Technical Report

UCAM-CL-TR-207
ISSN 1476-2986

Number 207

# Video replay in computer animation

Stuart Philip Hawkins

October 1990

This technical report is based on a dissertation submitted December 1989 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Queens' College.

ISSN 1476-2986

# Contents

# List of Figures

# List of Tables

# List of Plates

# Preface

This dissertation presents a design for an animation system that supports video-rate replay of frame sequences within a frame buffer based graphics architecture.

In recent years frame buffer architectures have become dominant, largely displacing other forms of graphics display system. But a frame buffer representation is not well suited to the support of animation. In particular, two main problems are faced: (1) the generation of each new frame within a single frame time (typically 40ms); and (2) the updating of the frame buffer with the new frame representation, also within one frame time. Both these problems stem from the fact that the large amount of data required to represent each frame has to be processed within a strictly limited time. The difficulty with updating the frame buffer representation has been largely addressed by the development of powerful new display processor architectures, made possible by improvements in semiconductor technology. The *generation* of frames at replay rates, however, represents a much greater challenge and there are numerous situations for which real time animation is simply impracticable. In such cases an alternative approach is that of frame-by-frame animation in which the frame sequence is pre-calculated off-line and stored for later replay at the correct speed. This technique is commonly referred to as real-time playback.

In this dissertation the requirements for real-time playback are discussed and a number of distinct approaches to the design of such systems identified. For each approach examples of previous real-time playback systems are examined and their individual shortcomings noted. In light of these observations the design of a new hardware-based animation system is proposed and its implementation described. In this system frames are stored digitally and image compression is used to address the non-video-rate transfer rate and storage capacity limitations of the frame storage device employed (an unmodified 5¼ inch magnetic disc drive). Such an approach has previously received little attention. Frame sequences are stored on the disc in a compressed form and during replay are de-compressed in real-time using a hardware implementation of the coding algorithm. A variety of image compression strategies are supported within a generalised coding framework. This introduces operational flexibility by allowing the system to be tailored according to the needs of a particular application.

# Chapter 1

# Computer Animation Systems

## 1.1 What is Animation?

The first picture animation system was invented in 1831 by a Frenchman named Joseph Antoine Plateau. This device, which he called the Phenakistoscope, consisted of a sequence of drawings fixed to a spinning disc that could be viewed through a small window framing the moving drawings. The Phenakistoscope idea was refined by Horner with his invention of the Zoetrope in which drawings were held on the inside of a spinning drum and viewed through regularly spaced slits around the drum's periphery. This idea was further refined by Reynaud who invented the Praxinoscope. Here the slits were replaced by mirrors spinning in the centre of the drum. Animation in its modern form emerged around the turn of the century when sequences of drawings were photographed onto movie film. It was the development of these first animated films which assured the future of animation and the continual development of techniques until the present day.

In all these cases the underlying principle is the same: the illusion of smooth and continuous change is created by rapidly presenting a sequence of images to the observer, where each image is a slight alteration of its predecessor. In fact, this description serves as a reasonably comprehensive definition of animation. Note the use of the word "change" in this definition to emphasise that animation is not restricted to movement alone but also includes possible variation of scene illumination, object colour, and shape—animation can exist without motion.

Perhaps the most readily identifiable type of animation is the two-dimensional animated cartoon of the form produced by the Disney studios and other production companies. However, animation is a general purpose technique that is widely used in other areas. Apart from cinema, two major users of animation are the television industry and educational and research establishments. Animation is widely used in the television industry for the production of title sequences, logos and programme inserts, and its use in the production of television commercials is increasing. In the fields of education and research the visualisation and understanding of complex behaviours is greatly aided by the use of animation through its introduction of an extra (temporal) dimension.

In the *conventional animation* process each of the hundreds or thousands of

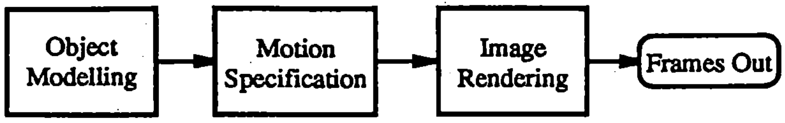


Figure 1.1: The animation pipeline

frames which go to make up a typical animated sequence must be individually drawn and coloured by hand. Finished drawings are photographed onto film one frame at a time. When the sequence is complete the animation can be viewed by replaying the film at the proper rate on an ordinary film projector. This manual process is both slow and tedious, and animation produced in this way is expensive because of the large number of people required. An alternative approach is that of *computer animation* in which a computer is used for the production of sequences and/or their replay. This raises the question of what exactly is meant by computer animation.

## 1.2 Computer Animation

As pointed out by Magnenat-Thalmann and Thalmann [Magn85*a*] the term "computer animation" is imprecise and may be interpreted in a number of ways. A common first step in classifying computer animation systems is to distinguish between *computer-assisted* animation and *computer-modelled* animation.

Computer-assisted animation refers to the use of computers to assist with various stages of the conventional animation process. This has been achieved with varying degrees of success. The introduction of paint systems and graphical editors has greatly increased the speed and accuracy with which drawings can be produced. Facilities are provided for the input of drawings (either freehand via a graphics tablet or from external sources via a scanner), colouring (via some form of area filling function), composition and image storage/retrieval. Unfortunately, such systems do little to reduce the overall time taken to produce an animated film as they do not reduce the total number of frames that need to be drawn and, therefore, the contribution required from the animators. For this assistance must be provided for the task of *in-betweening*, that is, the generation of intermediate frames from the *key frames* produced by an animator. Unfortunately, attempts at automating this process have proved to be less than successful due to the difficulty of the task [Catm78].

Computer-modelled animation is more interesting because here the computer assumes a fundamental role in the animation process, rather than simply providing assistance for a human animator. Modelled animation is oriented towards the production of 3-d animated sequences (contrasting with the mainly 2-d emphasis of assisted animation) and involves three main activities [Magn85*a*]: (1) object modelling; (2) motion specification and synchronisation; and (3) image rendering. This process is illustrated in Figure 1.1.

Object modelling refers to the process of defining and constructing the 3-d objects which form the scene to be animated. In the simplest case these are defined as wireframe models. For greater realism solid models are employed, the primitives for which fall into three categories: polygonal meshes, algebraic surfaces and surface patches. These techniques are drawn directly from the mainstream of computer graphics and are described in detail in many of the standard graphics texts (e.g., [Newm81,Fole82,Hear86]).

Motion specification and synchronisation is concerned with the movement of objects within the 3-d world space in order to achieve the desired animation. In specifying the required motion an analogy is often drawn with the equivalent process of controlling the action in a live-action film. A *script* is used which allows the animator, acting in the role of director, to manipulate the various objects in the scene and to control the positions of one or more virtual cameras and light sources. The virtual cameras and light sources can be manipulated in exactly the same way as the objects being animated in order to create different viewpoints, panning and scrolling effects, lighting changes, etc. Scripting systems have evolved from ordinary programming languages, particularly those with features well suited to the requirements of animation (e.g., the class mechanism found in SIMULA and in later object-oriented languages such as SMALLTALK). The abstraction provided by such mechanisms greatly assists in simplifying the specification and control of complex animation. More recently effort has been concentrated in defining special-purpose extensions to programming languages and in the development of completely new animation languages. Examples of such systems are discussed in detail in [Magn85*a*]. Most research into modelled animation has concentrated upon the motion specification and synchronisation aspect (as it is this which distinguishes animation from the mainstream of computer graphics) and a comprehensive bibliography of the work done in this area has been published by Magnenat-Thalmann *et al* [Magn85*b*].

