) the total number of leaves for both frames and for the resulting difference list; and finally, iv) the number of leaves for the quadtrees of both frames, but using a more compact representation that omits all leaves that repeat the same brightness value, as described at the end of Section 4.6.6.

4.7.4.3 Comments

As previously said, the data obtained did not quite match the original expectations. To start understanding what is happening let us look at scene 2. Not bothering with the actual fragmentation of the picture, a small difference list should have been obtained, as two shots of a motionless scene are supposed to be very similar. Instead, over half the leaves showed different brightness values, either by comparing frames 0 and 1 or frames 0 and 2. This also confirms the random nature of the noise.

Scene 3 is a floppy-disc box against a plain background, that should present much better figures due to the evenness of the picture. Again, figures were not as good as expected: around 80 % of fragmentation and a difference list of around 45 % of the maximum number of leaves.

Scene 4 had approximately 40 % of its area slightly moving as a result of the face of a person talking, yet it did not give worse results than scene 2. Scene 1, however, could be considered to have a more complex background.

Data given as percentages of the maximum number of leaves
All images 512 x 512 acquired at 6 bits/pixel

scene-frames	bits/pixel	first level leaves 1st fr	first level leaves 2nd fr	all leaves 1st fr	all leaves 2nd fr	all leaves diff	compressed list 1st fr	compressed list 2nd fr
1 – (0-1)	6	94.40	93.76	95.63	95.15	61.68	72.96	72.59
1 – (1-2)	6	93.76	94.20	95.15	95.48	61.69	72.59	73.00
1 – (2-3)	6	94.20	94.23	95.48	95.50	61.07	73.00	73.05
2 – (0-1)	6	88.57	88.41	91.41	91.28	52.00	62.86	62.73
2 – (0-2)	6	88.57	88.57	91.41	91.41	52.19	62.86	63.02
3 – (2-3)	6	77.58	77.66	79.89	79.96	44.52	57.14	57.43
4 – (0-1)	6	90.35	90.60	91.81	91.98	60.13	71.08	71.39
5 – (0-1)	6	90.01	90.01	91.20	91.20	59.38	73.07	73.18
6 – (0-1)	4	56.06	56.32	64.80	65.01	24.52	35.87	35.84
6 – (0-1)	5	83.84	84.10	87.61	87.81	45.01	58.18	58.23
7 – (0-1)	4	56.53	56.92	63.49	63.91	20.15	39.19	39.32
7 – (0-1)	3	36.65	36.97	43.40	43.70	10.84	25.44	25.62
8 – (0-1)	4	50.15	50.71	58.75	59.25	19.16	32.07	32.01
8 – (0-1)	3	26.61	26.97	32.66	33.06	9.68	17.15	17.15
9 – (0-1)	3	30.45	31.13	37.75	38.45	12.55	19.13	19.30

scene key:

1 lab scene, some moving regions

2 motionless

3 floppy disc box (lit by bulb lamps)

4 lab scene, face slightly moving (40 % area)

5 messy background, motionless

6 smooth background, face moving (50 % area)

7 messy background, motionless

8 smooth background, motionless

9 smooth background, face moving (50 % area)

Table 4.4: Summary of quadtree statistics.

With scenes 5 to 9 it was attempted to assess how the fragmentation and noise stood up when masking off the least significant bits of the brightness values. Results were not as good as expected: even scene 8, a motionless picture with only 8 grey shades, was still quite fragmented, and the relatively small difference list (under 10 %) was unacceptably long for a still sequence. By accounting for the number of higher level leaves, one can find that the difference list represents about 24 % of the whole area of the picture.

Let us compare scenes 6 and 8 with 4 bits/pixel: one is motionless, the other had $\approx$ 50 % in area with movement, yet their difference lists were not much farther apart: 19.16 % and 24.52 %. The effect of movement that was intended to be studied was completely overshadowed by noise.

4.7.5 Conclusions

Here is a summary of conclusions taken from the data obtained:

- camera noise is present in real world scenes (as opposed to computer generated ones) and therefore its presence should always be considered when devising compression schemes.

- the available equipment was perhaps not representative of the average modern equipment, particularly in regard to the camera, which could be less noisier than those used. It must be recalled that today's standards are 8 bits/pixel (and not 6) and state-of-the-art A/D converters go easily well beyond 25 MHz @ 12 bits/pixel, as opposed to 15 MHz @ 6 bits/pixel. The latter figures are the specification of the A/D converter fitted to the equipment). What remains to be checked is how immune to noise are the new CCD devices, common nowadays.

- if quadtree coding is to be used, noise prefiltering must be performed to eliminate a major cause of fragmentation.

Further experiments were done with images filtered using the sigma filter described in Section 4.4.5. There was a general improvement of 30-40% on the figures reported above, but this improvement was not large enough (i.e. 60-80%) to warrant further exploration.

It was hoped that, with noise removed, the stillness of most of the image would cause the large quadtrees from two consecutive frames to be mostly identical. This

implies at first examination that only the primary quadtrees would be too large. The difference list built from them would in theory be much shorter. It turned out that even with the filtered images the difference list did not shrink enough.

The conclusion was that normal image fragmentation alone generates too many leaves. The reason for this behaviour was that even after filtering the edge boundaries (contouring effect resulting from noise removal) were still sensitive to noise. Grey level boundaries were still described mostly by first level leaves, and these boundary descriptions constituted most of the quadtrees length. The more important conclusion was that the described encoding scheme would not be able to attain the compression figures hoped for, unless as part of a composite scheme employing more powerful techniques.

However, it is felt that the method is inherently strong in the hierarchical way of dealing with the image. This could be an advantage when devising such a composite scheme.

Nevertheless, the results obtained from the experiments with quadtree encoding were important for the rest of the research described in this thesis. The hierarchical way of treating images influenced one way or another several stages of the research reported in the next chapters. The quadtree encoding method was also used to some extent as a tool to analyse image fragmentation.

4.8 Conclusions

This chapter presented the equipment where most of the research was developed. First experiments with real world images were reported, and the most important fact that evidenced from them was the impact of noise. The presence of noise strongly affects the compression rates a given encoding method may present, as well as overall image quality. Surprisingly enough, the presence of noise normally contributes to a better looking image, with a more real look to it.

The experiments with frame freezing revealed that this compression technique cannot be used very frequently, since its side effects cause a great deal of annoyance to observers. It can only be used a a short temporary measure, if severe motion breakup is not desired. Frame freezing can, however, be put to good use if temporal interpolation is used to suppress most of the motion breakup effects.

Quadtree encoding turned out to present very poor results, as far as compression rates are concerned. The hierarchical structure can provide approximations

of what the complete image looks like, and this feature influenced the research described in the next chapters in search of a better compression scheme. In particular, the approximation of the image at higher levels of the tree was then exploited in the time dimension. The next chapter describes a basic compression scheme that spreads image updating information along time, using the early description (actually subsampling) provided by a quadtree view of the image.

Chapter 5

Basic Compression Method

5.1 Introduction

This chapter reports on an experimental compression scheme called thereafter BCS (*B*asic *C*ompression *S*cheme). It makes use of movement detection features to select blocks of image for further coding. It also updates blocks of image found with movement using a progressive transmission approach, spreading the updating information along 4 frame periods.

This scheme was inspired by the knowledge gained with image acquisition and quadtree processing presented in previous chapters, as well as by other work studied in the literature. BCS was not developed as a complete compression scheme, rather as a starting point upon which more elaborate techniques could be built.

Results of the first implementation are reported and weak points are discussed. Two issues were found to need improvement so that image quality remains sufficiently good: i) movement detection strategy; ii) profile of the updating strategy, i.e. the distribution of updating image information along the updating process period.

The former issue above is studied further and alternative solutions are presented. Compression rates and final image quality (supposing no other compression scheme is used apart from movement detection) are then compared for the alternative detectors. The latter issue (updating image strategy) is covered in the next chapter.

5.2 A Basic Compression Scheme

5.2.1 Compression Strategy

The conclusions drawn in Section 4.7.5 discouraged the view of quadtree encoding as a suitable compression method, unless it was part of a composite scheme.

That view was based on a scheme that was not tolerant to any changes in the image, that is, every brightness change of any pixel could cause an entry in the list of different quadtree leaves. Real-world scenes shot with a Vidicon camera presented so much noise that the majority of the difference leaves referred only to brightness changes that always happened from frame to frame, carrying no image information at all.

The main source of redundancy in an image sequence is, however, the frame-to-frame similarity, which could not be properly exploited with the quadtree encoding. Much redundancy was passing through and being coded in the difference list, since any amount of brightness change was enough to trigger an entry in that list.

One of the main assumptions driving this project is that the reconstructed image sequence does not need to be exactly the original sequence. This assumption immediately raises the notion of tolerance threshold levels.

To properly exploit the frame-to-frame redundancy it was then decided to completely separate the process of finding out which parts of the image needed updating from the encoding method to be employed on the data from these image regions.

Compression results from the two processes: i) a movement detection procedure is employed to determine which parts of the image are considered to have changed, based on given threshold levels, in most cases reducing the image data to a fraction of the original; ii) the image data coming from the first process is then suitably compressed and coded.

5.2.2 Progressive Transmission

The so called Basic Compression Scheme (BCS) was devised primarily to study the first of the above mentioned processes and to get some statistical measurements. However, the separation of processes reported in the previous paragraph allowed more thoughts for the second process already. Instead of plainly choosing

one of the conventional compression techniques already studied, the second process was treated rather unusually. The aim was to study compression techniques that relied on spreading the updating information through more than one frame, in the time sense. This is herewith generically called *progressive transmission.* Besides making a trade-off in picture quality (for the purposes of less data to code), another trade-off was to be investigated in delaying the reconstruction of an affected image region. Then, where movement was detected, the load of updating data at the moment the demand arose could be lowered.

The inspiration for investigating this method arose from the work of Dreizen [Dre87]. This was a lossless progressive transmission method for grey-scale images which used early transmission efforts to favour areas of greater image information content. The method was devised for image transmission through very low bandwidth channels.

Evidence from research on temporal and spatial resolution requirements also contributed to direct this thesis' research in the direction of a progressive transmission method. In the spatial aspect, Miyahara [Miy75] studied moving objects that are not so easily tracked (such as head or hand movements, quite common in video teleconferencing) and found that subjects could detect loss of resolution less easily. In the temporal aspect, Seyler *et al* [SB65] found that spatial resolution could be reduced significantly immediately after a scene change, with restoration to full resolution not needing to be fast. If full resolution came within approximately 0.75 s subjects would not notice the temporary resolution reduction.

The dynamics of a progressive transmission scheme are not simple, however. Assume, for instance, that the adopted scheme updates a portion of image during the next 4 frames. In the case of an image sequence where a single frame contains movement and all others are static, it is clear that the updating data will be nicely spread along 4 frame periods, effectively lowering the short term data volume to a quarter of the original. That was the main objective of the scheme, of course.

A real image sequence, however, is more likely to have every frame containing moving regions, and so the average amount of updating image data per frame is about the same as in the case of plain image updating, i.e. no progressive scheme employed. Apparently, then, it would be pointless to devise a progressive mechanism, since no advantage is in sight.

The analysis of the process, though, is not that simple. Locality in the mechanics of movement suggests that there is a great chance of an image region

being detected as having movement during several consecutive frames. The scenario one can see, in an average of the frames of a sequence, is as follows: a significant share of the moving image regions of a given frame are bound to be flagged as moving again in the following frame.

Let us take again the previous example with a progressive updating scheme lasting 4 frame periods. For an image region found moving in a given frame in the sequence, the first quarter of the updating information is sent during that frame period. If the same region is found moving again in the next frame, it is a waste to continue sending the rest of the old updating data. Instead, the obvious action is to throw away the current stage and start again from scratch, that is, to send the first quarter of the updating data, but taken from the new state of the image region. These events may repeat for the time span of the burst of movement in the region.

Generalising for a continuous moving scene, one can assume that the updating process of a great deal of the moving regions will not have to be done in its entirety (most updating processes will be started but never finished), so providing significant savings in the volume of updating data. Of course there is a drawback: if the updating process lasts too long, the observer will certainly notice the visual artifacts caused by malformed blocks of image. For such strategy to be useful, a compromise solution will have to be sought among i) the duration of the updating process, ii) the profile of such process and iii) the end image quality desired. By profile it is meant the speed with which information content is restored to the reconstructed image.

In summary, the BCS (Basic Compression Scheme) will tackle image compression in two ways: firstly, a movement detection strategy will attempt to remove frame-to-frame redundancy; secondly, a progressive updating scheme will attempt to spread the updating process along a number of frame periods. The results obtained will be useful in devising a more elaborate compression scheme.

5.2.3 Movement Detection

Movement from a three-dimensional world mapped to a two-dimensional plane, such as a video screen, may be composed of several basic components. The easiest and commonest to model is plain translation, but rotation also occurs very often (even if for short periods of time) and is more computationally demanding [NL80]. Other composite movements cause object distortion and are difficult to model.

This difficulty can be confirmed by realising that in motion compensation very little work has been done outside translational movement. This simplifying approach, however, provides results good enough for most situations, and the reason for that is the small size of the image region over which to apply the analysis, either in the field of motion compensation or, as is the case here, in movement detection.

In movement detection, though, the assumption of translational movement is only partly useful. This assumption can influence the design of the detection algorithm but is not essential, as will be seen following. Also, of great interest to this project is the presence of noise and how it affects movement detection. It has already been seen in Chapter 4 how camera noise can affect image quality and compression.

5.2.3.1 Single Pixel Detection Strategy

For broadcast quality image coding requirements for image motion breakup are more stringent than for video teleconferencing purposes. Early coder implementations cited in the literature employed straight conditional pixel replenishment to preserve image quality: individual pixel differences were computed over the whole image, no matter what the amplitude of these differences were. Compression rates were very low, in the range 1.5-3:1, yet such compression figures made no doubt a big difference for broadcast quality transmission under bandwidth constraints: in the recent past a number of coder implementations have been reported in the literature using this sole technique [O'N66, K. 77, III$^+$77].

The foreseen applications for this project put its requirements closer to video teleconferencing, however. Compromises in image quality are not only acceptable but very often assumed as a starting point. As previously said, basic to most detection strategies in video teleconferencing lies the concept of a tolerance or threshold value for the brightness changes. Differences are then considered *significant* if their magnitude is above the given threshold. This helps in the final compression rate, since small changes in the image are avoided (that is, not significant changes are left out), as well as part of the noise inherent in the video acquisition process.

To minimise the effects of unwanted odd changes, a significant pixel change is first checked against neighbouring pixels to see if it is isolated with respect to other significant changes [NL80]. The definition of 'isolated' varies in the sense

of which pixels to consider as neighbours, but this is not a major issue. The basis for not considering isolated changes is the reasonable assumption that the movement of a visible object on screen implies a number of pixels close together changing at the same time.

5.2.3.2 Detecting Movement over Blocks of Image

Detecting differences over single pixel values has some disadvantages: i) single pixel changes contribute almost nothing to the end quality but still add to the coded stream; and ii) single pixel occurrences are greatly increased by noise in the acquisition process, further worsening compression performance[1].

If differences are, however, computed over a group of neighbouring pixels as a whole, it is possible to avoid most of the overhead caused by noise on the compression process. If a pre-established criterion is set and its value for a given block of image found to be within the significance threshold, then no updating information is generated for the block of image considered. The image block size can usually go from a single pixel to regions as large as 16x16 pixels).

In video teleconferencing there is no preferred direction of movement. Contrary to normal broadcast programmes, where horizontal movement is slightly predominant, a person might move the head in all directions while speaking. The most obvious shape for detecting movement is then a square of pixels.

Unfortunately, this approach has its own disadvantages. If the significance threshold is set too high (for example, if the coder down the compression process feeds back that less data should be generated), the number of blocks to be updated will be smaller, as expected. This causes a number of blocks to fall in the situation of having *almost* deserved an update, i.e. their difference value falls just short of the threshold. This means that a non-updated block might enclose already visible brightness changes in its appearance and not be updated. Moreover, the larger the block size, the more likely that major brightness changes internal to the block translate into no significant difference on the selected criterion. This is specially true if such criterion is as simple as a sum or average of all pixel values inside the block.