An alternative way to specify motion is through the use of *simulation*. Here objects are manipulated according to a group of rules that are intended to accurately model some physical reality. The use of simulation for the production of animated sequences has traditionally received little attention in the computer animation field. Simulation is widely used in the engineering and scientific disciplines for applications as diverse as chemical modelling and stress analysis; if the results of a simulation can be produced graphically then a time-varying sequence of such results can provide a powerful way of understanding complex behaviours. Animation produced via scripts and programmed control, on the other hand, is mainly geared towards the entertainment field. Another distinction which might be made is that simulation aims to reflect reality accurately, whereas the aim of scripted animation is often the exact opposite! These disparate approaches were drawn together in the work of Pullen [Pull87] who investigated ways in which simulation could be used as a general-purpose motion specification technique for modelled animation. In his work simulation provided another tool for the animator, complementing the use of scripts or programmed control. The animator uses simulation as a powerful mechanism for automatically generating frame sequences, but is given the opportunity to override the behaviour of this mechanism at any

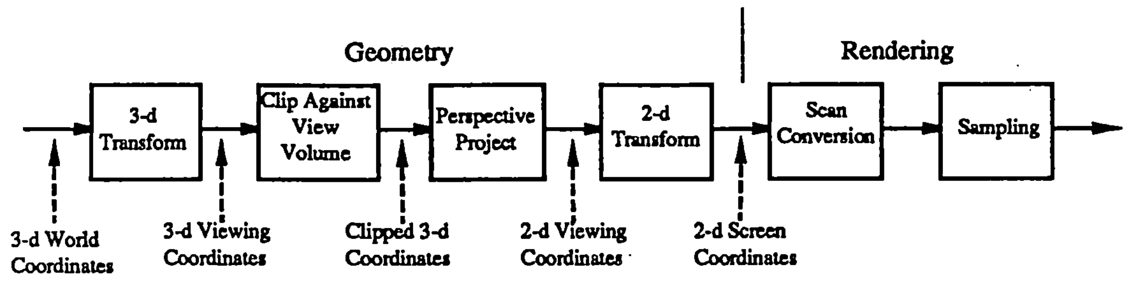


Figure 1.2: A typical rendering pipeline

time.

In the final activity of the modelled-animation process, image rendering, a 2-d image representation is produced from the 3-d model. As for object modelling most of the techniques used are standard ones drawn from the mainstream of computer graphics. The main techniques are discussed in detail in [Magn87], as well as standard graphics texts (e.g., [Newm81,Fole82,Roge85]). The details of the processing involved depend upon the technique employed, but a typical rendering pipeline is illustrated in Figure 1.2. The purpose of this example is to illustrate the complexity of the task involved. Typical computational costs of such a pipeline are discussed in [Akel88]. This is of particular relevance if the rendering is to be performed at speed (i.e., at viewing rates)—a point returned to shortly.

## 1.3 Computer Graphics Technology

By way of introduction to computer animation systems it is first necessary to review some of the relevant technology used in the encompassing field of computer graphics.

The only viable forms of display device for dynamic graphics are those based upon the cathode ray tube (CRT). CRT displays fall into two categories according to the way in which the image is refreshed. In a *raster scan* display the display area is divided into a number of horizontal scanlines and the image refreshed in a fixed (scanline) order starting at the top left and ending at the bottom right of the screen. The image produced during this cycle is termed a *frame* and the whole refresh process is repeated (typically) 25 times a second. Each scanline is divided into a number of *pixels* (where this number is usually related to the number of scanlines and chosen to make the pixels square). A pixel represents the smallest accessible unit on the display and a typical display for graphics use has between 1/4 million and 1 million pixels. In a *vector scan* display (also known as *random* or *calligraphic* scan), on the other hand, the refresh order is not fixed but is directly determined by the image being displayed. In such displays an image is composed of a number of vectors and each vector is generated in turn by directly controlling the deflection of the CRT's electron beam in $x$ and $y$ over time.

The main elements of a raster scan display system are illustrated in Figure 1.3. The frame buffer is a two-dimensional memory array which stores numerical values

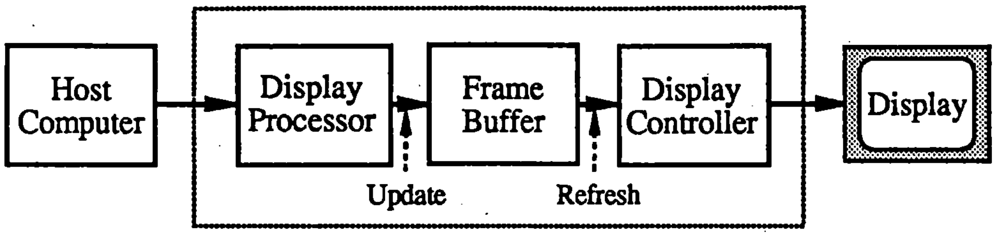


Figure 1.3: Principal elements of a frame buffer architecture

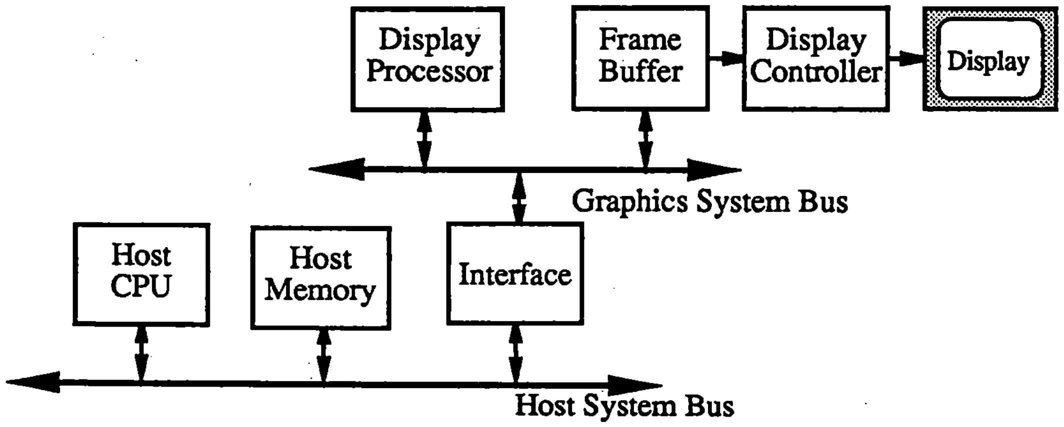


Figure 1.4: Typical raster scan display architecture

corresponding to the required intensities of the pixels on the screen. Generally there is a one to one mapping between the position of a screen pixel and the position of its value in the frame buffer. The display controller reads the frame buffer in raster order in synchrony with the electron beam scanning the CRT and the values read are translated to corresponding pixel intensities via a digital to analogue converter.

The display processor is responsible for translating higher level instructions from the host computer into the lower level representation of the frame buffer. The complexity of this processor varies considerably from system to system. In the simplest case it only provides a mechanism for setting the values of individual pixels, and all other graphical processing must be done by the host. As the complexity of the graphics processor is increased, more and more of these operations can be off-loaded from the host and executed in the hardware of the processor. Typical operations supported include scan conversion (conversion of higher level line, polygon, and curve descriptions into the corresponding lower level frame buffer representation), area filling and clipping.

Figure 1.4 illustrates a typical architecture for a medium performance raster scan graphics system.

The arrangement of a vector scan display system is similar to that of Figure 1.3 except the frame buffer is replaced by a *display file* containing vector descriptions.

The image is dynamically generated by the display controller which continuously cycles through the display file and directly draws a line on the screen for each vector description read. The display file is a higher level representation than the frame buffer and changes to the display are easily and rapidly made; any change to a vector description in the display file is instantly reflected on the screen during the next refresh cycle. Further, many vector displays support the use of *segmented display files* in which vectors can be grouped together and hierarchical image descriptions produced. This introduces even more power by allowing easy and rapid manipulation of compound elements. The result of this is that dynamic graphics are easily achieved and this explains the historical widespread adoption of vector scan displays in engineering, computer-aided design (CAD) and animation applications. Indeed, until comparatively recently this was the only effective way to achieve animation on a computer system. The principal disadvantages of this technology are the restriction to wireframe images only (no filled areas) and the fact that only a limited number of vectors can be maintained (i.e., refreshed) simultaneously (which limits the achievable image complexity).

In contrast, the lower level representation of the frame buffer means that a raster scan display is intrinsically harder to update than a vector scan display. In particular, vectors and other graphical primitives must be scan converted to produce a raster representation. Consequently raster scan displays are less suited to the support of animation. However, a frame buffer representation supports filled areas (including smooth shading) and imposes no limit on image complexity. In addition, more powerful display processors and improved memory technology mean that the dynamic performance of raster scan displays is constantly improving. These reasons have led to the universal adoption of raster scan displays for graphics workstations and the near-complete displacement of vector scan technology. Vector-scan displays are mentioned here because of their historical importance in the design of animation systems.

## 1.4 Requirements for Animation

In the definition of animation given at the start of this chapter, it was stated that the illusion of smooth movement can be created by rapidly presenting a sequence of images to the viewer. Given this, the following question arises: at what rate must frames be presented in order to properly achieve the desired effect? In determining this, two separate factors require consideration:

1. *Refresh Rate:* The refresh rate is the number of frames presented to the viewer every second. When this rate is too low the result on the frame sequence is image *flicker* and at very low rates the individual frames can be detected. As the rate is increased frames "fuse together" to form a steady image where the contributions from individual frames cannot be detected. The point at which this happens is dependent upon a complex mixture of factors which includes the brightness of the display, the angle of view, the properties of the phosphor (for CRT displays) and the properties of the

observer (that is, the combination of eye and brain which together form the Human Visual System (HVS)[1]). This point is reached somewhere around 30 frames per second (fps) but a rate of at least 40–50 fps is recommended in order to combat flicker effects properly. This rate is referred to as the Critical Fusion Frequency [Jain81].