The effects that such occurrences may cause to image quality on reconstruction are a bit disconcerting. Take, for example, two neighbouring image blocks, with

[1] In fact, the Vidicon cameras seem to generate pixel changes from frame to frame for most pixels, even with a still scene.

an edge lying across both blocks. Suppose that the edge moved slightly between the previous and current frames, and that one of the blocks happened to be updated in the current frame, whereas the other did not. On the one hand, the particular image block that missed updating is on its own a good quality image (supposing no other compression scheme is harming the image), since it is exactly the original image block as it was in the previous frame. On the other hand, if the direction of the edge movement is favourable to this argument, there is a great chance that the edge will be broken at the block boundaries. This problem can be observed later in Figure 5.3.

As seen in Section 3.3.8.2, the human visual system is quite sensitive to edge positional accuracy. The main problem with edge inaccuracy is that the problem is quickly spotted by an observer, whereas normal distortions caused by, say, a low-pass filter, are not so apparent. Also, the shape of both blocks tend to be highlighted, since the background images these blocks convey might not match any more (the human visual system is known to enhance edges with little brightness differences between both sides of the edge [Pra78]).

5.3 A First Implementation of BCS

5.3.1 Movement Detection Settings

In order to gauge the effectiveness of the approach described up to this point, the algorithm was implemented and some measurements and image quality assessment were done.

The image is subdivided in non-overlapping blocks of 8×8 pixels. The movement detection module performs a simple summation of all values in a block. This value is calculated for both the previous and the current frame being processed, from a given sequence. The difference of the block sums for both frames is compared to a given threshold, as a measure of image activity: if above it, the block is fed to the updating process; otherwise it is not touched. It should be recalled that this coding scheme will keep "still" blocks as they were in the previous frame, only sending updating information for the ones found with movement.

The purpose of choosing a certain threshold level is to try to avoid updating blocks whose brightness changes were only due to the inherent noise in the acquisition process. Rather, the updating process would ideally pick up only

the blocks where movement really occurred. A first idea of what this threshold should be can be obtained by looking at the noise figures of the acquisition process (Chart 4.1). In this particular case the acquisition process presents noise with a standard deviation (σ) of ≈2.5 brightness levels out of 256.

In order to allow brightness variations due to noise to pass unseen by the detector a range of allowable variations needs to be established. Starting with the single pixel case, the range of $\pm\sigma$ around the noise-free pixel brightness covers $\approx \frac{2}{3}$ of the possible individual pixel brightness occurrences. Multiplying this range by the number of pixels in each block gives:

single pixel: $\pm\sigma$ → range = 2 (2.5) = 5

8×8 block: 64 (single pixel) → range = 64 (5) = 320

This figure should only be used as a rough guide for starting any experiments. Firstly, only $\frac{2}{3}$ of the possible variations seem to have been covered. Secondly, by taking a collection of pixels the uncertainty caused by the noise is greatly reduced. Bearing this in mind, it was decided to perform initial experiments with a threshold level of 256 (there was no particular reason for choosing this particular value, apart from being less than the value obtained from the above chart).

5.3.2 Updating Process Details

The updating process follows the progressive transmission idea presented in Section 5.2.2, mixed with a variation of the linear quadtree encoding scheme presented in Section 4.6.6.

The image is subdivided in non-overlapping blocks of 8×8 pixels, giving 1024 blocks for a 256×256 image. Each block is coded as a quadtree, whose order is coded together with the block identification. To allow flexibility with block sizes and image resolution, 16 bits are allocated to code both the block number and its order, as can be seen in Figure 5.1. Extra bytes are appended to each block identification according to the block order.

When a block is detected with movement, the updating process starts the progressive transmission by sending a block of order 0 (i.e. a single brightness value corresponding to the average brightness for the whole block) in the current frame period. In the next frame period, the image information is enhanced by sending a block of the next higher order (i.e. 4 brightness values corresponding to the average brightness of the 4 leaves or quadrants of the order 1 quadtree). The

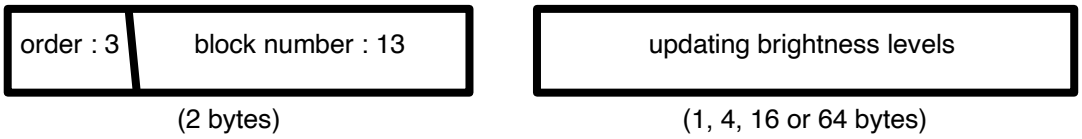


Figure 5.1: BCS encoding format for 8×8 pixel blocks.

process continues till the brightness levels correspond to individual pixel values (order 3, 64 brightness values). Since each stage in the updating process does not depend on past information, the brightness information is always derived from the current frame. It should be pointed out that the process only follows this route if no movement is detected during the span of the complete updating process. If movement is detected at any stage during updating, the updating process is restarted from scratch. The updating process is illustrated in Figure 5.2.

5.3.3 Preliminary Results of BCS

For this initial implementation two image sequences were used. Sequence *hand* was aimed at studying the effects of compression methods under a great deal of movement. It shows a person sideways lowering his arm from top to bottom of the screen along the 64 frames of the sequence (shot at a resolution of 256×256 pixels). In average there is about 40% of the image in movement. Sequence *talk* was aimed more at what a normal video teleconferencing scene should look like, i.e. a head and shoulders scene of a person talking. Resolution was the same as the above sequence.

As seen in Section 5.3.2, the encoding format employed was not intended to be optimum in terms of compression. This format was chosen for being convenient for research purposes; even so, the complete BCS coder produced quite good compression figures. The results on Table 5.1 are overall compression rates taken over 63 reconstructed frames out of 64 original frames, assuming that the first frame was already up-to-date.

These figures are, however, obtained at a large compromise in image quality. Figure 5.3 shows two frames, one intermediate and the final one, of sequence *hand*. Frames from the original sequence are shown in figures *a* and *b*, whereas frames reconstructed from the BCS coder are shown in figures *c* and *d*.

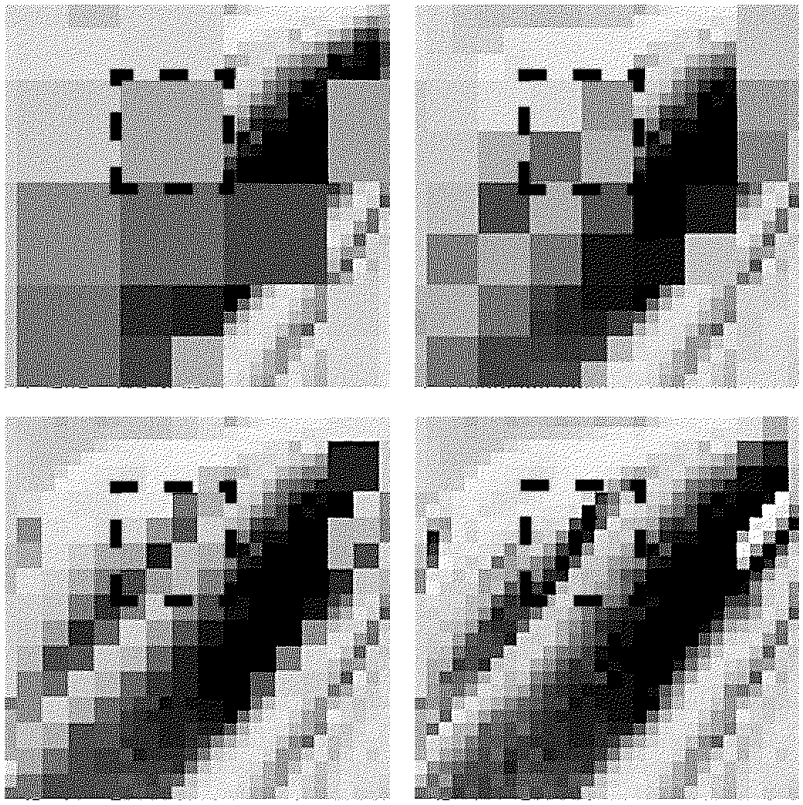


Figure 5.2: BCS updating process, if no further movement is detected.

Figure 5.3: Samples of BCS coded images of sequence *hand*.

sequence	compression rate	SNR (dB)
hand	21.00 : 1	28.35
talk	41.79 : 1	29.91

Table 5.1: Results of BCS coder.

There are basically four weak points with BCS:

- image regions where movement is more apparent (relatively fast movements) showed very poor rendition, although in the original sequence such regions are not good either.
- the progressive transmission scheme is very hard on apparent image quality, due to the brightness shape of the blocks during updating stages, particularly with blocks of order 0 (a single brightness level for an 8×8 image block). If 256×256 pixel images are displayed at full monitor size[2], each original pixel must be replicated three times for the image to occupy the whole screen. Blocks of size 8×8 then become quite distinguishable from the background, particularly if the block does not match the surroundings.
- at the threshold level chosen (256) many blocks escaped being detected with movement, even if these blocks did present movement. After a number of frames the result was a great deal of distortion, more visible on broken edges, but also happening with textured backgrounds
- the dynamics of the image sequence reveals more about this coder. Examining the reconstructed sequence on screen shows those large square blocks more disturbing than they look on static frames, such as the ones presented in Figure 5.3. As the arm swings downwards, the large order 0 blocks seem to crawl down the image.

Despite the bad end image quality, BCS showed that the combined use of movement detection and progressive transmission presents a good potential regarding compression rates. Ways of improving the weak points of this scheme are not difficult to point out. Firstly, the movement detection strategy needs refining to be able to better sort out movement from noise. Just decreasing the threshold level is not enough: measurements showed that even at very low thresholds the

[2]the goal of this project is to deal with standard television images. The research has been done using a monitor capable of displaying an interlaced image of size 512×512 pixels.

detector leaves severely broken edges. Secondly, the updating process starts very badly with the order 0 quadtree blocks and the strong blockiness effect these large blocks cause. A better approximation of the image for the first updating stage will certainly make a lot of difference (anything that smears the edges of the squares will make a difference).

Better movement detection is discussed in the next sections. Better image descriptions for the first stage of the updating process, however, will be dealt with in Chapter 6.

5.4 Movement Detection Strategy Variations

Movement detection following a plain sum of brightness levels may lead to wrong conclusions about whether a given block has no significant changes and should be left without updating, as seen in the previous section.

Alternative methods must be looked at to attempt to localise brightness changes inside a block, so that the sum of variations in one region is not compensated by the sum in another region, inside the block. Figure 5.4 shows three approaches that are more suitable to that intent than a plain sum over the whole block.

Figure 5.4a (*flat* detector) starts by subdividing the scope of the sum into four quadrants, so that changes in one quadrant that could be compensated by changes in another quadrant are kept unbalanced. All detectors presented in this section return a status of whether movement in the block was detected or not, and whether movement was detected in a single quadrant or in more than one.

Figure 5.4b (*crossed* detector) shows a variation that attempts to catch small movements inside a block quadrant. The sums of each diagonal are compared in turn with the ones in the previous frame to detect changes. The reasoning behind this approach is the high sensitivity of the visual system to edge positioning.

Figure 5.4c (*helical cum checkerboard* detector) elaborates further, trying to detect edge changes inside a quadrant, in a different orientation than *crossed* does. Also, the checkerboard pattern matching allows the detector to catch changes in textured regions.

A fourth approach (*mse* detector) is a variation of the *flat* detector that, instead of working with sums of brightness differences, works with squares of brightness differences, as the abbreviation might suggest. A sum of squares of

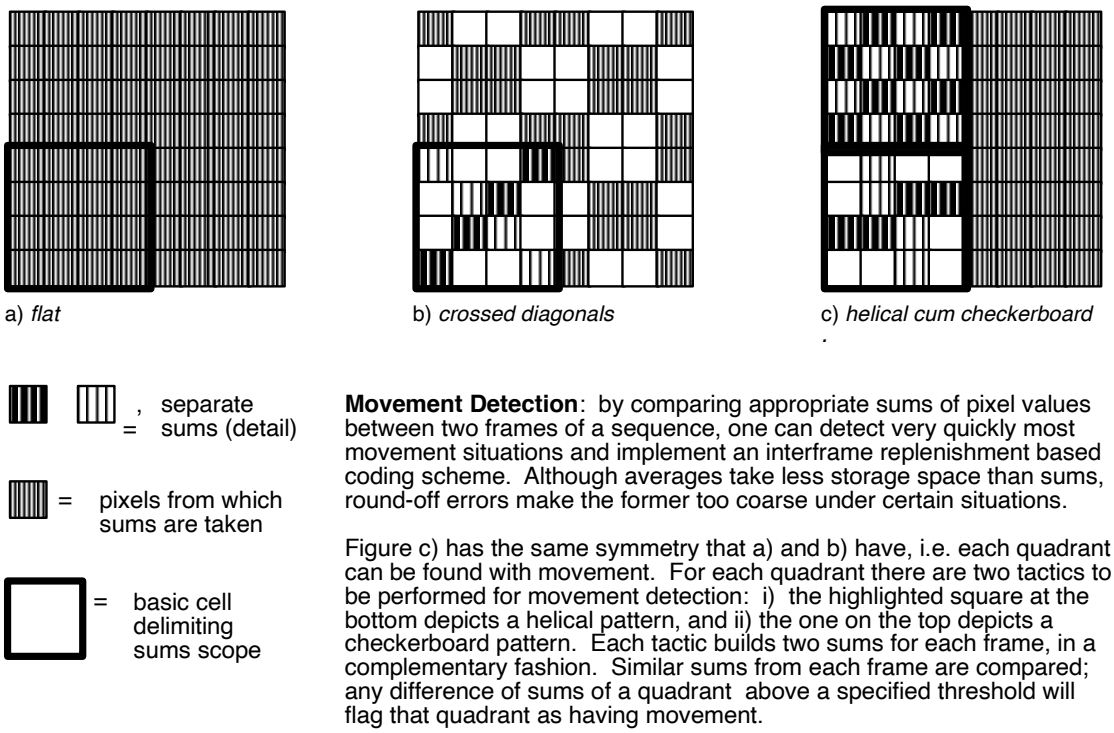


Figure 5.4: Alternative movement detection strategies.

pixel differences is calculated for each quadrant; if above the specified threshold the quadrant is flagged as with movement. As with the previous detectors, more than one quadrant flagged will cause the whole block flagged as with movement.

Establishing suitable thresholds for each detector proved to be an empirical exercise. As a starting point, the standard deviation of the noise (as measured in Section 4.1) was used, multiplied by twice the number of pixels over which the sum applies, as seen in Table 5.4.

Threshold levels were varied and tried to give good image quality, not only in the sense of isolated frames, but also in the overall appearance of the sequence when displayed in real-time[3]. Establishing what "good quality" means for each type of detector would require extensive testing with subjects. Instead it was decided to perform a not so precise comparison, for practical purposes. The chosen threshold values were as Figures 5.7 and 5.8.

[3]The other constraint in choosing suitable values was obviously to result in fewer updated blocks overall (less data to be coded afterwards).

block size: 8×8 pixels, standard deviation (σ): 2.5

detector	*pixels involved*	*suggested threshold*	*best practical values*
plain	64	320	40
flat	16	80	25
crossed	4	20	20
helical cum checkerboard[a]	4	20	20
helical cum checkerboard[b]	8	40	25
mse	16	80	600[c]

Table 5.2: First guess of suitable thresholds for alternative detectors.

[a]helical detector.
[b]checkerboard detector.
[c]the *mse* detector works with squares of differences, making this figure not follow the behaviour of other detectors.

5.4.1 Comparison of Alternative Movement Detectors

The author considered processed sequences to be of good quality if distortions would not be easily apparent in still frames and if observing the sequence in real-time would not be disturbing due to artifacts resulting from processing. The selection of thresholds reported does not intend to be considered as absolute limits for good quality, but rather as a means of practical comparison.

For purposes of comparing the detection methods, the following tactic was used to process image sequences:

- a frame from the sequence is compared with the previous frame using one detector;
- blocks detected with movement in one quadrant only had that quadrant copied over the corresponding region in the previous frame;
- blocks where more than one quadrant was detected with movement were copied over the corresponding block in the previous frame;
- the next frame was taken from the sequence, with the modified frame considered to be the previous frame.

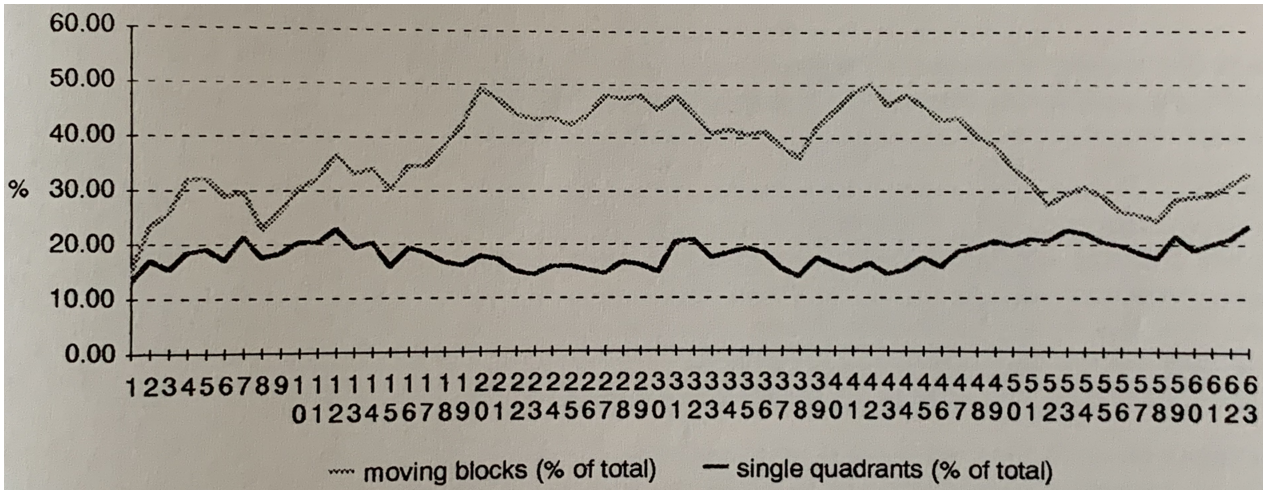


Figure 5.5: Block updating, *mse* detector, *talk2*.

- the very first frame of the sequence was considered to be present and completely up-to-date, so that a 64-frame sequence generates 63-frame processed sequences.

The sequence *talk2* (64 frames at 256×256 pixels) was chosen for being more representative of typical video teleconferencing scenes. The amount of movement in the sequence is above normal, since the head swings down and up again during the sequence. Also, slow movement of the body is present. First, let us see how the sequence behaves when reconstructed after being coded with one of the detectors, say the *mse* detector. Figure 5.5 shows the evolution of the number of updating blocks along the sequence. Figure 5.6 shows the evolution of the Signal-to-Noise Ratio (SNR) in decibels, when the reconstructed sequence is compared with the original one. In both plots there can be seen two peaks, which correspond to the major movements that occur along the sequence.

Figure 5.7 shows a comparison of the performance of the collection of detectors and the original detector (plain sum over the whole block), for image blocks of 8×8 pixels. The statistics were averaged over the whole image sequence. The number following the detector represents the threshold used; for the *helical cum checkerboard* detector the first number is the threshold for the helical part of the detector. Figure 5.8 shows the amount of image data generated by each detector, in fact normalising the figures for the two kinds of blocks (an 8×8 block and a 4×4 quadrant) into a common measure. It should be remembered that single

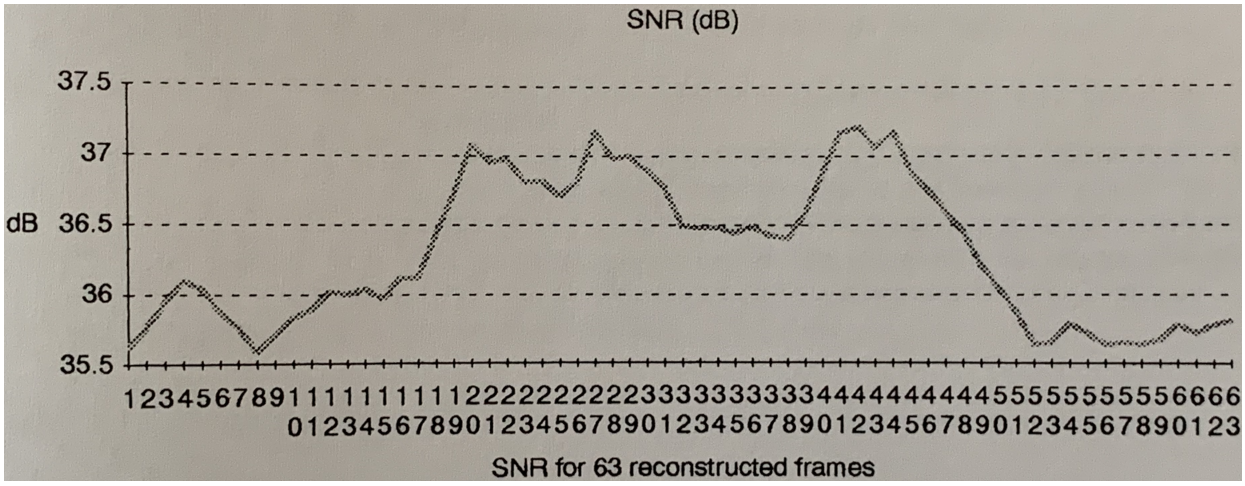


Figure 5.6: SNR, *mse* detector, *talk2*.

quadrant blocks generate a quarter of the image data generated by a full block.

5.4.2 Comments

When evaluating image quality the first thing to be aware of is that a single criterion is not sufficient for measuring final image quality. The signal to noise ratio figures are calculated using the Mean Square Error (MSE) measure. There is a lack of consensus in the literature about a criterion that measures image quality with regularity [Pra78]. The different types of possible distortion cause most of the criteria to succeed most of the time, but not always, in matching distortion measure with perceived image quality. Another difficulty is that some criteria pose high complexity of implementation. These are the reasons for MSE to be widely used in image processing.

Bearing that in mind, the figures in Figure 5.7 must be taken cautiously. The perceived image quality must be taken into account when evaluating the detectors. In fact, although all detectors present similar SNR figures, the resulting image sequences differ quite significantly in perceived quality.

It would seem that the *plain* detector is not too bad after all, when seen in the light of other detectors' performances. The average SNR obtained is about average in the collection. However, the resultant image quality is definitely below all other detectors', when examining the sequence in real-time. Lowering the threshold does not help too much, since the detector then is blinded by the noise

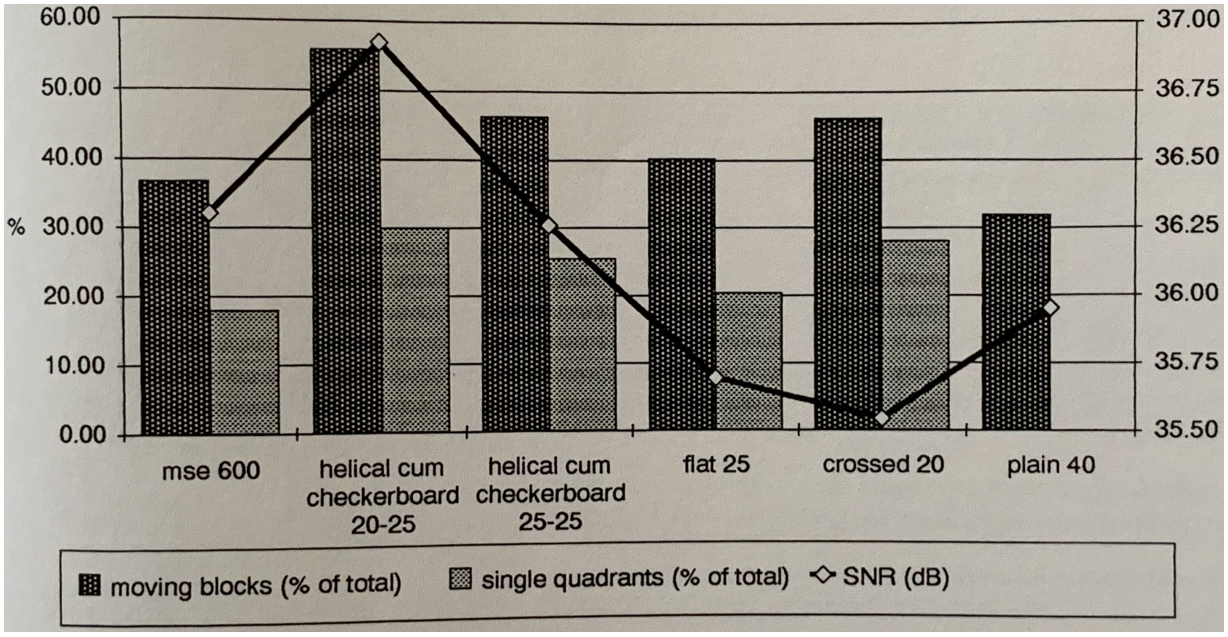


Figure 5.7: Comparison of alternative movement detectors, sequence *talk2*.

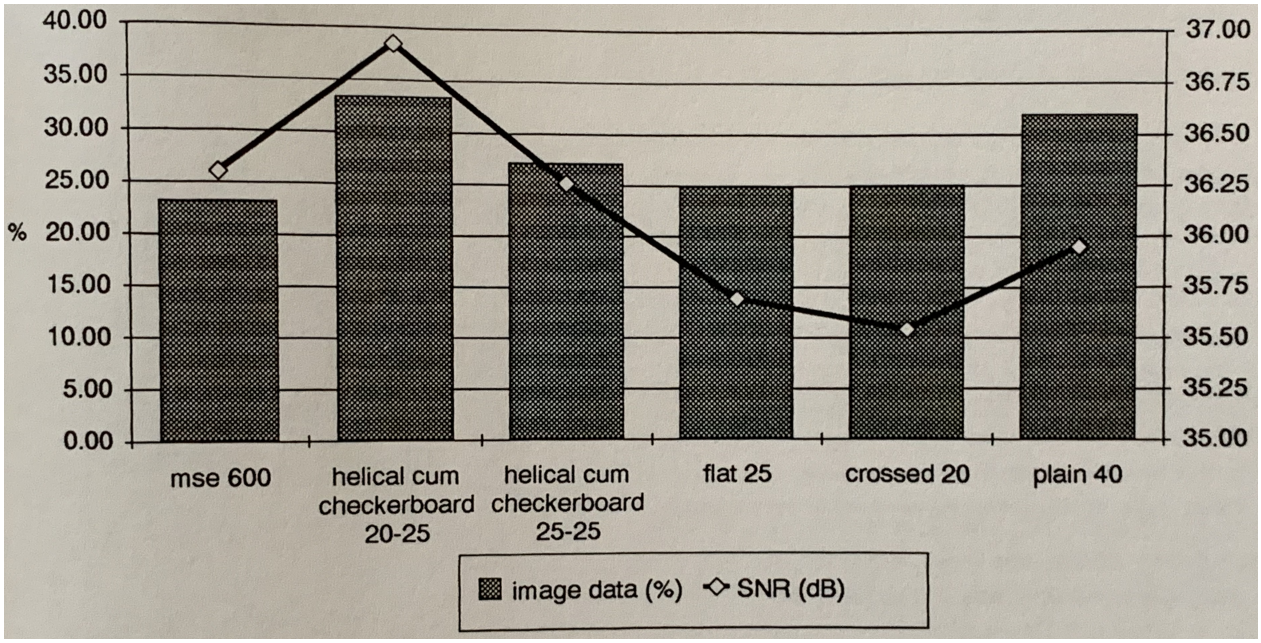


Figure 5.8: Normalised Comparison of movement detectors, sequence *talk2*.

and starts flagging blocks indiscriminately. The problem with this detector can easily be identified as the size of the block: whenever a block is left in an old state when the surrounding ones are all up-to-date, it becomes quite visible. It is highly likely that such a block will at last be updated in the next frames, and when it does happen the movement is highlighted from the normal smoothness of the background. This is particularly disturbing when the sequence is displayed in real-time.

Next in terms of quality come the *flat* and the *crossed* detectors, and slightly better than these come the *helical cum checkerboard* and the *mse* detectors. The difference in quality is not significant, though. Figure 5.9 shows that the image quality talked about here is indeed good: the processed frame is virtually identical to the original one, in terms of visual appearance. These pictures refer to frames number 32 in sequence *talk2*, from the original (a) and from the reconstructed one using detector *mse* 600 (b).

The plot in Figure 5.8 also shows that the *mse* detector might be the most economical one, given a certain target for the SNR. Again, this can be slightly misleading: since the detector employs the same criterion used by the distortion measure, it may be slightly favoured and present a better SNR figure. Examining the sequences in real-time, however, confirms a slight advantage for *mse* 600 against *helical cum checkerboard* 25-25, even though the SNR figures are comparable. All these detectors, tuned at this quality level, were able to produce good image dynamics when the sequences were displayed in real-time, whilst still providing good reduction in raw image data.

The four alternative detectors achieve better detection characteristics than the original plain sum of pixel values with a threshold of 256 (Figure 5.3); this, however, comes at a price, adding more data to be coded by the rest of the compression scheme to be used. Nonetheless, better detection strategies provide a better visual quality, both by decreasing the amount of blocks left without updating and by effectively decreasing the size of individual blocks. Large blocks contribute a lot to broken edges and related artifacts on the image, which are easily picked up by an observer.

An interesting feature of all the detectors studied here is their ability to be tuned, so that more or less image data is generated, according to what external restrictions might impose on a complete coder. As an example Figure 5.10 plots the performance of the *flat* detector as its threshold is varied over a suitable

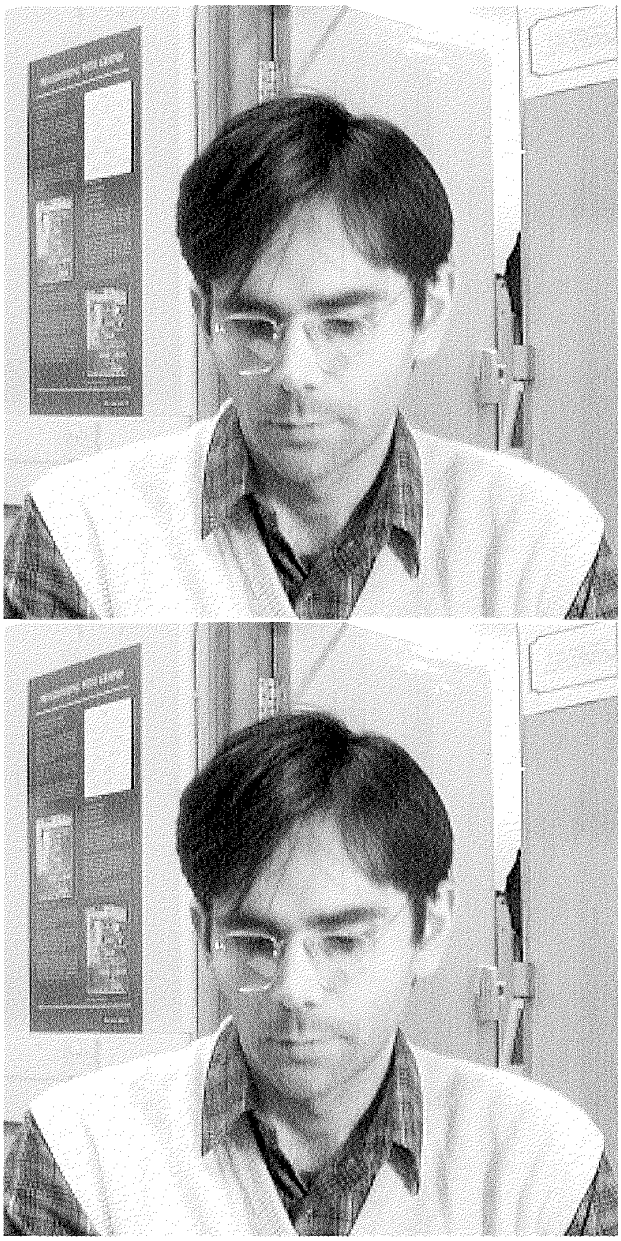


Figure 5.9: Comparison of original and *mse* detected, sequence *talk2*.

range. Naturally, image quality is affected and follows a good compromise with the amount of image data generated. The sensitivity obtained by judicious tuning of a parameter makes these detectors quite attractive for the sort of compression method desired. The output of the coder can very easily be slowed or accelerated as conditions require.

Figure 5.10 also shows the point where the detector turns blind, i.e. when it no longer can discriminate between noise and significant movement changes. Somewhere around a threshold value of 25 the amount of image data generated quickly shoots up, with no appreciable differences in the SNR figures. Another point of interest is that the number of single quadrant blocks does not increase when bursts of movement occur. The moving blocks plot (top) clearly shows the two bursts of movement this sequence has, but the single quadrants plot (middle) does not show a general increase at the burst regions. For high threshold values (low sensitivity) there is an increase in the number of single quadrant blocks; for middle threshold values the increase is less noticeable; and for low threshold values (high sensitivity) the pattern actually inverts, with the number of single quadrant blocks going down during the bursts of movement.