The standard rate at which films are replayed is 24 fps and in the U.K. the standard video frame rate for CRT displays is 25 fps. Clearly both of these fall short of the rates recommended above.

In the case of film this is overcome by presenting each frame twice during a frame-time to give an effective rate of 48 fps. To achieve this the projector employs a twin bladed shutter whose rotation is synchronised with the movement of the film. The image is blanked once during transition between frames and again mid-way through the frame-time.

To achieve a similar effect with raster scan CRT displays the most common solution is to employ scanline *interlacing*. In an interlaced display each frame is split into two *fields*—an odd field containing all the odd numbered scanlines and an even field containing all the even numbered scanlines. During a refresh cycle the odd field is scanned first followed by the even field. So although the frame rate remains unchanged at 25 fps examination of any particular small area of the screen reveals it to be changing at twice that rate, with a corresponding reduction in the amount of flicker perceived. This relies on neighbouring scanlines having similar intensities. There are problems associated with the use of interlaced displays for graphics. For example, a one-pixel wide horizontal line falls in only one field and consequently is updated at half the rate of the rest of the image. This introduces a distracting flicker for the line. The problems are worse for moving objects as these can interact with the field structure.

With graphics workstations it is possible to consider the adoption of alternative refresh schemes to avoid such problems. For example, non-interlaced displays can be used and the *frame* rate doubled to 50 fps. Recent advances in technology mean that it may be possible to consider such strategies for broadcast television in the future [Clar87]. Higher frame rates have been considered for film too [Fox88].

2. *Animation Rate:* The animation rate defines the rate at which motion must occur in an animated sequence for it to appear smooth to the observer. When objects move at less than this rate judder results and the animation effect breaks down. Rates as low as 10 fps have been used, although 15 fps is a more widely accepted figure for the production of truly smooth animation. Ideally, of course, the animation rate is the same as the refresh rate—that is, 24 or 25 fps. However, much animation is produced at lower rates. This

[1]For example, a television image is an illusion created almost entirely by the HVS since the short persistence of the phosphors used means that only a few scanlines are ever fully illuminated at the same time [Clar87].

is especially true of traditional 2-d cartoon animation which is nearly always produced by the technique of "shooting on twos". That is, each frame is duplicated on the film to give an animation rate of 12 fps at an update rate of 24 fps. The reason for this is historically related to the sheer magnitude of the task of putting together a cartoon film by hand. For example, even at 12 fps a 30 minute cartoon contains over 20,000 frames, each of which has to be individually created and photographed. Although techniques such as *cel animation* ([Magn85*a*]) have simplified this task, much of the work is still extremely slow and highly labour intensive.

Of these two rates it is the animation rate which has the greater significance to the design of an animation system as it is this which determines the number of new images which must be generated each second. The refresh rate can always be satisfied by the use of frame buffering which allows frames to be shown more than once.

## 1.5 Computer Animation Systems

### 1.5.1 Definitions

In the conventional animation process frames are created at a much lower rate than the final replay rate. This approach, which is generally referred to as *frame-by-frame* animation, is widely used in computer animation as well. Alternatively in computer animation it may be possible to generate frames at a sufficiently high rate so as to be able to achieve animation directly, without the need for intermediate frame storage (where "sufficiently high" implies at least the *animation* rate). The difference between this, *real-time* animation, and the frame-by-frame approach can be seen by recognising that in the production of animation there are two distinct activities involved: (1) frame sequence generation, and (2) frame sequence replay. In real-time animation these activities occur simultaneously whereas in frame-by-frame animation they occur sequentially.

In frame-by-frame animation frame sequences are conventionally recorded onto film or videotape and replayed using a film projector or videotape recorder. With computer animation an alternative technique is that of *real-time playback* in which the computer assumes the role of frame replay device. Such an approach offers a number of advantages over conventional methods, as discussed in more detail in Chapter 2. In this context frame replay by conventional means is sometimes referred to as *off-line playback.*

From the above it would seem that real-time animation and real-time playback represent two opposite extremes as approaches to the production of animation. Consider again the frame generation pipeline of Figure 1.1. In a pure real-time system all stages of the pipeline execute at frame rates with frames being displayed immediately. In a pure real-time playback system none the pipeline stages operates at real-time rates and frames are stored for later viewing. Unfortunately, however, the situation is not so clear cut in practice. The basic problem is that the term "real-time playback" is imprecise, having been widely applied to a range of different hardware and software systems. In particular, many so-called real-time playback implementations actually represent hybrid real-time and frame-by-frame systems where part of the pipeline is executed frame-by-frame and part in real-time. A common division point is between the motion specification and image rendering stages. Vector scan displays support dynamic graphics and increasingly it is possible to render frames at a sufficiently high rate on raster-scan architectures with hardware support for the rendering pipeline. However, it is often impossible to specify motion quickly enough—it may require user interaction (e.g., for interactive editing), the running of a complex simulation, or the execution of an animation script interpreter which cannot operate at animation rates. In these cases animation is specified frame-by-frame and some form of (higher-level) frame description stored.

This dissertation is concerned only with the subset of real-time playback systems whose operation are genuinely frame-by-frame. That is, where all stages of the frame-generation pipeline operate off-line and only completed raster frame representations stored for processing by the playback mechanism. Such systems will be termed *stored-frame animation systems* for the purposes of this dissertation. Note that this definition does not preclude the possible use of further frame processing for the purposes of reducing frame transfer and storage requirements.

## 1.5.2 Comparison of Approaches

Of the two approaches identified above, frame-by-frame animation is of greater relevance here as this is the form supported by the stored-frame system discussed in this dissertation. Having adopted this approach it is useful to consider its advantages and disadvantages in relation to those of the alternative of real-time animation.

### General

The principal advantage of a real-time system is its ability to handle interaction. Since frames are produced in real-time (i.e., at least 15 fps) the results of changes made dynamically to the underlying model are instantly reflected in the sequence produced. The major disadvantage of this approach is that such systems are generally difficult to implement. This is as a direct consequence of the need to produce each frame within 1/15s or less. In general, special purpose hardware must be provided and simplified or specialised forms of algorithms employed. It is often necessary to forgo features such as texture, shadowing, anti-aliasing and fine image detail in the interests of speed.

The advantages and disadvantages of frame-by-frame animation are essentially the opposite of the above. The frame creation process for frame-by-frame animation is not time-critical and any algorithms and techniques can be used. Any code employed does not have to be highly optimised and frames can be created on general purpose computers (although some special purpose architectures have been devised). With frame-by-frame systems, however, the entire sequence has to be determined before it is viewed and consequently cannot be changed during replay. In practice, a degree of interaction is possible by writing a number of alternative sub-sequences to the storage device and then dynamically selecting between them during replay. Thus interaction is possible but not generally down to the level of individual frames. This approach is usually called *interactive video* and has increasingly been the subject of attention recently.

### Applications

The implementation effort required to produce many real-time systems means that their application tends to be restricted to those situations where the sequence to be animated genuinely cannot be known in advance. In such cases it is the ability to interact with the system which takes precedence over other factors such as scene-complexity. Examples of such systems include the graphics systems of flight and ship simulators used for pilot training, and certain modelling systems for engineering applications.

Many other applications do not *require* that sequences be generated in real-time and so cannot justify the effort involved in achieving this. In such cases it is far more practical to generate frames off-line using more general techniques. Often the image sequence is naturally fixed and known in advance anyway. An example of this would be a scripted animated sequence that is to be included in a film. In other cases this can be arranged. For example, a "fly through" sequence for the architectural model of a proposed building can be generated off-line; interaction would be useful but is not essential. Often in such cases it is the image quality which takes precedence over the ability to interact with the animation.

For some current generation image synthesis techniques the computational demands are so high that the hardware which can generate frames rapidly enough for real-time animation (or anything approaching it in many cases) does not yet exist. For example, a scene produced using *ray tracing* ([Magn87, Chapters 10 and 11]) may take several *hours* per frame, even on a powerful processor. In such cases animation can only be realised with a frame-by-frame system.

Real-time systems are improving rapidly all the time. Some current generation flight simulators, for example, have impressive image generation capabilities including texturing and shadowing. Even so, widespread and low cost access to such systems is still some way off in the future. In addition image synthesis techniques continue to improve in order to satisfy appetites for ever greater frame complexity, with the result that the disparity between computational demand and availability is not reduced. For these reasons frame-by-frame systems look set to continue for some time.

In the remainder of this chapter examples of the real-time and real-time playback animation systems are examined. This discussion excludes consideration of stored-frame animation systems as these are covered in detail in the next chapter.

## 1.6 Real-Time Animation Systems

Under the previous definition of a real-time animation system, a wide range of implementations are possible. Here these are considered under three categories.

### 1.6.1 Special Purpose Architectures

A special purpose architecture is one that is optimised for a particular animation application. For a high performance system it is usually necessary to provide hardware support for all stages of the frame generation process (Figure 1.1, including the stages in subordinate pipelines such as the 3-d rendering pipeline illustrated in Figure 1.2). Typically such systems are implemented as a pipeline of processing stages corresponding closely to the stages shown in the figures. The very high cost associated with the design and construction of such systems limits their application to a few specialised areas. Examples already mentioned are those of flight and ship simulators. Here the cost involved is justified by savings gained in other areas (such as reductions in the number of (expensive) real flying hours required, for example).

The high cost of such systems generally rules out their use for more everyday animation applications, although this situation is continually changing as prices fall. Eccles *et al* [Eccl83] have considered the application of simulator technology to more general animation.

### 1.6.2 General Purpose Architectures

More accessible for ordinary animation applications are graphics workstations of the form depicted in Figure 1.4. In such systems there is less specialised hardware support for the animation pipeline with the result the attainable real-time performance is generally lower (as more of the processing has to be done by software).

In 1979 Baecker [Baec79] surveyed the state-of-the-art of dynamic graphics systems at that time and outlined a methodology for describing such systems. One of the conclusions reached was that the use of a frame buffer based representation was not desirable for dynamic graphics. To a large extent this is still true today, although improvements in technology mean that this situation is continually changing. Perhaps the greatest advances have been made possible by the development of high performance VLSI graphics processor chips. A brief history of the evolution of these devices is traced by Fontenier *et al* [Font88] who also present their own design for a 2-d graphics processor. Typical of recent VLSI implementations of graphics processor chips are the 34010 from Texas Instruments [Asal86], the 82786 from Intel [Shir86] and the Geometry Engine from Silicon Graphics [Clar82]. A good example of a current high-performance graphics system is the Silicon Graphics' IRIS workstation [Akel88]; this provides hardware support for all stages of the rendering pipeline illustrated in Figure 1.2.

Another way to improve the performance of a display system is to consider alternative arrangements of the frame buffer or novel forms of display processor or display controller. Over the past few years a number of experimental systems have developed. Mention should be made of the following: DisArray [Page83], the 8-by-8 display [Gupt81,Spro83] and the Rainbow Display [Wilk84,Styn85].

### 1.6.3 Other Real-Time Techniques

A number of methods have been devised which can achieve real-time animation on ordinary frame buffer based graphics systems by using the existing hardware in novel ways. Perhaps the best known of these is ***look-up table animation*** as described by Shoup [Shou79] which uses the system's video look-up table[2] to manipulate pixels on the screen whilst leaving the frame buffer representation unchanged. The small size of the look-up table means that entries can be easily and rapidly reloaded, unlike the values in the frame buffer. Changes are usually made during the field vertical retrace period and their effects are instantly reflected on screen during the next frame period. This provides a basis for real-time animation, and a number of examples are given in [Shou79]. Extensions are possible to this technique. For example, experiments with look-up tables in conjunction with the special-purpose hardware of the Rainbow Display showed a wide range of effects were possible. These included transparency and real-time anti-aliasing [Glau85*a*,Glau85*b*].

With some ingenuity on the part of the implementor, look-up table animation can provide some interesting effects for some real-time applications. However, it also has some severe limitations, particularly related to the complexity and range of movement and length of sequence which can be achieved, and it is these which limit its use as a general purpose animation technique.

## 1.7 Real-Time Playback Systems

The idea of real-time playback is not new and can be traced back to the early work of Baecker [Baec69]. One of the earliest attempts at analysing a real-time playback system was made by Potel [Pote77] who considered the design of a vector display based graphics system and presented a model for describing its key parameters. This work was based upon experience with a commercially made system comprising a DEC PDP 11/40 and a DEC VT-11 vector display processor. Potel used a ten parameter model which included factors such things as display file size per frame, display processor speed and secondary (i.e., disc) storage space available. This model was used to evolve four conditions that are required for real-time playback feasibility, taking into account such factors as secondary storage latency, and synchronisation and buffering requirements. The remainder of the analysis was concerned with two special cases which Potel called Real-Time

[2]A common feature of many graphics workstations is a *look-up table* (or *colour table*) RAM that sits between the frame buffer and display logic. This table provides a mapping between values in the frame buffer and the pixel values displayed. Its principal use is to overcome the restriction of the limited range of colours supported by many frame buffers.

Reversal and Upstream/Downstream effects. A desirable property of a real-time playback system is that the animation may be viewed in either the forwards or backwards direction. Real-time reversal is a further property whereby the direction of traversal may be reversed at any instant in real-time. The difficulty in achieving this stems from the fact that in most real-time playback systems it is necessary to pre-fetch and buffer frames in advance of their being displayed in order to overcome latencies and discontinuities associated with the frame storage device. Upstream/downstream effects are caused by there being a fundamental difference in the forward and reverse retrieval rates of data from most secondary storage devices. This results in a difference in which forward and reverse viewing may proceed.

A later analysis, along similar lines to that given by Potel, is presented by Egan *et al* in [Egan84]. Here the analysis is done by way of comparison on two real-time playback systems. The first of these is similar to that described by Potel and comprises a PDP 11/45 with an Evans and Sutherland vector display. This can achieve a replay speed of between 15 and 18 fps. The second system is based upon a raster scan black and white (i.e., bi-level) display and microcomputer architecture (TERAK 8510a using an LSI–11 processor). The description of this as a real-time playback system is somewhat charitable since it employs floppy disc drives for frame storage which have a capacity of only 26 frames and the overall replay speed as low as 1 or 2 fps!

A real-time playback system based on a raster scan display has been described by Ackland *at al* [Ackl80]. This design employs a custom microprogrammable display processor (See Figure 1.4) called GUMBI (Graphical User Programmable Bit-Slice Interpreter) for translating high-level primitives supplied from the host (e.g."draw filled polygon") into the corresponding lower level frame buffer representation. To improve the performance of the system a new polygon fill algorithm (designated "edge-fill" by the authors) was developed and a microcode implementation produced for GUMBI. The system supports the real-time playback of 2d or 2½d[3] frame sequences. Frames are represented in a high level polygonal format as a means of overcoming bandwidth limitations during replay. Real-time playback is achieved with the use of a software package which reads coded polygonal data stored on the host's disc and feeds this data through GUMBI in order to update the frame buffer representation at frame rates. The performance of this system is limited by the rate at which GUMBI can process polygons. An analysis of this performance [Ackl80] estimates the maximum picture complexity to be 55 polygons per frame for a frame rate of 15 fps (assuming 50 pixels per polygon edge and 6 vertices per polygon). Also the system cannot handle general purpose frame data but is restricted to the polygonal data for which the design is optimised.

The above examples serve to typify previous work on real-time playback mechanisms. The number of systems which have been described in the literature is surprisingly small, given the usefulness of such a mechanism. Even so, two general observations can be made:

[3]2½d refers to the use of a finite number of discrete 2d drawing planes organised in a priority ordering in the z dimension.

1. Most real-time playback systems have been implemented on unmodified commercially available (i.e., non-custom) hardware using software utilities to achieve real-time playback. In particular the use of vector scan displays is common in such systems. In these cases the playback performance is limited by the properties of the underlying system—DMA rates, bus speeds, main memory cycle time, effects of resource conflicts, etc.

2. Many real-time playback systems are "impure" in the sense that playback involves an element of real-time processing (as in the case of Ackland's system above, for example). In such cases the class of image supported, the achievable image complexity or animation rate is limited in the same way as for a real-time system (Section 1.5.2). This limits the complexity of animation which can be achieved.