5.5 Conclusions

There is evidence from visual inspection by the author that the more elaborate detection strategies provide better image quality. Also, for approximately similar SNR figures the alternative detectors generate much less data than the *plain* detector. The computational power required is hardly different amongst all strategies, with the *mse* detector requiring a little bit more than the others. Extra memory is needed to store the quadrant sum values, as compared with the original strategy. The use of 16-bit memory locations for the sums is enough to cover sum values for all strategies. This store unit size is used not only for simplifying purposes, but also because it provides faster access for most microprocessor implementations. With the original plain sum strategy, the extra memory needed to store the sum values amounts to just over 3% of the whole image data (at 8 bits/pixel), whereas with the worst variation (*helical cum checkerboard*) the memory required amounts to 50%, which is not really significant with today's memory cost and abundance.

It is worth noting at this point that the computational cost of movement

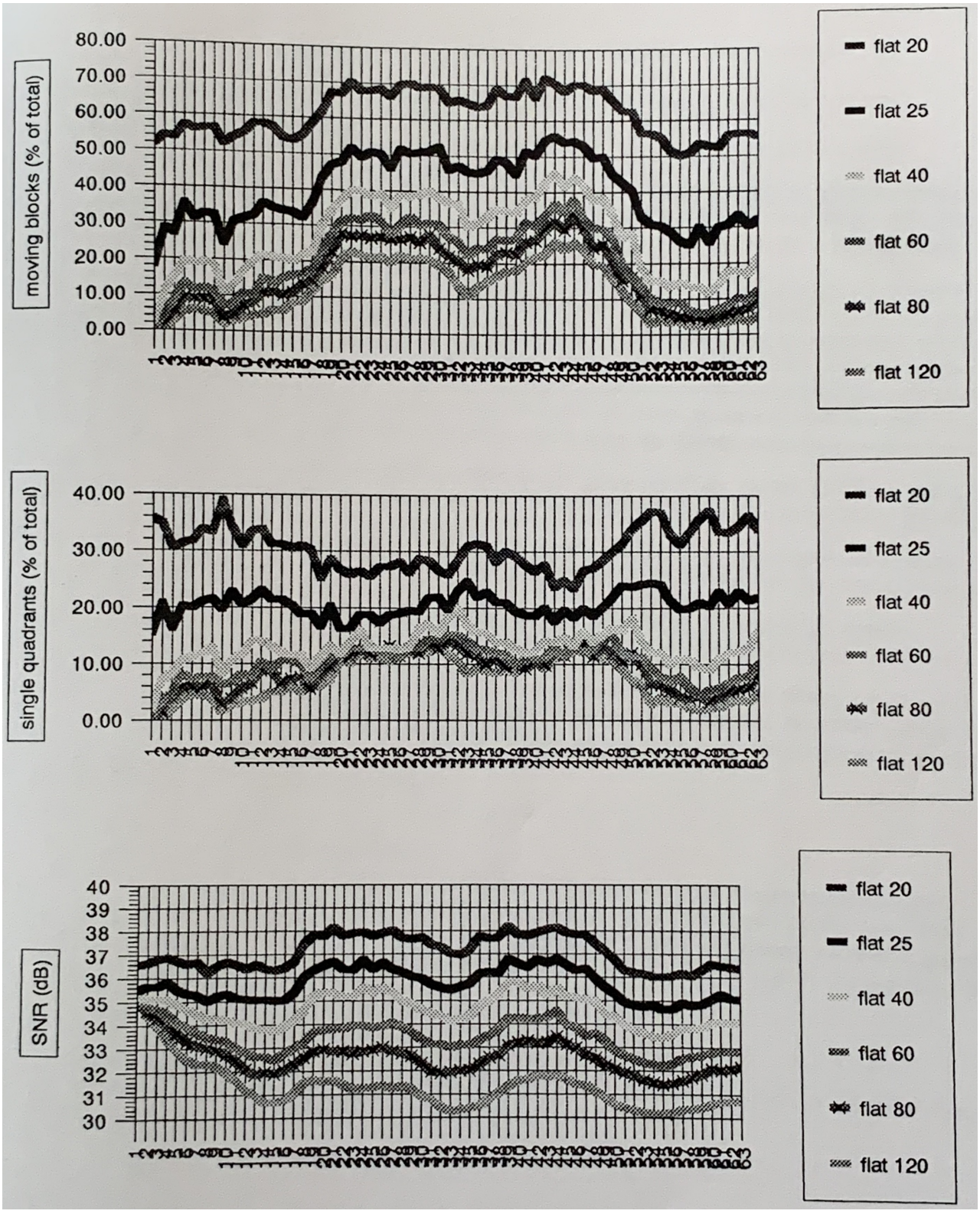


Figure 5.10: Performance of *flat* detector at several thresholds, sequence *talk2*.

detection, following one of the strategies above, is a fraction of that of any noise prefiltering that might be used in a complex compression scheme[4]. There is little penalty in using one of the more sophisticated approaches discussed above, and the benefits are well worth the effort. Any of these strategies are recommended for use with the more complex compression scheme to be developed. The savings in image data to be coded amount typically to three quarters to half of the full image data (*mse* detector, Figure 5.8), in other words a compression rate of $\approx$ 2–4:1.

Regarding the first image description of blocks of order 0, the approach described in Section 5.3.2 proved to be of bad image quality. The length of the updating process may also have a share in the bad results. Besides finding a better image description for blocks of order 0, the length of the updating process needs investigation. These issues will be examined in the next chapter.

[4]the movement detection strategies presented in this chapter are all dependent on the magnitude of the inherent acquisition noise. An implementation of these strategies requires setting the system for the particular camera–frame grabber combination.

Chapter 6

An Image Compression Scheme Based on Vector Quantization

6.1 Introduction

This chapter continues with the improvement of the BCS coder, this time tackling the updating of blocks detected with movement. The initial phase of this process will be referred to as *early image shape description*.

First, candidate techniques for the task of early shape description are examined and the ones satisfying the initial requirements are selected for further thought. Next, the choice of Vector Quantization is justified.

A complete compression scheme (MDPT/VQ, Section 6.3) that builds upon the knowledge gained with BCS is then devised, and experiments with three variations are reported, to evaluate the suitability of the general scheme for the implementation of a Video Viewer.

The chapter finishes with the conclusions drawn from the experiments, discussing the flaws and suggesting ways of further improving the compression scheme.

6.2 Alternative Early Image Descriptors

6.2.1 BCS Revisited

BCS, as reported in Section 5.3.3, showed some weak points, severely affecting the image quality. The compression rates achieved, however, hinted that the method

had a good potential, as long as improvements in the right areas were made.

The first improvements were presented in Section 5.4 and concerned the movement detection strategy employed in BCS. Both the strategy employed in BCS and the threshold for the pilot trials were not adequate for smooth frame updating. The alternative movement detection strategies presented there were capable of achieving perceived image quality almost identical to the original, whilst retaining good savings on raw image to be coded (fewer flagged blocks). All those alternative detectors achieved comparable results and any choice among them would eliminate most of the problems listed on Page 5.3.3.

The remaining issue was the appearance of a block detected with movement, at the moment of the first update. A single brightness value for the whole block proved unsatisfactory due to the strong blockiness effect perceived visually. A better image block description was needed for the first update, one that more closely resembled the brightness shape of the original block. At the same time, this description should be encoded using a very tight code. By using tight codes for all blocks flagged with movement the conditions for a good final compression rate are guaranteed; if a block is flagged with movement again in the next frame, it does not cause the transmission of the bulkier image information that follows the first shape description. This is the basis of the progressive transmission embedded in the compression scheme being developed in this thesis.

To illustrate the importance of a tight code for the first shape description, recall the characteristics of the BCS coder: for an 8x8 block flagged with movement, a single brightness value is coded; in other words, 1 8-bit wide tag represents the image area occupied by 64 brightness values of 8-bit width, providing a 64:1 compression rate at this stage.

For an effective operation of a compression scheme based on the progressive transmission idea, the first shape description should provide a compression rate of at least 16:1 in the first stage. The next section will examine what alternatives the author found to suit this requirement. These alternatives should take into account that another technique sends complementary image information in the next frames (if no further movement is detected). It would be desirable, but not essential, to integrate both actions (early and complementary shape description) and exploit extra compression that this would allow.

In the next section, a few image coding techniques that fit the above requirements are examined in order to improve the early shape description stage.

6.2.2 Candidate Techniques for Early Block Shape Description

The candidate techniques should work in a block-based approach so that integration with the compression model introduced by BCS is possible. Among the techniques examined in Chapter 3 most are not suited to working with blocks of image. Variations in the average brightness of a block from frame to frame are likely to be of low amplitude, but high variations do occur. Therefore, the early shape description must not be limited by a scheme such as DPCM, where increments cannot exceed certain amplitudes. If differential modulation is employed, escape situations must be provided by the coder so that higher difference amplitudes are properly encoded.

The possible choices for this task include: i) subsampling with interpolation; ii) Block Truncation Coding (BTC); iii) transform coding; iv) Vector Quantization (VQ).

6.2.2.1 Subsampling with Interpolation

A block of image can be given a reasonable description by subsampling the original block at a certain reduction factor and, on reconstruction, by interpolating to give an array of the same dimensions. The basic requirement of at least 16:1 compression makes a possible use of subsampling more difficult to be seen. With 4x4 blocks the coder would have to work with only one brightness value, which falls back into the the original BCS approach, i.e. a single value to describe the brightness of a block. With 8x8 blocks, however, 4 values could be transmitted and interpolation at the reconstruction phase could yield a fairly decent early shape description.

The trouble with this approach is that the interpolation function should be fairly elaborate to allow a reasonably good reconstructed shape, and the computational power required for high order interpolation is considerable, as seen on Page 3.3.7. Coding would be easy, but reconstruction would make a real-time implementation more difficult. Furthermore, some particular renditions could turn out to be very poor due to aliasing, and proper measures must be taken to avoid it, such as prefiltering or sample averaging. Prefiltering, for example, is a computationally-intensive task.

A block-based compression technique normally benefits from encoding the

average brightness of the block separately. In some techniques this is really part of the technique itself. In transform coding, one of the coefficients is the average brightness and is coded with almost all of the original quantization levels. In Figure 3.13, for instance, 7 bits are allocated to the top left coefficient, which is the average brightness of the block. In BTC, the average brightness or a similar measure is part of the encoded information. In VQ, some variations encode the average brightness or similar measure separately, with improved results.

A similar approach could be extended to the subsampling plus interpolation process. Instead of coding a number of separate brightness values at 8 bits/pixel, one codes a single value at this resolution, most probably the average brightness, and then codes a subsampled array of elements with fewer bits each. Taking again the example of an 8x8 block, subsampling the array at one every four pixels in both dimensions results in 16:1 compression, with four 8-bit values (a 2x2 array). If, however, the average brightness is coded separately with 8 bits, there are still 24 bits to be used at the same overall compression rate. These bits could be used to convey an array of higher resolution and fewer bits per element, whose values could be DPCM-coded in relation to the average brightness, scaled by a predefined factor. As an example, coding the values with 2 bits allows 12 sampled values to be included, which are somewhat imprecise, but that certainly give a better shape description. Also, aliasing effects are reduced due to the higher resolution.

This approach might be very useful for a simpler coder, where to keep computational requirements low one would use simple bilinear interpolation at the reconstruction end.

Subsampling can also be considered in the sense of decreasing the total image resolution to 256×256, either as a temporary measure to satisfy external constraints in the network, or as part of a less complex implementation of a Video Viewer. A reduced resolution image cannot reproduce background details well, particularly written material that might be in the field of vision. However, together with a less complex coder, reduced resolution can be effectively applied to widen the use of a Video Viewer.

6.2.2.2 BTC

BTC might be useful for blocks of up to 4x4, since the nonparametric quantizer used limits the sort of shape description that is possible with the technique. For

8x8 blocks BTC does not yield good renditions. With the above mentioned block size, the technique uses 16 bits to code the array information alone, apart from the average brightness or similar measure. The result is a compression rate of 8:1 for the array alone, which is a little bit over the pre-specified minimum (16:1). The shape description achieved with 1-bit elements is not very good either.

Note that BTC could be seen as part of a more general technique that includes the extension described in the last paragraphs of the previous section above.

Since the compression rate is out of the range sought, and the quality of shape description is not particularly impressive, its use in the compression scheme being sought was ruled out.

6.2.2.3 Transform Coding

Transform coding looks well suited for the task, since it provides a very good shape description, up to the final quality needed. It also allows reconstruction of an image block in stages, in a sort of hierarchical view. This is due to the intrinsic organization of the array of coefficients generated by most transforms. The top left coefficient carries most of the image energy of the whole block, as it is the block's average. Other coefficients close to it carry proportionally decreasing energy and are usually coded with fewer bits per coefficient. If a progressive transmission scheme is set up so that a bunch of the most representative coefficients is sent first, a reasonably good early shape description could be reconstructed.

Let us consider the example of a 16x16 block and the bit allocation depicted in Figure 3.13, where the coefficients that carry the most energy are concentrated around the top left corner. One could choose to send the first 13 coefficients closer to the top left corner, adding up to 62 bits and achieving a compression rate of 33:1. One could instead choose a slightly better shape description, at the expense of a lengthier code, by also transmitting the next 24 coefficients, totalling 125 bits and giving a compression rate of 16.4:1.

The above scheme also benefits from making use of already transmitted image information. If no further movement is detected the complementary updating information that must be sent in the following frames would then be the next coefficients in order of importance (away from the top left corner). At the reconstruction end the inverse transformation is repeated, this time with more coefficients available and so providing a better image shape rendition. The disadvantage is that the inverse transform has to be repeated a number of times (for

the duration of the updating process).

The directions set in the beginning of this thesis to satisfy real-time requirements and cheap hardware implementation dictate that the compression scheme sought could not demand too much processing power. The analysis of transform coding in Section 3.5.3 revealed that good quality transforms demand excessive processing power. The use of transform coding is therefore not being considered for the early shape description phase.

When cheap hardware becomes available it is reckoned that the use of transform coding together with the rest of BCS will provide good image quality results, at very good compression rates.

6.2.2.4 VQ

Vector Quantization, as reviewed in Section 3.6.2, can provide results that rate reasonably well in the spectrum of compression techniques. At the same compression rate most transforms perform better in image quality.

VQ operation fits the requirements for the early shape description phase, but it only does so when fast search variations are considered. This implies that the tree-searched approach must be used to find the best match among the codewords[1].

Best results are obtained by removing an energy factor from the image block before actually doing the vector quantization. This way the code is split into two parts, one being scalar quantized and the other vector quantized. This vector quantizer uses fewer codewords than the original one to obtain comparable reproduction quality, which allows for less processing load to perform a search in the codebook. The energy factor may be the gain of the elements of the block (conveniently scaled to a common factor) or the average brightness, which is removed from the elements of the block. As with the previous techniques seen above, the separate energy component is usually coded with DPCM to save a few extra bits.

A VQ coder can work with both 4x4 and 8x8 blocks, and still achieve the required compression rate. With 4x4 blocks, a 128-word codebook requires 7 bits for the VQ part of the code. Adding the bits required for the DPCM-coded energy component, the total code length for a block can be, say, 11 bits

[1] the Multistep VQ also has the property of fast search, but quality is not as good as with a normal VQ, due to the restriction on the number of real vectors actually stored in the codebook.

long. The resultant compression rate is 11.6:1, a bit lower than the target set in the beginning of this chapter, but still worth considering. For 8x8 blocks the compression rate can be far better: a 1024-word codebook needs 10 bits to identify a codeword, plus 4 bits for the DPCM-coded energy component., giving a compression rate of 36.6:1.

The advantage of the 8x8 coder above is misleading, however. The 4x4 coder will present a much better image quality, since it has a larger rate in bits per sample. However, the 8x8 coder might be more attractive due to the extra compression rate achievable. On the positive side, this would allow more room for the complementary updating information to follow in the next frames.

An image block reconstructed from a VQ coder has an advantage over simpler methods like the subsampling approach: there is no regularity in the shape description of the reconstructed block. A subsampled block reconstructed with a simple bilinear interpolation, for example, presents regular gradients along the block, whereas a VQ coded one usually presents a shape disturbed by noise. This contributes to a less conspicuous appearance, in the sense of a more natural looking image.