## 1.8 Real-Time Playback in Animation Packages

Another application of real-time playback is with animation packages used for developing animated sequences. In some such packages, particularly those where animated sequences are generated interactively, a real-time playback facility is provided to test the motion dynamics of a piece of animation. This enables the animator to gain some idea of what the finished sequence will look like and so judge whether the animation appears correct with respect to smoothness, speed of motion, etc. This facility corresponds to the *pencil test* technique used in conventional animation, where rough pencil sketches are produced and viewed at frame rates prior to the laborious task of inking and painting of frames takes place (see [Magn85*a*])

With this form of real-time playback effort is concentrated on producing accurate motion representation, often at the expense of image quality, since this is more important when gauging the correctness of a piece of animation. Consequently, wire-frame models are often employed for this purpose as these are computationally much less expensive (and ideal for use on vector displays which are widely used for this task). When the sequence has been checked and approved the individual frames are then generated at full resolution with full detail and written to either film or video tape for later viewing at the proper speed.

Examples of systems having this facility have been described by Levoy [Levo77] and Gómez (the "TWIXT" animation system) [Gome85].

# Chapter 2

# Stored-Frame Animation Systems

## 2.1 Introduction

The previous chapter considered animation in general and mentioned some examples of both real-time and real-time playback systems. This chapter concentrates upon that subset of real-time playback systems identified in Section 1.5.1 and known as stored-frame animation systems for the purposes of this dissertation. The chapter is divided broadly into two parts. The first part considers possible frame storage devices for such a system. The second part examines previous examples of such systems and introduces the motivation for the development of a new system as described in this dissertation. The new system is discussed in the context of previous approaches.

## 2.2 Framework

The general organisation of a stored-frame animation system is shown in Figure 2.1. This is the simplest conceivable model for such a system, but nevertheless one which serves as a useful point of reference for the following discussion.

The system controller block is responsible for ensuring that frames fetched from the storage device arrive at the display in the correct format, in the correct order and at the correct rate. Exactly what is required to achieve this obviously depends

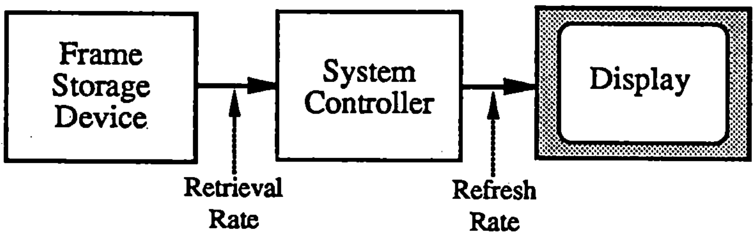


Figure 2.1: General framework for a stored-frame animation system

heavily upon the details of the frame storage and display devices. In addition to the control aspects involved, the system controller may have to process the frame data arriving from the storage device in order to get it into a form suitable for display; a simple example of this is analogue to digital conversion.

In Figure 2.1 the *display rate* is the rate at which data must be sent to the screen in order to keep the image properly refreshed, as discussed in Section 1.4. This rate depends upon a number of factors such as the resolution of the display, the refresh rate required, and whether or not interlacing is required. As an example, consider the CCIR Recommendation 601 [Kret85] which defines a digital representation of a T.V. signal for a resolution very close to that currently used in the U.K. analogue PAL system[1]. In this representation there are 625 lines of 720 visible samples per line, refreshed at 25fps interlaced. This is a component coding scheme with luminance sampled at 13.5MHz and the two colour difference signals sub-sampled at half this rate (6.25MHz)[2]. Samples are quantised to 8 bits. The refresh data rate required is thus 216Mb/s[3]. Much higher resolutions are often used in modern graphics workstations, 1280×1024 pixels at 24 bits/pixel is commonplace. At 25 fps (interlaced) this corresponds to a transfer rate of approximately 786 Mb/s.

The *retrieval rate* is the rate at which frame data can be recovered from the storage device. Depending upon the type of device used, this may or may not be equal to the required display rate (a point discussed at greater length later on). The achievable transfer rates vary widely across the range of possible storage devices; examples of actual rates are given in the discussions of individual devices below.

## 2.3 Frame Storage Devices

In the following subsections the suitability of a number of storage devices for use in a stored-frame animation system is considered. The candidates considered represent a complete list of the most practicable devices and are: film, semiconductor memory, analogue and digital videotape, videodisc/optical memory systems and rigid magnetic disc.

First, however, some general requisites of a frame storage device can be listed. These constitute the properties which would be found in the perfect device:

- *Large Storage Capacity:* A major problem with stored-frame animation systems is the vast amount of information required to represent even short animated sequences. For example, a one minute sequence at 25 fps at the

[1]Actually, Recommendation 601 defines an extensible family of coding standards intended to unify the current diverse standards of PAL, NTSC and SECAM

[2]Component coding is widely used in the broadcast television industry. A signal is defined in terms of a luminance component (Y) and two colour difference components (U,V). The figures quoted here refer to a form of Recommendation 601 known as the 4:2:2 studio standard, the most widely implemented variation to date

[3]A general convention adopted in this dissertation is that lower case 'b' refers to *bits* and upper case 'B' to *bytes*. Thus, the rate quoted is 216 Mega*bits* per second.

CCIR recommendation 601 resolution outlined above requires storage of approximately 1.4 GBytes. A similar length sequence at the higher resolution quoted would require nearly 6 GBytes. Thus, the capacity of the storage device is a major consideration in the design of an animation system.

- *High Retrieval Rate:* A related requirement is that the data for each frame must be fetched from the device within (on average) one frame time. Again, for some classes of storage device this represents a major problem. If the retrieval rate is to be equal to the display rate then a transfer rate of several hundred megabits per second may be required. For example, the rates for the two resolutions cited above would be 216Mb/s and 786Mb/s.

- *Digital Representation:* For a digital hardware animation system it is desirable that frame data also be represented and stored in a digital format. This removes the necessity for conversion between analogue and digital representations, and the associated possibility of image quality degradation. Also a digital representation is intrinsically more robust and error tolerant than the analogue form. This advantage can be further consolidated by the use of error detecting and correcting codes which allow the effects of frame errors, distortions and noise in a replayed sequence to be minimised.

- *Random Access/Special Mode Access:* In some applications, particularly when long animated sequences are involved, it is useful to be able to access the sequence starting at an arbitrary point without having to replay from the start each time. This requires random access to the storage device. Another useful facility is to be able to vary the rate and/or way in which frame data are fetched from the storage device. This allows for "special-play" modes such as still-frame and slow or fast motion.

- *Locally Writable:* The process of recording frames onto the storage device should be straightforward and accomplished locally by animation system users themselves. For some classes of storage media (e.g., certain forms of videodisc) this is infeasible and data can only be recorded using a complex mastering process at remote sites. Such processing may take weeks or months.

- *No Post-Processing:* Again related to the recording process, it is desirable that the media be readable immediately after recording. Some media require post-processing after recording—an obvious example is film which requires developing and printing.

- *Media Re-usability:* The writing of frames to the storage device should be a reversible process, that is, recording should not be destructive. Examples of media for which this does not hold include film and some forms of optical disc. A related consideration is that of the recorded data's robustness and permanence. For some classes of storage device there is a limit on the number of times that the recorded information can be read because each read cycle slightly degrades the recorded representation.

- *Low cost:* Clearly low cost is advantageous for the storage device and media because of the large amount of storage required for an animation system. Cost is an especially important consideration when the storage medium is not re-usable.

### 2.3.1 Film

This section is included for the sake of completeness. Film is still a widely used medium for recording the output of frame-by-frame animation systems, mainly for applications within the film (movie) and television industries. Computer animated sequences recorded on film are replayed using an ordinary film projector. Film could be used as a storage device in a digital animation system by using a modified form of telecine equipment to scan frames from film. The advantages of film as a medium are its very high capacity, both per frame and overall[4], and its low cost. The disadvantages are that it is not easy to incorporate into a digital system, it is not re-usable, it is fragile (care is required to avoid dust and scratching), it needs post-processing and it is not randomly accessible.

### 2.3.2 Semiconductor Memory

Semiconductor memory is widely used for single frame storage in frame store architectures. But its use for multiple frame storage (as the frame storage device in an animation system) has, even in the recent past, been ruled out on economic grounds. The large amount of storage required has meant that such an approach would be prohibitively expensive. However, continued dramatic increases in available capacity and performance, matched by falling chip prices, mean that the situation is constantly changing. Riley [Rile87*a*] traces this trend in relation to the digital storage of television signals. It is now reasonable to consider the design of semiconductor storage systems with capacities of the order of tens (or perhaps hundreds) of megabytes, a figure which competes with the lower capacity magnetic drives.