6.2.3 Chosen Technique

Among the techniques examined for the task of early shape description, the ones that satisfy the requirements are subsampling with interpolation and VQ. At comparable compression rates VQ provides a better rendition, due to the more detailed shape description a vector can give. However, the computational load is higher than with subsampling with interpolation, for simple interpolation functions.

The BCS coder can benefit from the use of any one of these alternatives. The subsampling with interpolation approach could be useful as a less computing-intensive alternative. The experiments described in the next sections are the result of working with VQ only, due to its more promising results.

6.3 Proposed Compression Scheme

The proposed compression scheme combines the basic features of BCS, but improved with better movement detection abilities and the early shape description

provided by a vector quantizer. It will be referred thereafter as MDPT/VQ, for Movement Detected, Progressive Transmission with Vector Quantization.
In summary, MDPT/VQ is constituted of the following techniques:

- efficient movement detection that reduces the raw image data to be coded. Image blocks not detected with movement are not encoded.
- an early block shape description scheme to update a block of image flagged with movement. Vector quantization is used to generate the shape description.
- a progressive transmission strategy to complement the image information provided by the early block shape description scheme. The progressive transmission strategy uses either vector quantization or DPCM to replenish the original image shape.

The reasoning behind the progressive transmission strategy is the smaller sensitivity of the human visual system to details of moving regions. This is also helped by the fact that the integration process performed by the camera during video acquisition blurs the moving regions and most edge information is lost. These issues were covered in Section 5.2.2.

The method is actually a general technique, which can use other techniques in place of VQ for the early shape description stage. Both subsampling with interpolation and transform coding may, in theory, perform well for this task.

The combination of the three techniques to achieve the final MDPT/VQ coder can be done in several ways. The next sections will present a few of these alternatives and study their performance in terms of both the compression rate and quality of the reconstructed sequences.

6.3.1 Details of the Codebooks Used with VQ

In order to use vector quantization in the coding process, it is necessary first to build suitable codebooks. For a codebook to approach optimality it is necessary to build it with a sufficiently long (read representative) training sequence.

The vector quantization method chosen was the Separating Mean VQ, since the separated mean adapts well to a DPCM-replenished scheme. To build the codebooks, images from the training sequence were scanned and subdivided into

non-overlapping blocks of size 8x8 and 4x4. For each block, its mean was calculated and this offset removed from the elements of the vector, in other words normalising the vector around its mean (vector elements were signed quantities).

The algorithm for the codebook generation followed the splitting technique described in Section 3.6.2.6. The distortion measure used was the squared error distortion.

Two codebooks were built for the experiments that follow in the next sections. One for 4x4 blocks, with 256 codewords (rate 8) and one for 8x8 blocks, with 512 codewords (rate 9).

The training sequence consisted of four frames from each sequence from a collection of seven sequences. Four sequences of the collection were 64 frames long, at a resolution of 256×256; the remaining three were 16 frames long, at a resolution of 512×512.

In summary, the codewords in the codebooks carry only shape information. Brightness offset in the coding process is handled with a DPCM technique.

6.4 Performance of Some Variants of MDPT/VQ

The next sections report on several possible implementation alternatives for MDPT/VQ. The aspects of the coder where alternatives were tried were: i) whether the VQ coder operated on vector differences or on vector amplitudes; ii) the progressive transmission strategy; iii) the early block shape description strategy.

In all strategies reported in the next sections, the movement detection scheme was not varied. Section 5.4.2 showed that the proposed movement detectors performed comparatively well, with very little overall difference. All experiments were performed with the *mse* detector (set at a threshold of 600), first presented in Section 5.4.

6.4.1 Differential Vectors

The first strategy tried made use of differential vectors. Blocks were reconstructed by taking i) the block's average in the previous frame, ii) the difference of the averages (from the previous to the current frame), iii) the shape signal from the block reconstructed from the previous frame and iv) a vector quantized shape

description. All these terms were added at the pixel level to reconstruct the elements of the block.

The approach revealed basic flaws in the complete algorithm and gave poor overall results, both in compression rates and end image quality. These flaws are summarised below:

- movement detection was operated by comparing the current frame with the reconstructed frame. If the codebook is not good enough, the reconstructed block may take long (i.e. several frame periods) to approach the shape of the input block. The movement detector may then detect false movement, since the error signal is high. This triggers the algorithm to reset to the first phase of the updating phase (early block description), and the algorithm may take slow to converge. Such long updating processes generate a lot of coding overhead, with little improvement in image quality (if the algorithm does not converge, quality may become worse).

- with the differential approach to vector quantization, a codebook not good enough may fail badly on edge tracking. If a sharp edge is in movement, the reconstruction of the background scene suffers due to the high differences in brightness where the edge was. If movement is not detected for many frame periods, the vector quantizer may not be able to compensate for these high differences. The result is a ghost-like contour delimiting the previous state of the edge. The effect is illustrated in Figure 6.1: the image on the left shows the contour of a hand in vertical position, and the right hand one shows a v-shaped contour. These shapes delimit the previous locations of features in the scene.

The above mentioned examples showed that differential VQ coding could not be employed, unless better codebooks were used. This could be achieved by either enlarging the codebook or using different codebooks for different activity levels in a block. Using the VQ coder with the amplitude of the shape signal, instead of the difference signal from frame to frame, produced much better results, for same sized codebooks. A possible explanation is the way in which the codebook was built: the training signal was built using the amplitude of the shape signal (i.e. average brightness removed). Perhaps a proper codebook for a differential VQ coder should be built for useful results.

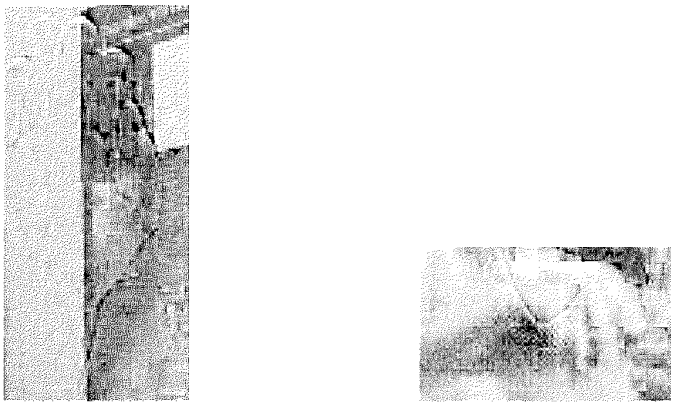


Figure 6.1: Ghost-like contours left by poor rendition of differential VQ.

For the remaining experiments the coder was changed to work with the amplitude of the shape signal.

6.4.2 Block Eight

The main change introduced with this strategy is the way in which movement is detected. In the first item that occurs in Section 6.4.1 it was seen that the basic operation of the movement detection algorithm did not integrate well with the early shape description strategy being tested (Differential Vectors). The inability of the early shape description scheme to completely replenish a block caused the movement detector to become blind and flag blocks indiscriminately.

The solution for this inconvenience is rather simple, given that memory usage is not of much concern nowadays. Another image buffer is kept and updated according to the detector chosen, as if no other compression scheme was present. This way, the movement detection process performs exactly as was described in Chapter 5. Now, the information on which blocks were flagged with movement is now fed to the previous image buffer, the one which holds the image reconstructed by the complete compression scheme (MDPT/VQ). MDPT/VQ, in turn, does its best to reconstruct a block during a few frame periods, but if movement is detected again in the "clean" image buffer, this means that real changes occurred in the incoming image. MDPT/VQ, then, restarts the reconstruction of the given block from scratch.

With the Block Eight strategy blocks were reconstructed by taking the block's average in the previous frame, the difference of the averages (from the previous

to the current frame) and a vector quantized shape description, by addition of these terms.

The above procedure concerns the early shape description only. If movement is not detected in the block during the next frames, the progressive transmission scheme is put into practice.

Other features of this strategy are: i) the early shape description is done by a codeword that maps 8x8 blocks; ii) subblock detection is on, which means that if only a single quadrant of the block is found with movement the codeword sent corresponds to a 4x4 block. The sequence of events, in case no further movement is detected, is summarised below:

1. at the moment movement is detected over the whole block, a codeword for an 8x8 block is sent.

2. in the next frame, the two quadrants corresponding to the positive-tangent diagonal of the reconstructed block are tested for their distortion in relation to the current frame. If the distortion in these quadrants is higher than a certain threshold (t1), two codewords corresponding to the 4x4 blocks involved are sent. If the distortion is lower, the algorithm advances to the DPCM step.

3. in the next frame, the two quadrants corresponding to the negative-tangent diagonal are covered, similarly to the previous step.

4. in the next frame, each quadrant is tested in turn for its distortion in relation to the current frame[2]. If the distortion is below a certain threshold (t1), the updating process is abandoned; otherwise a residual DPCM-based replenishment is done.

5. the previous step is repeated till the distortion falls below the threshold (t1). Each quadrant is treated separately.

Below are some details about the DPCM codes used in the algorithm:

- the difference of averages is coded with a 3-bit DPCM code, multiplied by a factor of 3. Values outside the covered range are coded with full 8-bit PCM code.

[2]note that updating is always compared with the latest frame available; if deeper changes occur, the movement detector will probably catch it and the process reverts to one of the initial phases (either 8x8 or 4x4 codeword).

- the residual DPCM replenishment code employs 1 bit/pixel, so that a low overhead is incurred. One state means an addition of 6 levels, the other state means a subtraction.

- the settings of the above parameters are dependent on a number of other parameters, all rooted in the camera noise level. In tests the value of 6 caused most of the DPCM updating processes to last one frame period, only occasionally going to two frame periods.

Since the image is not coded as a whole, the movement detected blocks must be identified before coding. A coding structure was devised to for storage and accounting of the compression rates achieved. Notice that the coding structure adopted for Block Eight, which can be seen in Figure 6.2a, is not optimal. A more compact code could be devised, but this one allows easy changes to accommodate new strategies.

6.4.2.1 Results

Figure 6.3 shows statistics of the Block Eight coder from three sequences:

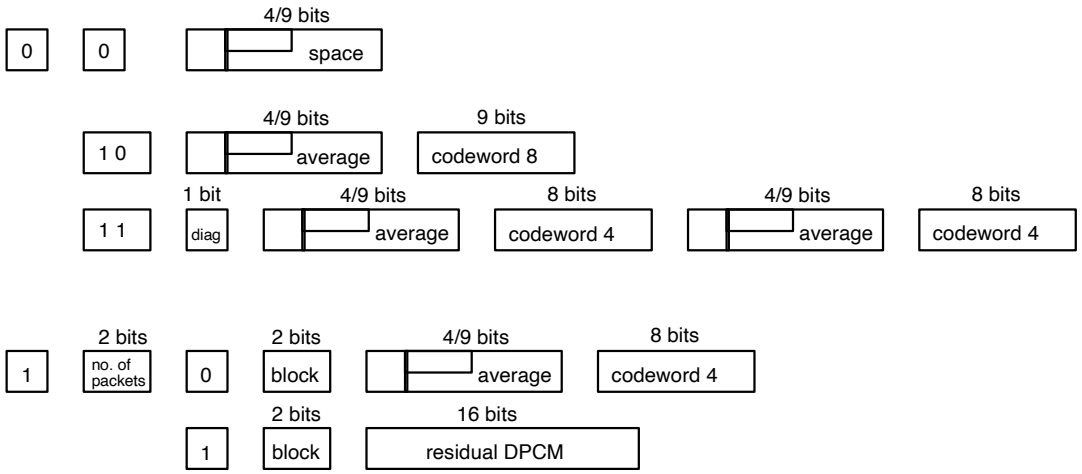
- sequence *talk4* is 16 frames long, with a resolution of 512×512 pixels. It shows relatively high movement contents for a head-and-shoulders scene. It was stored by dropping every other frame from the original sequence, in other words its total capture time was twice as long as a normal 16-frame sequence.

- sequence *hand3* is 64 frames long, with a resolution of 256×256 pixels. It has high movement content, and was designed to test background reconstruction.

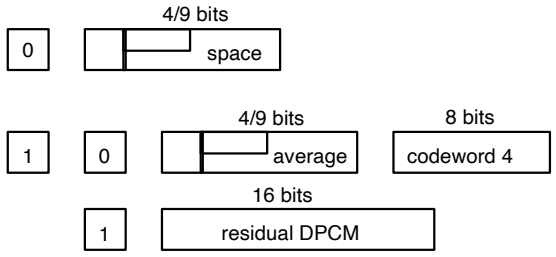
- sequence *talk2* is 64 frames long, with a resolution of 256×256 pixels. It has normal movement content for a head-and-shoulders scene.

Figure 6.4 shows the last frame from sequence *talk4*, together with samples of other frames in the sequence. Figure 6.5 shows frame 15 as reconstructed by the Block Eight coder.

Figure 6.6 shows frames 32 and 63 of sequence *hand3*. The images on the left hand side are from the original, with the ones reconstructed with Block Eight on the right.



a) Block Eight coding structure



b) Block Four coding structure

Figure 6.2: Coding structure used with Block Eight and Block Four.

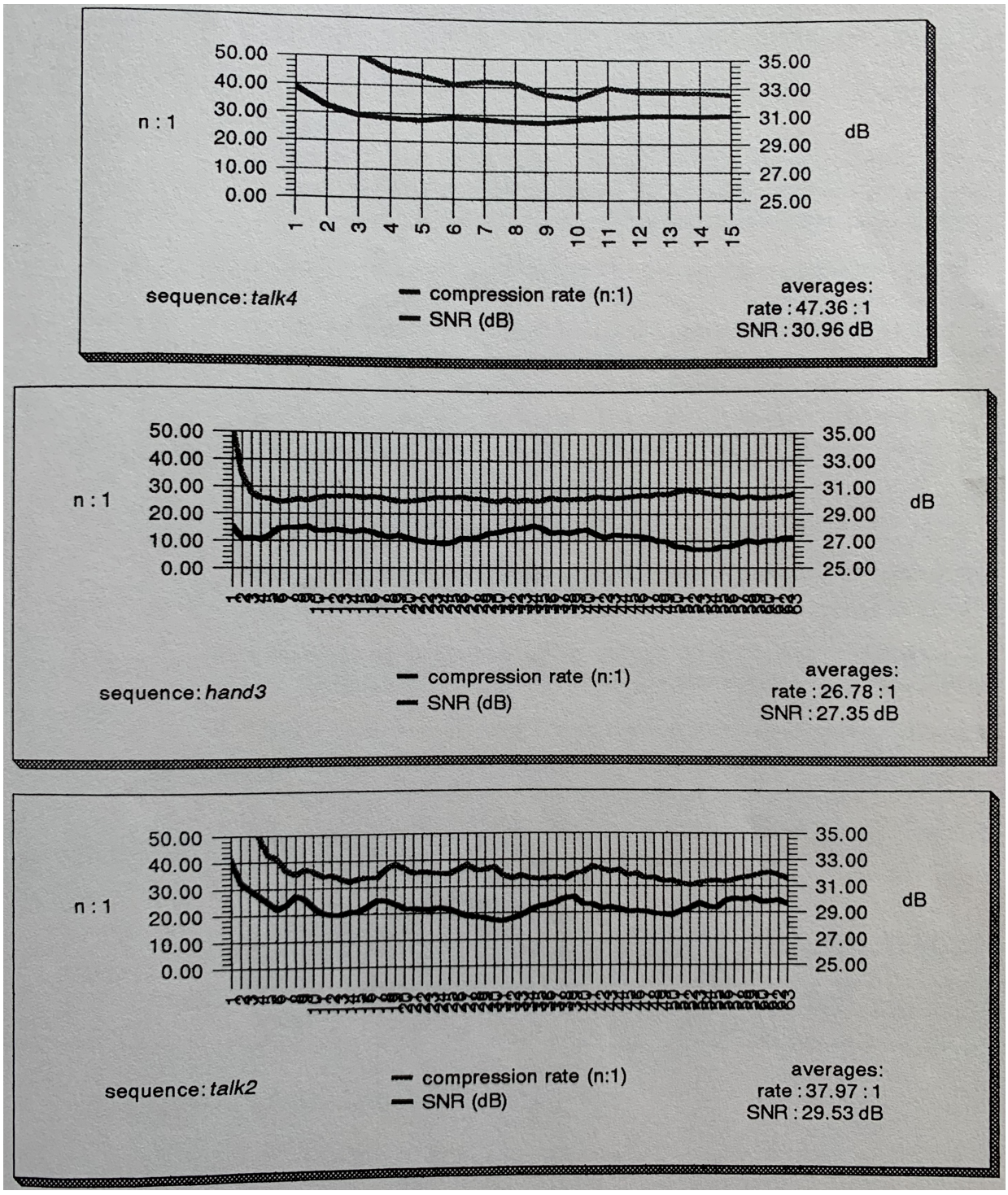


Figure 6.3: Performance of the Block Eight coder with three sequences.

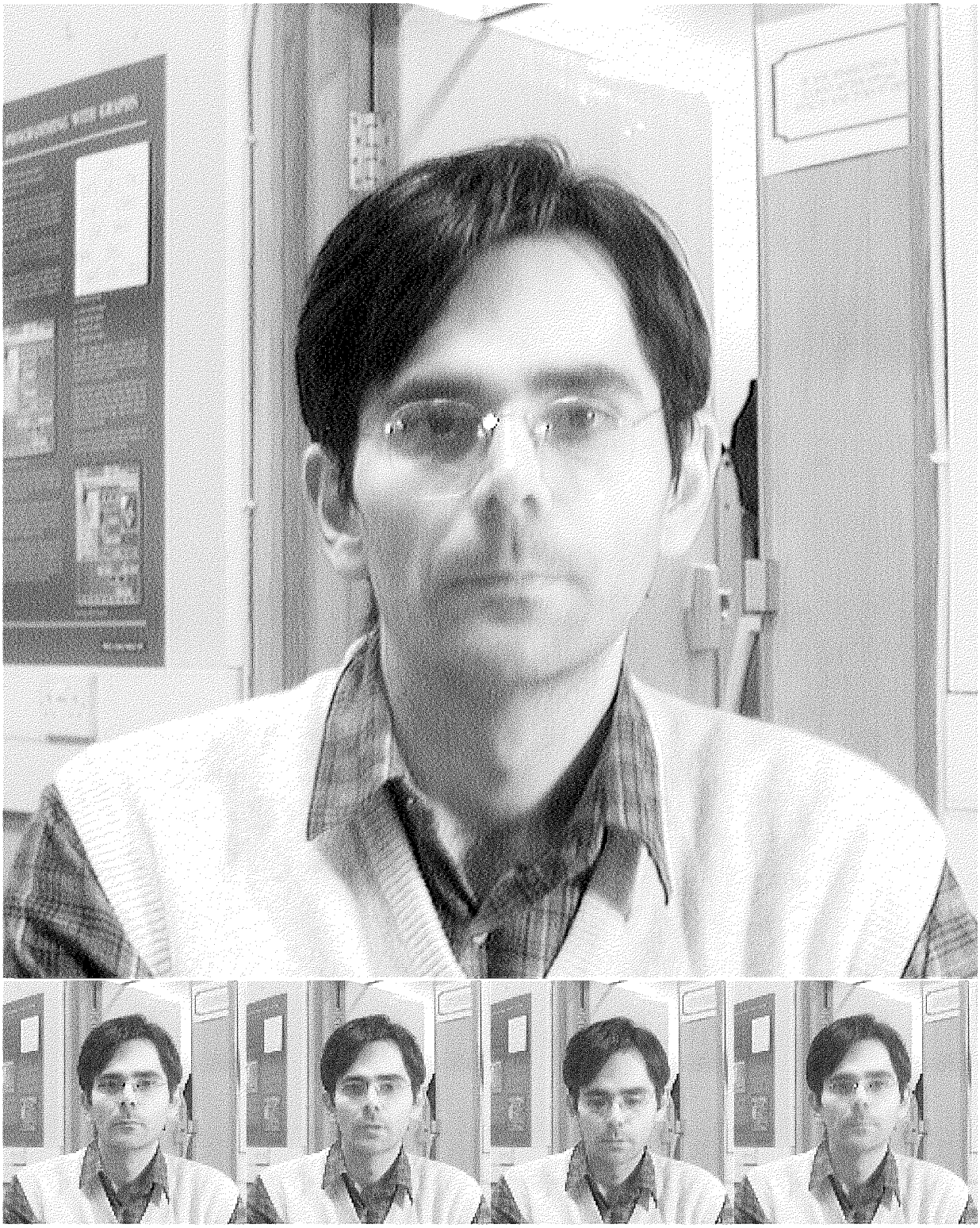


Figure 6.4: Sequence *talk4* (original), frame 15, and samples of frames 0, 5, 10 and 15.

Figure 6.5: Sequence *talk4*, frame 15, as reconstructed by Block Eight.

Figure 6.6: Sequence *hand3*, original (left) and as reconstructed by Block Eight (right).

Likewise, Figure 6.7 shows frames 32 and 63 of sequence *talk2*. The images on the left hand side are from the original, with the ones reconstructed with Block Eight on the right.

6.4.2.2 Analysis of Results

The results obtained from the Block Eight coder, summarised in Figure 6.3, show overall good performance regarding the compression rate. Typically, the compression rate can be in the range 20-50:1 (corresponding to 0.4–0.16 bits/pixel), which is a very good figure. However, the image quality is still poor, with most details in moving regions being lost. This is confirmed by the low SNR figures obtained for the 256×256 sequences: 27.4–29.5 dB.

It is clear that the 9-bit codebook used is not good enough for the purposes of this thesis, since most of the visible distortion is entirely due to operation of the vector quantizer. The following conclusions can be obtained from visual inspection of the images:

- image quality on low gradient regions is reasonably good, and the Block Eight early shape description could be used for most purposes.
- edge tracking is the cause of the overall bad quality of the images. Examining the sharper edges it can be seen that the edge is approximated, but not with enough accuracy. Many discontinuities can be seen, particularly in inclined edges. A larger codebook could do a better job here, though this may not be enough.
- The blockiness effect is much less pronounced than with the original BCS coder (Chapter 5). The effect is still quite visible, but the main reason now is not the brightness differences at the borders of an updated block, rather the broken edges betray the presence of the blocks. It should be pointed out that the process employed for image rendition on paper artificially enhances contrast and contributes to more visible block edges.

The remaining blockiness effect is not a major concern: a simple averaging algorithm operating on the borders of reconstructed blocks reduces the visibility of the block's edges significantly without adding any overhead to the coding process (the decoder keeps a copy of the original state of the blocks whose borders were

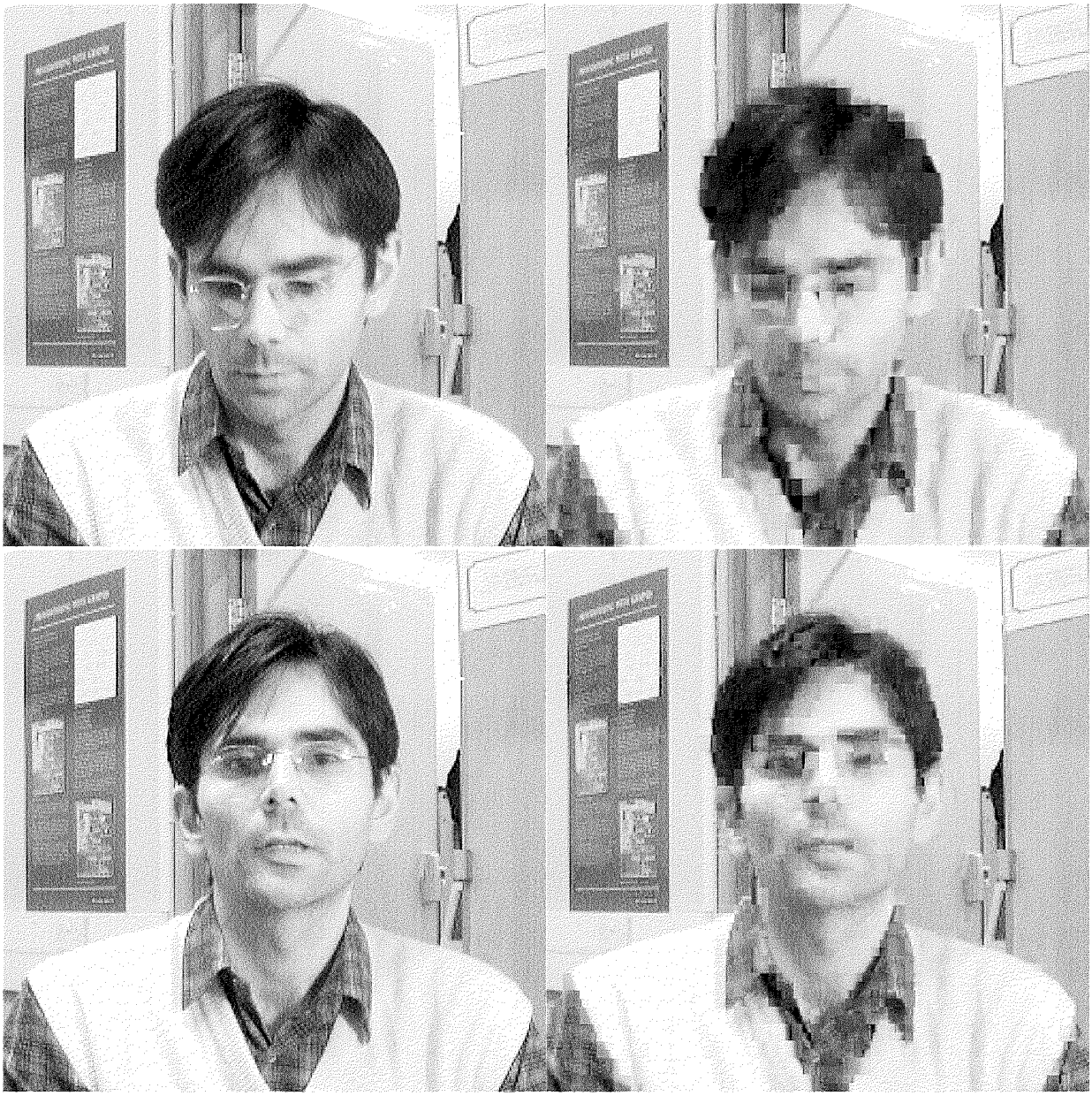


Figure 6.7: Sequence *talk2*, original (left) and as reconstructed by Block Eight (right).

altered). The signal-to-noise ratio may not get better (it could get worse), but the edge enhancing effect of the human visual system is conveniently fooled.

The progressive transmission scheme reconstructs image details fairly well, and one would ask why most details in moving regions are not present at any given frame. Examining the coding process in action reveals that the locality of movement causes a block to be flagged with movement for several frames in a row, which is the most common situation. Recalling that each time a block is flagged with movement the updating process is restarted from scratch, the replenishment abilities of the progressive transmission scheme do not have a chance to enhance the early shape description already transmitted.

It seems clear that for a block size of 8x8, the codebook used is not able to provide the required quality for the early shape description. The next strategy uses a block size of 4x4, and the benefits of better matched codebook and block sizes are clearly seen.

6.4.3 Block Four

This strategy does not attempt to exploit higher compression rates and is directed to a better image rendition. The operation of the vector quantizer is the same as in the previous strategy: the old block average, the scalar quantized average difference and the vector quantized shape description are added in a pixel basis to reconstruct each pixel value in the block.

The main features of the strategy are: i) the image is subdivided in 4x4 blocks, instead of 8x8, with the early shape description being provided by a codeword that maps 4x4 blocks; ii) subblock detection is off, in other words there is no partial block updating, as in the previous Block Eight strategy. The algorithm, in case no further movement is detected, is as follows:

1. at the moment movement is detected over the whole block, a codeword for a 4x4 block is sent.

2. in the next frame, distortion over the reconstructed block, as compared with the current block, is measured. If higher than a certain threshold (t1), a DPCM-based replenishment is performed by transmitting an array with 1 bit/pixel, as described in Section 6.4.2. The scaling factor applied to each updating bit is f1 = 6. If the distortion measure is already lower than t1,

no action is taken[3].

3. in the next frame, distortion is again compared with the current frame, but this time the comparison is more sensitive, with a smaller threshold (t2). If the distortion is higher, the 1-bit DPCM-based replenishment is done, this time with a scaling factor f2 = 3[4]. If the distortion is already lower, the block is considered up-to-date and the progressive transmission process is finished for the block.

4. the previous step is repeated till the distortion falls below the threshold (t2).

The coding structure for the Block Four strategy is depicted in Figure 6.2b, following the general model used in the Block Eight strategy.

6.4.3.1 Results

Figure 6.8 shows statistics of the Block Four coder collected from two of the sequences described in Section 6.4.2.1, *hand3* and *talk2*.

Figure 6.9 shows frames 32 and 63 of sequence *hand3*. The images on the left hand side are from the original, with the ones reconstructed with Block Four on the right.

Likewise, Figure 6.10 shows frames 32 and 63 of sequence *talk2*. The images on the left hand side are from the original, with the ones reconstructed with Block Four on the right.

6.4.3.2 Analysis of Results

The results for the Block Four coder, summarised in Figure 6.8, reveal that this coder is not able to sustain the compression rates that Block Eight does. The typical range of values to be expected for the compression rate is 10–30:1, which is still a good performance. Now, the image quality is much better than what the noise figures (29.7–30.6 dB) hint. Edges are still broken, but since the block size is much smaller, the visibility is also much more confined. More importantly, the 8-bit codebook for 4x4 blocks was much more able to reproduce features of the

[3]there is no reason why the next step could not be immediately started, in a variation of this algorithm.

[4]again, the values for the thresholds are experimental and depend on the camera noise level.

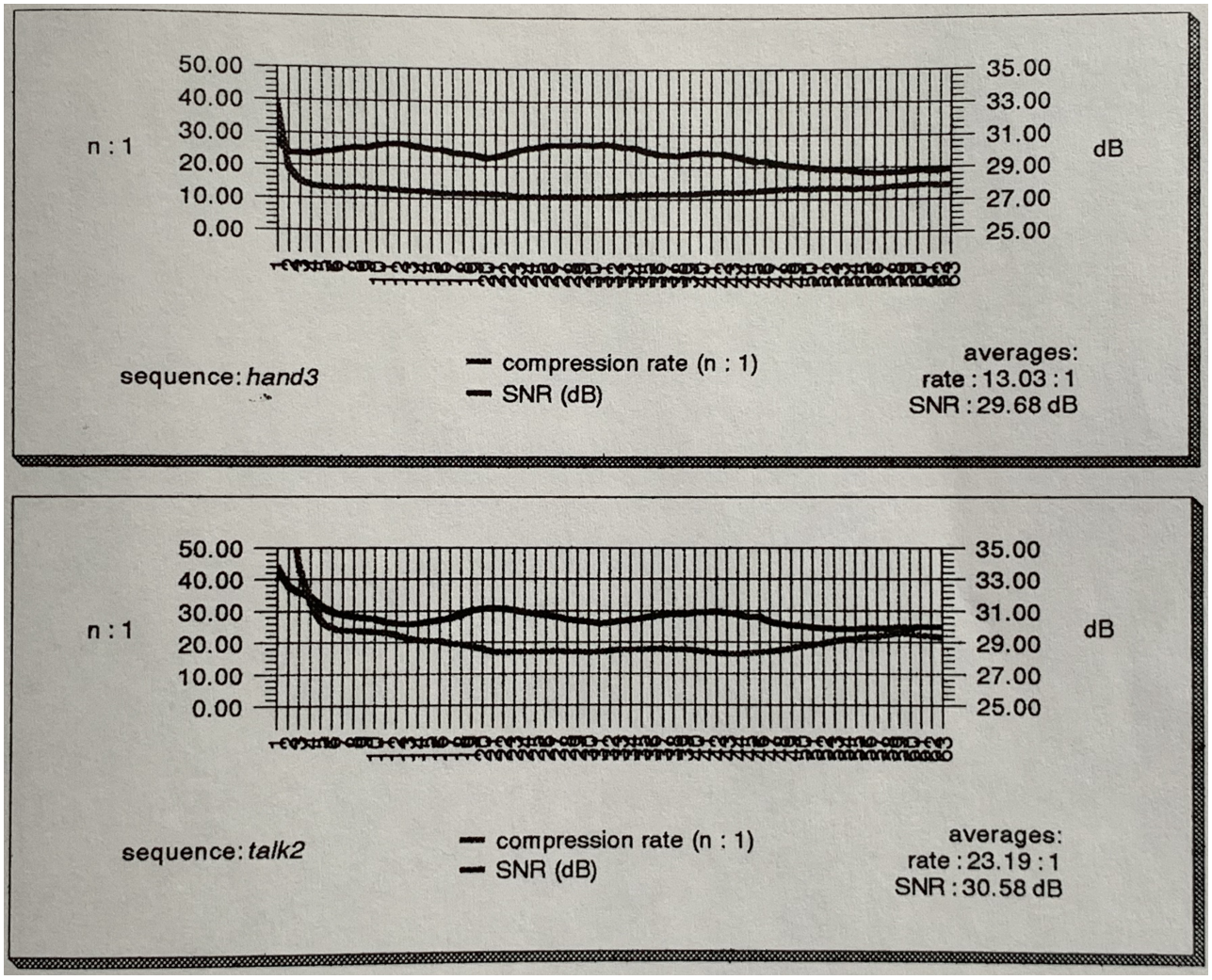


Figure 6.8: Performance of the Block Four coder with two sequences.

moving blocks, so that the overall appearance of the images is quite good, given the compression rates the sequences were subjected.