In spite of this it is currently uneconomical to provide the hundreds or thousands of megabytes of storage which are required for a general purpose animation system. For some applications less storage can suffice. For example in [Rile87*b*] Riley considers the design of a large capacity picture store for use in an image processing system for television sequences. The proposed design employs multiple storage modules, each with a capacity of 50MBytes. This gives a playback time at full television resolution of only a few seconds (five or so), but this is sufficient for the intended application. With sufficiently small frames a similar technique can be employed on ordinary graphics workstations, provided they have a moderate amount of main storage. Frames are computed off-line and placed in memory.

[4]A 70mm film frame (52.6mm$\times$23.01mm) has an equivalent resolution of approximately 4208$\times$1841 pixels (assuming 80 pixels/mm vertical resolution). At 24 fps this corresponds to a data rate of approximately 4.5 Gb/s. A 90 minute film has an equivalent total storage capacity of approximately 2 Terabytes ($2 \times 10^{12}$ bytes)!

Each frame is then briefly displayed in turn by transferring it rapidly to screen memory using a BitBlt operation. Alternatively, hardware panning and scrolling hardware can be used to achieve a similar effect. Since only a very limited number of frames can be displayed in this way, the technique works best for cyclical animation (e.g., rotating objects) which allows continuous motion from only a few frames.

The advantages of semiconductor memory as a frame storage device are the very high transfer rates possible, random access (down to pixel resolution if required) with no seek delays or other latencies, digital image representation, high reliability (no moving parts), infinite re-usability and local writability with immediate playback. The principle disadvantage is that the affordable capacity is limited, falling short of that demanded by many applications. A second limitation is due to the fact that the most likely devices to be employed are volatile and so data are lost at power-down. Thus, some form of secondary storage device (disc or tape) must be employed for any animation system implemented in this way.

### 2.3.3 Videotape—Analogue and Digital

Videotape has also been widely used, as an alternative to film, to record the output of frame-by-frame animation systems. As a real-time playback medium videotape has a number of significant advantages over film, as well as some drawbacks. The advantages of videotape include: low cost, very high capacity, video-rate frame transfer, re-usability, and local recordability with immediate playback. The principle disadvantage is that to achieve frame-by-frame writing a *stop-frame* recording mode is required, something normally found only on the more expensive machines designed specifically for professional videotape editing purposes.

Videotape recording evolved as a logical progression from audio tape recording. Work started in the early 1950's and in its present form video recording was pioneered by the Ampex Corporation in 1956. The main problem when recording video data compared with audio data is the much higher bandwidth involved (300 times that of audio signals). This requires a much higher head-to-tape speed. Early attempts at simply increasing linear tape speeds were not particularly successful[5] and the solution now adopted, as demonstrated by Ampex, is to employ rotating heads within a drum which scan across the tape at high speed whilst the tape moves relatively slowly past the drum. The first machines employed four heads which scanned transversely across the tape, the so called *quadruplex* or *quad* recorder.

A number of problems associated with quad recorders were overcome by the introduction of *helical-scan* recording. With this technique the tape forms a helix around the drum and the combination of tape and drum movement result in long tracks being recorded at a shallow angle across the width of the tape. The use of two recording heads in conjunction with a $180^{\circ}$ wrap is common. A large number of recording formats have been defined and standardised over the years. The domestic

[5]In 1954 RCA produced a longitudinal track recorder which had a tape speed of 360 i.p.s. and was still incapable of recording a full bandwidth video signal. (Compare this rate with $1\frac{7}{8}$ i.p.s used in audio cassette players.)

and educational formats are well known—for example, U-Matic (Sony, 1971), Betamax (Sony, 1976), VHS (JVC, 1978) and most recently the new 8mm standard (various, 1984). The two main helical-scan standards for broadcast use are less familiar—the Segmented B-Format and the Non-segmented C-format, developed by the SMPTE (Society of Motion Picture and Television Engineers) and EBU (European Broadcasting Union).

The long tracks produced by helical scanning make it possible to record an entire video field on a single track. Such a non-segmented format has a number of attractions; for example, the perceptibility of scanning errors (caused by geometrical changes during reading) tend to be reduced. A side effect of this arrangement is that a stop-motion feature becomes easy to implement. If the tape is stopped but the heads kept spinning then the same field is shown repeatedly to give the effect of still frame. However, this simple approach is not perfect and without the motion of the tape there is a misalignment between the recorded track and the path of the rotating head. This results in a portion of the guard band (the unrecorded region between tracks) being read in place of video data and the production of an unwanted noise band on the screen (although with careful tape positioning this can be displaced to the top or bottom of the screen). Similarly, if the tape is moved at less than full speed, "slow-motion" effects are produced by repeating fields. In this case the noise bar drifts though the picture at a rate determined by the tape speed. Proper performance in these cases can only occur by providing a more sophisticated tracking mechanism known as Automatic Scan Tracking (AST) which was developed by Ampex.

Unfortunately this ability of even relatively unsophisticated recorders to read single fields is not matched by an ability to *write* frames individually. It has already been seen that such a capability is vital for implementing a stored-frame animation system. The process of adding a single frame to the end of a partially completed sequence is a special case of video editing and in general the editing of sequences on videotape is not trivial. For helical scan recorders this must normally be done electronically, although limited physical editing of tape is possible [Robi81]. The basic problem is that new fields must be positioned very precisely in relation to the rest of the sequence if the splicing is not to be visible because of unwanted phase changes in the playback signal caused by field misalignments. This requires very precise control over the various head and capstan servo mechanisms controlling drum and tape speeds and the record signal sent to the heads in order to ensure proper timing and synchronisation. In the general case is is also necessary to control a flying erase head which tracks some distance ahead of the record heads. To identify field positions on the tape some form of longitudinal cue track is employed upon which cue tones are recorded. Recently the trend has been to fully electronic editing in which much more sophisticated codes are recorded on a control track (e.g. SMPTE/EBU Time Code Addressing). For more details of the editing process refer to Chapter 14 of [Robi81].

The upshot of the above is that whilst videotape has a number of attractions as a storage medium for use in a real-time playback system, a low-cost practical implementation is impossible to achieve because of the difficulty of laying down frames one at a time. Even if this was not the case, for performance reasons it is probably necessary to use one of the professional videotape formats—that is, U-Matic or better. Analogue recording is inherently imperfect because of linear and non-linear distortions introduced during the record and replay processes. Careful design of the system helps to reduce these effects to acceptable levels. Even so, over multiple read/write cycles image degradation does occur. The problem with domestic and semi-professional formats is that more emphasis is placed on compact tape size and longer play-time, to the detriment of image quality. In particular, two techniques are widely used in such systems: "Colour-under" (for video bandwidth reduction) and Slant Azimuth recording (for increasing track densities). Both of these were introduced by Sony (See [Wood86]).

Recent advances in *digital* video recording may provide a solution to the problem of image quality. Much of this work is still at the experimental stage (e.g. [Bell86]) but recent moves towards the standardisation of formats [Wilk87] may result in Digital Videotape Recorders (DVTRs) gaining wider acceptance. Digital recording has many advantages over analogue recording, not least the ability to use error detecting and correcting codes [Gill87]. These allow loss of data caused by *tape dropouts*—relatively large areas of tape in which the magnetic coating is defective—to be corrected such that a much lower proportion of recorded frame information is lost. In such a system it also makes sense to implement effects such as stop- and slow-motion digitally (i.e., using frame storage) rather than by mechanical tracking systems.

Such technology, however, is not yet widely available outside of the broadcast studio and the basic requirement of accurate track placement during recording remains (and so, therefore, does the difficulty with single frame recording).

Finally, mention should be made of magnetic videodisc technology [Robi81]. This was developed specifically for stop- and slow-motion before such effects became practicable on helical-scan machines with AST. A typical system used a 16 inch diameter disc with a nickel cobalt recording surface. The usable recording area was a 4.5 inch band which allowed for 450 concentric tracks on each side, with one field recorded per track. The principal disadvantage of this technology was the short sequence length attained (18 seconds at 25 fps), although this was sufficient for the intended applications such as slow motion replay of sports events. The technology has now been superceded by more advanced helical scan videotape recorders. However, a number of its more elegant features have survived and can be identified in some current optical and magnetic disc technologies.

### 2.3.4 Videodisc and Optical Memory

One of the most promising developments over recent years has been that of optical storage. A variety of technologies exist today, all of which have evolved from the analogue videodisc systems that first appeared during the 1970s. Current systems represent the state of the art of a technology whose origins can be traced back to work started in the late 1950s. As the name suggests, such discs are written and (usually) read optically, normally by a low powered semiconductor laser diode whose beam is focussed to a spot size of around $1\mu$m. Work on optical storage was largely motivated by a desire to produce an inexpensive consumer video replay system. The concept was exactly the same as that of the vinyl audio record; thousands of inexpensive discs would be replicated with a pressing process employing stampers made from an original master recording. The first videodisc system to use a laser for both mastering and replay was demonstrated by Philips in September 1972 and a number of domestic videodisc systems followed. However, these were never commercially successful, principally because they were in direct competition with the emerging videotape technology[6].

Work on a *digital* disc system for optical data storage was begun in 1975. This was the forerunner of the current generation of optical disc drives. The arrival of these drives has been much heralded over recent years (e.g., [Bart78,Stra83,Fuji84]), but many of the associated claims and predictions (such as the imminent demise of magnetic recording) have not been realised. Current limitations, in particular the write-once nature of the media, have so far severely restricted the number of application areas.

In addition to the basic distinction between analogue and digital recording of data, many other technological variations are possible. These include differences in the types of media used and the means by which data are recorded and read, and are discussed in detail in [Isai85]. Broadly speaking optical storage systems divide into two categories: analogue mass-replica discs which are pre-recorded and read-only, and digital optical read/write discs which are supplied "blank" and written by the disc drive[7]. In keeping with convention, this dissertation uses the term *videodisc* to refer to the former and the term *optical disc* to refer to the latter type.

### Videodisc (Analogue)

A video signal is represented on the information plane of a videodisc as a series of tiny pits moulded into a plastic substrate. This surface is made highly reflective by coating it with aluminium and is covered by a clear layer to protect against dust, scratching, and oxidisation. Recording relies upon a process of *diffraction*; much less light is reflected from the base of a pit than from the information plane surface because the size of the pits, and in particular their depth, is accurately controlled to be some fraction of the laser wavelength (typically $\lambda/4$ or $\lambda/8$). A frequency modulated video signal is carried on a videodisc by variations in the pit lengths and inter-pit gap. Normally two audio channels are also recorded, superimposed on the video signal. Information is recorded along a single spiral track, similar to that on an ordinary audio LP record and the signal is replayed from the centre outwards. Two types of videodisc are used. With a Constant Angular Velocity

[6]Optical disc technology has survived in the domestic market in the form of the digital audio optical disc—more commonly known as the Compact Disc (CD). The first CD system was demonstrated by Philips in 1979.

[7]Because of the preponderance of systems employing write-one media the acronym WORM (Write Once Read Many) is often used to describe such systems.

(CAV) videodisc, the disc is rotated at a constant speed. Typically in such systems one frame is recorded for each revolution and the speed of rotation is set such that each revolution takes exactly one frame time—i.e., 1500rpm for PAL/SECAM (25 revs. per second) and 1800rpm for NTSC. The advantage of CAV is that special play modes are easy to achieve. For example, still frame is produced by continually skipping back one "track" at the end of a frame so that the frame is read again during the next frame time. With the alternative Constant Linear Velocity (CLV) videodisc the rotation rate is decreased as the read beam tracks towards the disc's outer edge. This results in a uniform linear packing density across the whole disc surface and a corresponding increase in total playing time compared with the CAV videodisc. The disadvantage of CLV discs is that they are restricted to forward play at the full frame rate.

The principal drawback of videodiscs for use in a stored-frame animation system is that a complex mastering process is required to write video data to a disc. The videodisc process was specifically designed to enable large numbers of duplicate discs to be produced easily and cheaply. One-off production is highly uneconomical. Also, the mastering process normally requires that data be recorded in a single operation and at video rates. This is necessary for producing a continuous spiral track [Isai85, Chapter 2] and mastering at less than full rate has been found to be difficult. Recall that the key objective of stored-frame animation is to remove the need to generate frames at frame rates. This clearly doesn't apply to videodisc.

Another drawback of analogue videodiscs is that image quality tends to be lower than, say, videotape. This is because the high areal packing density results in a relatively high percentage of dropouts and these appear as noise in the replayed sequence.

### Optical Disc (Digital)

The problems encountered when writing frame data to a videodisc are overcome with the optical disc. Optical discs are designed to appear outwardly similar to the magnetic disc drives with which they are intended to compete. Data on an optical disc are recorded on a spiral track, but this is made to appear similar to a magnetic disc with tracks and sectors. Control information is recorded in each sector along with the data bits for the purposes of synchronisation, sector identification, error control, etc. Again, this is similar to magnetic disc technology. Optical discs are supplied pre-formated, (that is, with tracks already marked and sector header information written). The *pre-grooving* technique employed for this was developed by Philips [Bult79] and is normally accomplished by a mastering process similar to that used for videodiscs.

A variety of media and recording methods have been developed for optical discs. However, nearly all of the current commercially available drives employ ablative thin film discs in which information is represented in a similar way to that for the videodisc. Writing is accomplished by modulation of a higher intensity record beam. Where this beam hits the information surface a highly localised heating occurs resulting (irreversibly) in the formation of a either a pit or raised bump (depending upon the medium). The most commonly employed material for the film is the element Tellurium, usually used in an alloy form.

Optical discs are of particular interest to the design of an animation system because of their large storage capacities (1000 MBytes on each side of a digital 12 inch disc is typical), made possible by the high data packing density. Such systems are now available commercially, three of the earliest were from Thomson-CSF (Gigadisc 1001 series), Hitachi (301 series) and Optical Storage International (LaserDrive 1200). The principal disadvantages relate to transfer rate and cost. Typical transfer rates are in the range 3—5Mb/s, much lower than the rates achievable with equivalent magnetic storage. Cost is a particular problem with current discs due to their write-once nature. Such discs cost over £500 each in 1986, a price at which an economic implementation of an interactive animation system was simply not feasible.

**Future Trends**

Optical storage has been under development for many years and during this period considerable experience has been accumulated. Further technological improvements are already evident in the most recent generation of drives which are physically much smaller than earlier systems. At least nine major manufacturers now produce 5¼ inch optical drives. These have storage capacities of between 200MB and 800MB per disc and transfer rates in the range 2.2—6.5Mb/s.

The major limitation with present optical storage is the write once nature of the media and much research is currently underway to find erasable alternatives. Candidate media include magneto-optical, chalcogenide films, thermoplastics, photochromics and photoferroelectrics (see [Bart78] for further details of these). The most promising technique currently is magneto-optical recording (a combined magnetic/optical process) and at least two manufacturers now produce magneto-optical drives. Details of this technique are discussed by Nomura *et al* in [Nomu87]. A number of question marks remain with many of the proposed new media. Problems to be addressed include limited data retention, limited disc life and partial destruction of recorded data during reading (i.e., a limited number of read cycles).

## 2.3.5 Magnetic Disc

Perhaps the most familiar type of mass storage device for computer applications is the magnetic disc drive. The discussion below is limited to the most commonly found form of this device, the Winchester disc [Wood86]. The first Winchester disc was developed by IBM (the model 3340) and appeared in 1973. Since then the adoption of such discs has become widespread. The principal characteristic of this technology is the use of a hermetically sealed head and disc assembly (HDA) which protects the operating mechanism and data surfaces from all external contamination. At manufacture the drive is assembled in an ultra-clean environment and during normal operation air is continually re-circulated and filtered internally to ensure no unwanted particles reach the data surfaces or heads. This arrangement allows the head-disc gap to be reduced to a tiny distance (typically 0.2 to 0.4 $\mu$m) with a corresponding increase in the data packing density. Also, because the discs are not removable, much tighter mechanical tolerances can be achieved. Capacity is further increased by the use of multiple (up to 10) disc platters, each with its own read/record head.

The years since 1973 have seen a remarkable increase in the performance and capacity of Winchester drives. This can be largely attributed to developments in two main areas. Firstly, improvements have been made to the materials used for both the recording surfaces and the heads. The most common recording media are based upon particulate iron oxide. More recently, however, manufacturers, particularly those of small drives, have turned to metallic film media (e.g., Nickel Cobalt) which can support much higher packing densities. Such drives usually employ thin-film heads rather than the more common ferrite heads. The second main area of development has been with the head positioning mechanisms used for tracking and seeking. The greater track densities required for higher capacity drives demand much more sophisticated control over head positioning if data are to be recovered accurately and access times are to be fast (i.e., seek delays low). For track densities of more than about 1000 tracks per inch (tpi) a linear or rotary voice-coil actuator is used in place of the simpler stepper motor found in lower capacity drives, and a servo mechanism employed to control head positioning. Generally one disc surface and head is dedicated to servo information. The special "quadrature" pattern recorded on this surface during manufacture is used to provide track following during normal tracking and track crossing information during seeks as well as other timing information. In higher performance drives the control system within the servo loop is normally intelligent (i.e. microprocessor based). For example, during a seek to a new cylinder, the driver current applied to the voice-coil actuator which governs the acceleration and deceleration phases of the heads is carefully controlled according to a pre-determined profile. The result is that the seek time is minimised across all seek distances and the heads are brought to rest accurately over the new cylinder with a minimal chance of seek error.