Even so, the example images show the weakness of the codebook to track edges properly. The edges are approximated, but not accurately enough, so that broken edges are common. A larger codebook tends to improve the performance, but with this particular strategy there is not much room left for extra bits. Extra bits for the codebook would drop the compression rate for the early shape description further, causing a lower overall compression rate. At levels lower than 12:1, the whole compression scheme would not be suitable for the model network chosen, the Ethernet (though, in practice, the minimum compression rate required is $\approx$

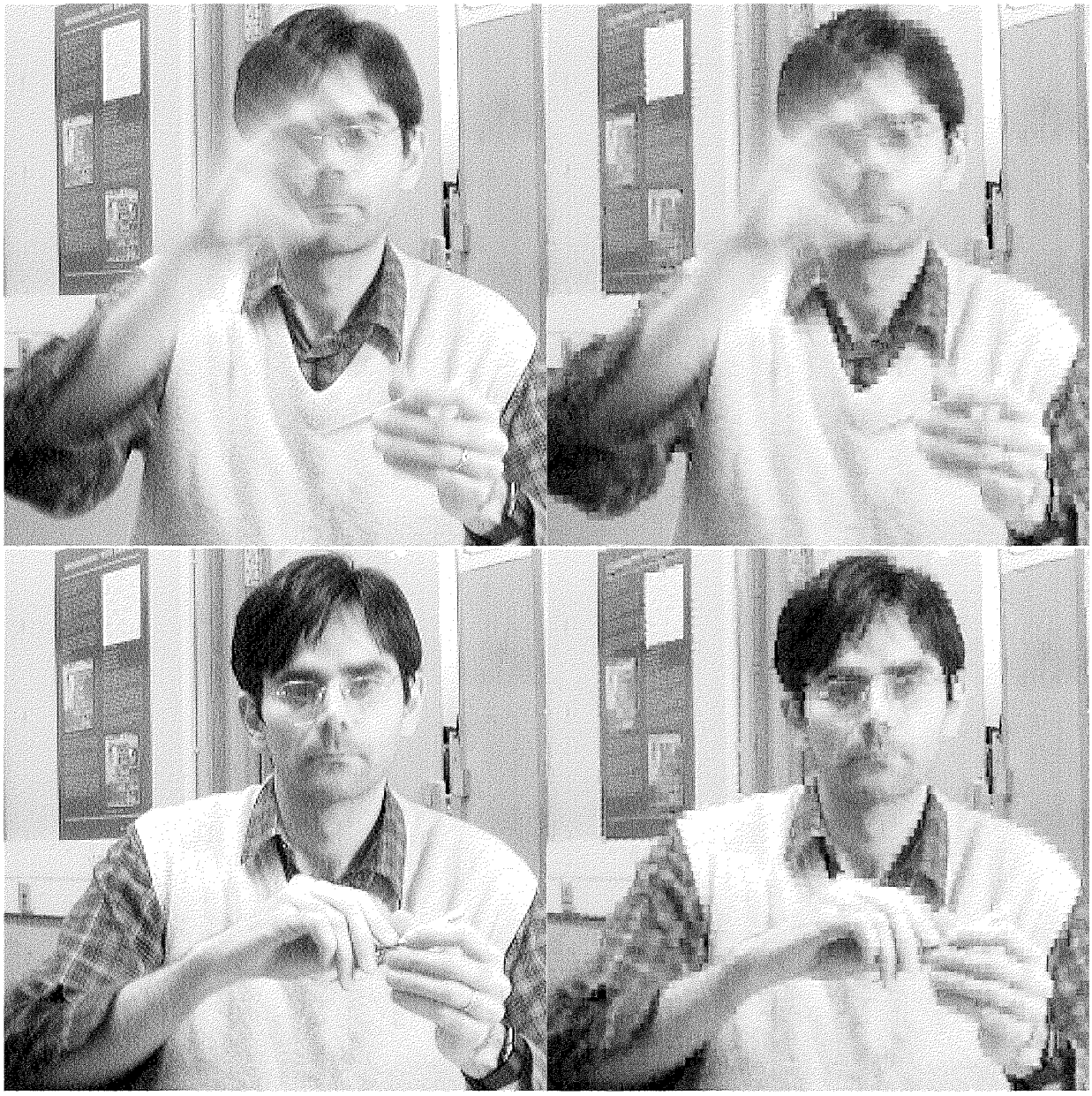


Figure 6.9: Sequence *hand3*, original (left) and as reconstructed by Block Four (right).

Figure 6.10: Sequence *talk2*, original (left) and as reconstructed by Block Four (right).

6:1).

The blockiness effect is only noticeable due to broken edges. In low gradient regions block borders are indistinguishable from the background and no special treatment needs to be added to the whole scheme to allow good reproduction.

The reconstructed sequence also performed reasonably well in the dynamic test (real-time display). This is almost guaranteed to happen by the movement detection strategy chosen, as seen in Section 5.4.2. Unless, of course, the flaws of the early shape description disrupt the image too much.

6.4.4 Comments

The Block Eight strategy provides good compression rates, but at the expense of poor quality, whereas Block Four already gives usable quality, at a compression rate that satisfies the bandwidth requirements of the model network chosen.

Some problems and advantages were detected with the MDPT/VQ scheme and are listed below:

Vector Quantization. The use of Vector Quantization for the early shape description task proved satisfactory. The performance depends heavily on the quality of the codebook used, particularly as the size of the block is enlarged.

Computational Load. It is very light, when compared with traditional video teleconferencing approaches. Since MDPT/VQ has many variables, it is difficult to establish an equation for the computational load. The best way to evaluate the performance of MDPT/VQ is by comparison with a common image processing task. A sequence of 15 frames of resolution 512×512 (approximately 40% of moving blocks in average) was coded with the Block Eight variation. In average the coding of a frame required the same as a 3x3 convolution operation performed on a full frame. With a general purpose machine of $\approx$ 4 MIPS, the algorithm took in average 10 seconds to encode a 512×512 image. This is with very little optimisation in the code and keeping the generality of the development environment. The reconstruction phase is much faster.

Codebook. The experimental codebooks used did not seem to excel in reproduction quality. The quality of a codebook can be improved by several

ways, the more immediate one being increasing the rate of the quantizer (that is, the codebook and codeword sizes). The process of building the codebook can also affect the quality of reproduction. For example, a more elaborate technique generates an optimum codebook according to the original LBG algorithm (see Section 3.6.2.5), and then designs a tree-search into the codebook. This can be done by grouping close pairs of codewords and joining them to form the next lower level codeword, and to continue working backwards till the root node is found [GC83].

Edge Tracking. This is the major obstacle for the use of VQ in early shape description. The examination of the example images in the previous sections gives the hint that enlarging the codebook alone may not be enough, since edges may occur at a large number of orientations, offsets and gradients. A small inaccuracy in an edge becomes quite visible to a human observer.

Residual Updating. The 1-bit DPCM replenishment scheme used proved satisfactory for the completion of these experiments, but it is still very inefficient in terms of the overhead it causes for the coder. More efficient approaches can no doubt be easily devised. For example, a VQ scheme specialized in coding small differential signals may provide good results, as was the original intent with the first strategy presented (Section 6.4.1).

Variable Flow. One important feature of MDPT/VQ is the way in which the output rate can be controlled to suit the temporary constraints that a local area network usually imposes. All three stages in the coding process are inherently adjustable in real-time operation and together offer a powerful match for the unpredictability of a local area network under normal use. There is, of course, a trade-off to be made: image quality should decrease as the allowed bandwidth does so.

The weakness of the codebooks used when tracking edges immediately suggest the approach called *Classified VQ* (Section 3.6.3.5). The separation of a single, large codebook, into small, specialized codebooks dedicated to a special type of image feature seems an elegant solution. Edges are sorted by orientation and gradient, for example, and mirrored patterns are detected to ensure small codebook sizes. Other types of features receive similar attention.

The trouble, however, is with the computational power required for good classification of features. An interesting proposition is to elaborate a set of feature

classes that can be more easily identified, together with the procedures required for their identification. If the proposed method is able to more accurately track edges, then many fewer broken edges will occur and the improvements in image quality might be significant.

6.5 Conclusions

The experiments with the MDPT/VQ coder showed its feasibility to overcome the severe bandwidth problems faced by the operation of a Video Viewer over local area networks.

This chapter started with a review of image compression techniques that could be used for the early shape description task. Among them, VQ and subsampling with interpolation satisfied the requirements set in the beginning of this thesis and were recommended for use with MDPT. VQ was chosen due to its more promising potential, evidenced by the literature. Simple subsampling was recommended as a less complex solution that might produce useful results for not so high compression rates.

Three variations on the basic scheme of MDPT/VQ were examined and the latter two were more deeply developed.

One of the variations tested (Block Four) was found to satisfy the compression rate requirements imposed in the beginning of this thesis, and to produce usable quality images.

Chapter 7

Conclusions

7.1 Summary

Chapter 1 introduces the concept of a Video Viewer and its possible applications. In Chapter 2 such system is analysed in more detail and requirements for its implementation are determined. Experiments with transmission of high data volume are performed to determine the range of compression rates required for proper operation of the Video Viewer in a local area network. The Ethernet is chosen as a model network implementation.

Chapter 3 presents a survey of image coding and compression techniques. Techniques are examined for possible use in the compression scheme to be devised, under the restrictions imposed in the first two chapters. Chapter 4 describes a series of early experiments with real world images. The knowledge gained from these experiments led to the development of the basic compression scheme (BCS), and ultimately to the final scheme (MDPT/VQ).

Chapter 5 introduces BCS and experimental results are reported. The identification of the scheme's weak points showed the way for future improvements of the scheme. The first such improvement concerned the movement detection process and more effective detectors are studied. Chapter 6 deals with the other major flaw of BCS, that is, the early shape description process. Candidate techniques for the task are examined and a choice is made. The final compression scheme is introduced (MDPT/VQ) and three variants of the general method are tested. The chapter ends with the conclusions drawn from the results and suggests ways of improving the operation of the MDPT/VQ coder.

7.2 Contributions of this Thesis

A number of research studies and experiments were required to elaborate this thesis. Some were directly involved in obtaining the suggested compression scheme, others were of importance at early stages of the research and set directions for the work developed later. Some of these results may be of interest to other researchers working in this field. What follows is a summary of the main issues dealt with in the course of this thesis:

Video Viewer. Some research activity is going on to provide better communication facilities as described in Chapters 1 and 2. The concept of a Video Viewer is, however, not well known in the literature. In the video teleconferencing field the main objective is the same, i.e. high image compression, but the intentions are in principle different, as is the equipment used for display. Important practical aspects for the implementation of such system are discussed in these two chapters. The Video Viewer system should allow easier communication between workstation users in a research environment, particularly those situated at a distance from each other. The facility is easily extended to a more general communication medium. However, the application that would benefit most from such a system is a Multimedia-based database, where information is stored in several different ways, ranging from text, drawings, sounds, images and video sequences.

Ethernet Throughput. These experiments were made in an early stage of the research. At that time I could find little reference to high data volume transmission via Ethernet, apart from the first papers on the Ethernet development, mainly related to voice transmission applications. On the contrary, users in general were keen to point out to me how 'slow' the Ethernet was when transferring files and doing other end-user applications. My experiments clearly demonstrated that the main bottlenecks were the machine architectures and operating software themselves, and not the network. I was able to achieve duty cycles in the high eighties percent for the particular type of transmission required by the research, and I am sure others will benefit from having knowledge of these figures. On the other hand, just recently a new implementation of TCP/IP was released in the academic community. This release has been benchmarked at full Ethernet

throughput for certain user operations, provided suitable hardware is available. This knowledge should make things easier for researchers who want to exploit Ethernet and other local area network based applications.

Survey of Image Compression Techniques. It is a thick chapter in this thesis, but even so is far from claiming completeness. However, it presents a useful summary of most common techniques. It should be useful as an introduction to the field, without being overly complex.

Frame-Freezing. Any compression scheme contemplating the use of frame freezing will benefit from the data collected in this experiment. It concerns the acceptability by users of video sequences subject to frame freezing or frame dropping (no interpolation is assumed).

Quadtree Statistics. This experiment provides interesting facts about image fragmentation for quadtree image coding. Such data is hard to find in the literature.

MDPT/VQ. This general compression method combines a number of stages to allow high image compression rates with little computational power, working with moderate noise sensor devices. Image quality under certain situations is traded-off for easier implementation, proper for a Video Viewer system. It relies on three main ideas: i) a movement detecting process that avoids most noise perturbations without any prefiltering; ii) an early image shape description based on VQ to allow for a quick update of an image block, whenever movement is detected on that block; and iii) a progressive transmission algorithm that replenishes the incomplete image information provided by the early image shape description phase, along a number of frame periods.

7.3 A View on Machine Architecture

Machine architecture implications were covered with some detail in Chapters 1 and 2. This referred mainly to how the internal architecture of a typical workstation might cope with the enormous flow of data that a video viewer generates after the image is reconstructed. This section takes a view on the processing power required for real-time implementation.

Although the processing power required by MDPT/VQ is much less than what video teleconferencing approaches based on transform coding require, the computational load is still very high. An implementation using today's technology will face some hard problems to satisfy the requirements defined at the beginning of the thesis.

The computational load is essentially integer-based, which helps in choosing a processing unit. However, most processing is done byte-wide, i.e. the CPU must make an excessive number of external references to memory whose contents are one byte wide. A good share of the processing power of 32-bit CPUs comes from the fact that data is normally wider than a byte, and is normally fetched by a single external reference. With this application the CPU wastes most of its external bandwidth abilities, or even the internal ones.

A good starting point to solve this problem is to consider a number of tightly-coupled CPUs. Provided enough communication throughput is made available, loosely-coupled designs can also perform well. This last statement almost implies the use of a very popular processing module called *Transputer*. The reason for its suggestion stems from its ability to perform serial communication at high speeds between similar devices, with no overhead onto the main processing unit. Provided the serial channels cope with the data volume that must go through them, the design can be made to use only serial communications between concurrent CPUs, which makes the hardware design a lot easier. The initial requirement of fitting the whole hardware into an expansion card for the workstation can then be complied with.

Another approach departs a little from the basic principle defined: with some custom VLSI designs, the designer could make a better use of available CPU cycles. The suggested approach resembles a VLIW architecture: since most operations performed by the coder are actually byte additions, a battery of four adders could be given external byte-wide access. When an addition is started, the set performs 4 additions at once, with a single external reference (assuming that external memory is probably 32-bit wide). It should be remembered from Section 2.2.4.1 that the speed with which a pixel must be processed is ≈ 70 ns.

7.4 Future Research

The suggested compression method left room for improvements in two of its stages, only the movement detection task seemed to function efficiently.

In the analysis of suitable techniques for the early shape description phase, it was said that both VQ and subsampling with interpolation satisfied the predefined requirements, and that transform coding was a potentially good candidate, if it were not for the high demand in processing power.

Two logical ways can be seen for future research towards improving the knowledge gained with MDPT/VQ:

- the current coder should be improved to get better looking images, whilst keeping current compression rates. This can be achieved by improving both the residual updating (from the progressive transmission process) and the VQ coder in the early shape description.

- in a parallel route, a broader characterization of the basic MDPT strategy should be done, by studying both subsampling with interpolation and transform coding, and delimiting their areas of suitable operation. It must be kept in mind that technology advances may soon make real-time transform coding easily accessible.

Bibliography

[A1089] INMOS. *INMOS Digital Signal Processing Databook*, 1989.

[AK86] A. A. Abdelwahab and S. C. Kwatra. Image data compression with vector quantization in the transform domain. In *Proc. IEEE Int. Conf. Commun.*, pages 1285–1288, 1986.

[AL87] Steven Acker and Steven Lewitt. Designing video conferencing facilities for improved eye context. *Journal of Broadcasting and Electronic Media*, (Spring issue), 1987.

[AR72] Henry L. Alder and Edward B. Roessler. *Introduction to Probability and Statistics*. W. H. Freeman and Company, San Francisco, 5 edition, 1972.

[Bal58] M. W. Baldwin. Demonstration of some visual effects of using frame storage in television transmission. *IRE Convention Record*, page 107, 1958.

[Ber81] H. C. Bergmann. Motion-adaptive interpolation of eliminated tv-fields. presented at the Picture Coding Symp., Montreal, Canada, 1981.

[Ber82] H. C. Bergmann. Displacement estimation based on the correlation of image segments. In *IEEE Proc. Int. Conf. on Electronic Image Processing*, pages 215–219, York, England, July 1982.

[Ber84] H. C. Bergmann. *Ein Schnell Konvergierendes Displacement-Schätzverfahren für die Interpolation von Fernsehbild-sequenzen*. PhD thesis, Tech. Univ. of Hannover, Hannover, Germany, February 1984.

[BG82] R. L. Baker and R. M. Gray. Image compression using non-adaptive spatial vector quantization. In *Proc. Conf. Rec. Sixteenth Asilomar Conf. Circuits, Syst., Comput.*, pages 55–61, October 1982.

[BG83] R. L. Baker and R. M. Gray. Differential vector quantization of achromatic imagery. In *Proc. of the International Picture Coding Symposium*, March 1983.

[BGGM80] A. Buzo, A. H. Gray, Jr., R. M. Gray, and J. D. Markel. Speech coding based upon vector quantization. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-28:562–574, August 1980.

[Cho71] M. C. Chow. Variable-length redundancy removal coders for differentially coded video telephone signals. *IEEE Trans. Commun. Tech.*, COM-19(6):922–926, December 1971.

[Com87] Douglas Comer. *Operating System Design - Volume II: Internetworking with Xinu.* Prentice-Hall, New Jersey, 1987.

[CP84] Wen-Hsiung Chen and William K. Pratt. Scene adaptive coder. *IEEE Trans. Communications*, 32(3):225–232, March 1984.

[CR83] C. Cafforio and F. Rocca. The differential method for image motion estimation. In T. S. Huang, editor, *Image Sequence Processing and Dynamic Scene Analysis*, pages 104–124. Springer-Verlag, Berlin, Germany, 1983.

[CS77] W. H. Chen and C. N. Smith. Adaptive coding of monochrome and color images. *IEEE Trans. Commun.*, COM-25:1285–1292, November 1977.

[CY83] Roland T. Chin and Chia-Lung Yeh. Quantitative evaluation of some edge-preserving noise-smoothing techniques. *Comp. Vision, Graphics and Image Process.*, 23(1):67–91, July 1983.

[DM79] Edward J. Delp and O. Robert Mitchell. Image compression using block truncation coding. *IEEE Trans. Communications*, COM-27(9):1335–1342, September 1979.

[DR78] L. S. Davis and A. Rosenfeld. Noise cleaning by iterated local averaging. *IEEE Trans. on Syst., Man and Cybernetics*, SMC-8:705–710, 1978.

[Dre87] Howard M. Dreizen. Content-driven progressive transmission of grey-scale images. *IEEE Trans. Communications*, COM-35:289–296, March 1987.

[Dub85] Eric Dubois. The sampling and reconstruction of time-varying imagery with application in video systems. *Proceedings of the IEEE*, 73(4):502–522, April 1985.

[Eth80] Version 1.0, Digital Equipment Corporation, Intel, Xerox. *The Ethernet, a Local Area Network: Data Link Layer and Physical Layer Specifications*, September 1980.

[FWC84] J. D. Foley, V. L. Wallace, and P. Chan. The human factors of computer graphics interaction techniques. *IEEE Comp. Graphics and Appl.*, 4(11):13–48, Nov 1984.

[Gar82] Irene Gargantini. An effective way to represent quadtrees. *Comm. ACM*, 25(12):905–910, December 1982.

[GC83] A. Gersho and D. Cheng. Fast nearest neighbor search for nonstructured euclidean codes. In *Abstracts of the 1983 IEEE International Symposium on Information Theory*, page 88, September 1983.

[Ger82] Allen Gersho. On the structure of vector quantizers. *IEEE Trans. on Information Theory*, IT-28(2):157–166, March 1982.

[Gim75] T. I. Gimlett. Use of activity classes in adaptive transform image coding. *IEEE Trans. Commun.*, COM-23:785–786, July 1975.

[Gon83] T. A. Gonsalves. Packet-voice communication on an ethernet local computer network: An experimental study. *ACM Comp. Comm. Rev.*, 13(2):178–185, 1983.

[GR82] A. Gersho and B. Ramamurthi. Image coding using vector quantization. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 428–431, April 1982.

[Gra84] Robert M. Gray. Vector quantization. *IEEE ASSP Magazine*, pages 4–29, April 1984.

[GS86] M. Goldberg and H. F. Sun. Image sequence coding using vector quantization. *IEEE Trans. Communications*, COM-34:703–710, July 1986.

[GW77] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing.* Addison-Wesley, Reading, Massachusetts, 1977.

[GY85] A. Gersho and M. Yano. Adaptive vector quantization by progressive code-vector replacement. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 133–136, 1985.

[Har52] C. W. Harrison. Experiments with linear prediction in television. *Bell Syst. Tech. J.*, 31(4):746–783, July 1952.

[HM87] D. Hutchison and M. Merabti. Ethernet for real-time applications. *IEE Proceedings - E*, 134(1):47–53, January 1987.

[HR85] M. Haqhiri and C. Renius. An adaptive block-quantizer approach to transform coding of pictures. *SPIE*, 594:66–71, 1985. Image Coding.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. IRE*, 40(9):1098–1101, September 1952.

[HW85] H. M. Hang and J. W. Woods. Predictive vector quantization of images. *IEEE Trans. Communications*, COM-33:1208–1219, November 1985.

[I. 88] I. Greif *et al. Computer Supported Cooperative Work: a Book of Reading.* Morgan Kaufmann Inc., San Mateo, California, 1988.

[III+77] T. Ishiguro, K. Iinuma, Y. Iijima, T. Koga, S. Azami, and T. Mune. Composite interframe coding of ntsc color television signals. In *Proc. Int. Communications Conf.*, pages 6.4–1–6.4–5, 1977.

[int85] Peripheral Volume II, Intel Corporation, Santa Clara, CA. *Microsystem Components Handbook*, 1985.

[Jag52] F. De Jager. Deltamodulation: A method of pcm transmission using a one-unit code. Technical report, Philips Res. Rep., 1952.

[Jai81] Anil K. Jain. Image data compression: A review. *Proc. IEEE*, 69(3):349–389, March 1981.

[JG82] B.-H. Juang and A. H. Gray, Jr. Multiple stage vector quantization for speech coding. In *Proc. IEEE Int. Conference on Acoustics Speech and Signal Processing 1*, pages 597–600, April 1982.

[JJ81] J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. *IEEE Trans. Communications*, COM-29:1799–1806, December 1981.

[K. 77] K. Iinuma *et al.* Netec-6: Interframe encoder for color television signals. *NEC Res. Devel.*, (44):92–96, January 1977.

[KEM83] Eiji Kawaguchi, Tsutomu Endo, and Jun-Ichi Matsunaga. Depth-first picture expression viewed from digital picture processing. *IEEE Trans. PAMI*, 5(4):373–384, July 1983.

[KIH+81] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion-compensated interframe coding for video conferencing. In *NTC 81, Proc.*, pages G5.3.1–G5.3.5, New Orleans, LA, December 1981.

[KIK85] Murat Kunt, Athanassios Ikonomopoulos, and Michel Kocher. Second-generation image-coding techniques. *Proc. IEEE*, 73(4):549–574, April 1985.

[KK88] K. L. Kraemer and J. L. King. Computer-based systems for cooperative work and group decision making. *ACM Computing Surveys*, 20(2):115–146, June 1988.

[LBG80] Yoseph Linde, Andrés Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Communications*, COM-28(1):84–95, January 1980.

[Lee83] Jong-Sen Lee. Digital image smoothing and the sigma filter. *Comp. Vision, Graphics and Image Process.*, 24:255–269, 1983.

[LH87] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–296, September 1987.

[Llo82] S. P. Lloyd. Least-squares quantization in pcm. *IEEE Trans. Inform. Theory*, IT-28:129–137, March 1982.

[LMKG85] Jean Paul Lauzon, David M. Mark, Lawrence Kikuchi, and J. Armando Guevara. Two-dimensional run-encoding for quadtree representation. *Comp. Vision, Graphics and Image Process.*, 30:56–59, 1985.

[MAIY84] T. Murakami, K. Asai, A. Itoh, and E. Yamazaki. Interframe vector coding of color video signals. In *Proc. Int. Picture Coding Symp.*, July 1984.

[Max60] J. Max. Quantizing for minimum distortion. *IEEE Trans. Inform. Theory*, IT-6:7–12, March 1960.

[Mey87] Mike Meyer. Disk timings. USENET Newsgroup Comp.sys.amiga (informal), May 1987.

[Miy75] M. Miyahara. Analysis of perception of motion in television signals and its application to bandwidth compression. *IEEE Trans. Communications*, COM-23:761–766, July 1975.

[ML86] J. P. Mareseq and C. Labit. Vector quantization in transformed image coding. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 145–147, April 1986.

[Mor66] G. M. Morton. A computer oriented geodetic data base, and a new technique in file sequencing. Technical report, IBM Canada Ltd., March 1966.

[Mou69] F. W. Mounts. A video encoding system with conditional picture-element replenishment. *Bell Syst. Tech. J.*, 48:2545–2554, September 1969.

[MPG85] Hans Georg Musmann, Peter Pirsch, and Hans-Joachim Grallert. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, April 1985.

[Nas85] N. M. Nasrabadi. Use of vector quantizers in image coding. In *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing*, pages 125–128, March 1985.

[NB82] G. J. Nutt and D. L. Bayer. Performance of csma/cd networks under combined voice and data loads. *IEEE Trans. on Communications*, 30(1):6–11, January 1982.

[NHN87] Ari Nieminen, Pekka Heinonen, and Yrjö Neuvo. A new class of detail-preserving filters for image processing. *IEEE Trans. PAMI*, PAMI-9(1):74–90, January 1987.

[NK83] N. M. Nasrabadi and R. A. King. Image coding using vector quantization in the transform domain. *Pattern Recognition Letters*, pages 323–329, 1983.

[NK88] Nasser M. Nasrabadi and Robert A. King. Image coding using vector quantization: A review. *IEEE Trans. Communications*, 36(8):957–971, August 1988.

[NL80] Arun N. Netravali and John O. Limb. Picture coding: A review. *Proc. IEEE*, 68(3):366–406, March 1980.

[NPM77] A. N. Netravali, B. Prasada, and F. W. Mounts. Some experiments in adaptive and predictive hadamard transform coding of pictures. *Bell Syst. Tech. J.*, 56:1531–1547, October 1977.

[NR79] Arun N. Netravali and J. D. Robbins. Motion-compensated television coding: Part I. *Bell Syst. Tech. J.*, 58(3):631–670, March 1979.

[NR81] Arun N. Netravali and J. D. Robbins. Motion-adaptive interpolation of television frames. presented at the Picture Coding Symp., Montreal, Canada, 1981.

[O'N66] J. B. O'Neal, Jr. Predictive quantizing systems (differential pulse code modulation) for the transmission of television signals. *Bell Syst. Tech. J.*, 45:689–721, May-June 1966.

[PCB76] I. M. Paz, G. C. Collins, and B. H. Batson. A tri-state delta modulator for run-length encoding of video. In *Proc. Nat. Telecomm. Conf.*, volume I, pages 6.3–1–6.3–6, Dallas, TX, November 1976.

[Pik85] Rob Pike. Graphics in overlapping bitmap layers. *ACM Trans. Graph.*, 2(2):135–160, April 1985.

[Pos82] Jonathan B. Postel. Internetwork protocol approaches. In Paul E. Green, Jr., editor, *Computer Network Architectures and Protocols*, pages 511–526. Plenum Press, New York, 1982.

[Pra78] William K. Pratt. *Digital Image Processing.* John Wiley & Sons, New York, 1978.

[Rob62] L. G. Roberts. Picture coding using pseudo-random noise. *IRE Trans. Inform. Theory*, IT-8:145–154, February 1962.

[Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[SB65] A. J. Seyler and Z. L. Budrikis. Detail perception after scene changes in television image presentations. *IEEE Trans. Inform. Theory*, IT-11:31–43, January 1965.

[SB66] J. W. Schwartz and R. C. Baker. Bit-plane encoding: A technique for source encoding. *IEEE Trans. Aerospace Electron. Syst.*, AES-2(4):385–392, July 1966.

[SC80] George W. Snedecor and William G. Cochran. *Statistical Methods.* The Iowa State University Press, 7 edition, 1980.

[SDRC82] John F. Shoch, Yogen K. Dalal, David D. Redell, and Ronald C. Crane. Evolution of the ethernet local computer network. *Computer*, 15(8):10–27, August 1982.

[SG84] M. J. Sabin and R. M. Gray. Product code vector quantizers for waveform and voice coding. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-32:474–488, June 1984.

[SGS71] C. L. Song, J. Garodnick, and D. L. Schilling. A variable step-size robust delta modulator. *IEEE Trans. Commun.*, COM-19:1033–1044, December 1971.

[SH69] D. R. Spencer and T. Huang. Bit-plane encoding of continuous-tone pictures. In *Symposium on Computer Processing in Communications*, New York, April 1969. Polytechnic Institute of Brooklyn.

[SHT72] W. F. Schreiber, T. S. Huang, and O. J. Tretiak. Contour coding of images. In Thomas S. Huang and O. J. Tretiak, editors, *Picture Bandwidth Compression*, pages 443–448. Gordon and Breach, New York, 1972.

[SIK+82] D. C. Smith, C. Irby, R. Kimball, W. Verplank, and E. Harslem. Designing the star user interface. *Byte*, 7(4):242–282, April 1982.

[SP85] Shaker Sabri and Birendra Prasada. Video conferencing systems. *Proceedings of the IEEE*, 73(4):671–688, April 1985.

[SR84] R. Srinivasan and K. R. Rao. Predictive coding based on efficient motion estimation. In *ICC 1984, Proc.*, pages 521–526, May 1984.

[STA+86] T. Saito, H. Takeo, K. Aizawa, H. Harashima, and H. Miyakawa. Adaptive discrete cosine transform image coding using gain/shape vector quantization. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages 129–132, April 1986.

[Ste87] Jeff Stearns. Disk-based vs. diskless workstations performance measurements. USENET Newsgroup Comp.unix.wizards (informal), April 1987.

[SW49] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[TW71] M. Tasto and P. A. Wintz. Image coding by adaptive block quantization. *IEEE Trans. Commun. Technol.*, COM-19:957–971, December 1971.

[Uli87] Robert Ulichney. *Digital Halftoning.* The MIT Press, Cambridge, Massachusetts, 1987.

[UNI85] *UNIX Programmer's Manual*, 1985. COMPRESS(1), manual page.

[UR87] Vishwas R. Udpikar and Jewan P. Raina. BTC image coding using vector quantization. *IEEE Trans. Communications*, COM-35(3):352–356, March 1987.

[VG87] D. J. Vaisey and A. Gersho. Variable block-size image coding. In *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing*, pages 1051–1054, April 1987.

[Wat85] Gregory C. A. Watson. A flexible graphics subsystem. Master's thesis, Department of Computer Science, University of Manchester, Manchester, October 1985.

[WH84] Zhongde Wang and B. R. Hunt. Comparative performance of two different versions of the discrete cosine transform. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-32:450–453, April 1984.

[YS77] J. K. Yan and D. J. Sakrison. Encoding of images based on two-component source model. *IEEE Trans. Commun.*, COM-25:1315–1323, November 1977.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, May 1977.