Winchester drives are available from a large number of manufacturers with a wide range of capacities and performance. Drives are produced in various sizes, most commonly 14 inch, 8–10 inch, 5¼ inch and, most recently, 3½ inch. Available capacities vary from a few megabytes to well over one gigabyte; 1 GB is available even on medium range drives (e.g. the 8 inch Sabre-1230 from Control Data), and more recently on 5¼ inch drives. A typical transfer rate for a 5¼ inch drive is 5–10Mb/s, but 15Mb/s drives are now emerging (e.g. the Maxtor XT-8000 series). In larger drives transfer rates of up to 24Mb/s are available, and 45Mb/s has been demonstrated in prototype form by NTT [Wood86].

### Parallel Transfer Drives

The data transfer rates cited above represent the currently obtainable limits of performance, obtained only with much innovation and considerable design effort on the part of drive manufacturers. For even higher performance alternative strategies must be sought. One recent development of particular relevance to the design of an animation system has been the emergence of *parallel transfer* disc drives (PTDs). In such systems data are transferred from several heads in parallel instead of the more usual one head at a time; this results in an immediate $n$-fold increase in the transfer rate (where $n$ is the number of participating heads).

At the time of writing parallel transfer drives are only available from one manufacturer, Fujitsu (with models M2350A (474MB) and M2360A (689MB)), although Ampex and Century Data have also carried out development work in this field. The Fujitsu drive is based upon their standard 10½ inch Eagle drive which is augmented with additional logic to support the multiple data channels. Data are transferred using either 1, 4 or 5 channels with peak transfer rates of 19.7Mb/s, 78.6Mb/s or 98.3Mb/s respectively (for the M2360A).

### Multiple Disc Drives

A variation on parallel transfer drives is to employ multiple ordinary drives operating in parallel. In [Kim86] Kim enumerates the advantages of a multiple drive system over alternative approaches and presents an analysis of the performance of such systems. The advantages include improved system performance and reliability. Reliability can be improved by the adoption of a dual-mode error correction scheme which not only detects and corrects the normal single bit errors but also allows correction when entire bytes are lost, that is, when one of the drives fails. The modular nature of the multiple drive approach means that a faulty unit can be replaced without affecting the remainder of the system.

Suitable disc controllers have appeared recently. An example is the Concept 51 from Storage Concepts which can handle transfers from a row of between two to nine 5¼ inch drives in parallel, giving a sustained data rate of up to 135Mb/s. The controller supports up to eight such rows of drives (i.e., 72 drives maximum). A maximal system can have a total capacity of up to approximately 50 GBytes. A disadvantage of this approach, compared with a PTD, is the potential high cost, largely due to the duplication of mechanical components (especially arm and actuator mechanisms); for the system just described the cost of drives alone could exceed £400,000 (1987 price).

The design of a controller is simplified if the drives are mechanically synchronised to reduce data skew. A number of drives now provide control signals to facilitate this. These signals are derived from spindle speed and position information generated by the drive and form part of the spindle motor's servo feedback loop. With such a system skew can be limited to less than $3^\circ$ ([Kim86]) requiring only a few hundred bytes of deskewing buffering. However, synchronisation is not necessarily *required* for the implementation of an animation system. Ng [Ng88] has also considered the design of disc arrays. In his analysis he makes the observation that, whilst there is a performance penalty associated with unsynchronised drives, for sustained data transfer data skew represents only an initial overhead which becomes insignificant in relation to the data transfer time. More precisely, the average time taken before all drives have come online does increase, but thereafter all drives supply data continuously and there is no outward difference from a synchronised array.

## 2.4 Previous Stored-Frame Animation Systems

### 2.4.1 Introduction

In the remainder of this chapter some previous approaches to the design of a stored-frame animation system are presented. The number of systems that have been described in published material is actually quite small and the following discussion represents what is believed to be a complete survey of work relevant to that presented later in this dissertation. A number of potential frame storage devices have already been considered for their suitability for use in an animation system. Referring back to Figure 2.1 (page 15), frame storage devices may be divided into two classes according to their data retrieval rates: (1) those for which video rate bandwidth is available directly, and (2) those for which less than video rate bandwidth is available. These two cases are now examined in turn.

### 2.4.2 Systems Using Video Rate Storage

This class obviously includes those devices such as film projectors and videotape recorders which are specifically designed for video-rate replay. It has been mentioned that these devices are widely used for the production of frame-by-frame animation, and some of their shortcomings have already been examined. This approach—*Off-line Playback*—is of marginal interest to the current discussion because of its inflexibility; it generally provides only a straightforward replay facility. A less well understood and more challenging approach is to embed the frame storage device in some form of computer system with the aim of providing much greater control over the frame recording and replay processes. This provides the potential for a much higher degree of interaction with the animated sequence during replay.

#### Movie Maps

An example of this more flexible approach is the Movie-Maps system developed at MIT [Lipp80] which employs two videodisc players for frame storage. The purpose of the system was to investigate the degree to which interaction could be obtained with a pre-calculated image sequence. Thus, it was an early attempt at what is now called *Interactive Video.* Movie-Maps is very much an application-specific system. The map of the title refers to a model of a town. This space can be explored by "driving" through the town interactively, with the route and speed being specified entirely by the user. The frames in the animated sequence were obtained by filming a drive through an actual town, using four 35mm single-frame movie cameras mounted at 90° to each other in a horizontal plane. The filming was restricted to certain times of day and the route carefully planned in order to minimise discontinuities in the sequence caused by such things as abrupt lighting changes or shadow movement. The completed film was used to master the analogue videodiscs. One frame was recorded for every 10 feet of travel, which corresponds to a speed of approximately 200 miles per hour when replayed at 30 fps. The actual replay system allows animation rates of between 0 fps and 10 fps (68 mph maximum). This variable speed operation is achieved by recording one frame per videodisc "track" and by using track skipping in order to repeat frames (See Section 2.3.4). As the user "drives" down a street the frames for that view are fetched from one videodisc player. During this time the second player is instructed to search for the frame sequences corresponding to the streets reachable from the next upcoming junction. Should the user decide to make a turn at the junction the frame sequence for the new street is selected and the videodisc players reverse roles. (The implementation of this scheme is greatly helped by the gridded nature of the town's layout.) Also recorded on the videodisc are sequences of computer synthesised views of the same town that provide an alternative way of exploring the map. These were generated off-line, frame-by-frame, and recorded on the videodisc at the same time as the real sequences. One of the goals of the Movie-Maps project which was to explore the interaction between computer generated and real sequences.

Videodiscs provide access to a vast amount of stored information, but their read-only nature means that material recorded on a disc cannot be altered after manufacture. However, if this material is randomly accessible then there is the potential to view it in a variety of ways and in an order not pre-determined at the time of recording. This is the basis of interactive video. As already observed in Chapter 1 fully random frame selection is generally impossible because of seek and transfer constraints imposed by the storage device. Consequently, a more typical solution is to allow a selection only from a fixed number of sequences at a limited number of decision points (e.g., at road junctions in the Movie-Maps system). The major application areas of interactive video so far have been in education and training. One of the first commercial interactive video systems available was the Doomsday Project developed jointly by the BBC and Acorn Computers Ltd. and launched in 1986. This is based upon a Philips Laservision videodisc player and uses a microcomputer to control the system. The project aimed to re-create the famous Doomsday census of 900 years earlier by gathering and storing information about many aspects of life during the 1980s in Britain. This information takes the form of digitised stills, text and figures and short video sequences which can be browsed interactively using the microcomputer. The Laservision disc is interesting in that it stores both digital and analogue information—the digital partition is used for text, stills and figures whilst the analogue partition is used for video segments. Video sequence replay is a less central element than in the MIT system, but the project is nevertheless a good illustration of the aims of interactive video.

### 2.4.3 Systems using Non-Video-Rate Storage

When a sufficiently high data retrieval rate is not available from the storage device some action must be taken to ensure frame data can be transferred quickly enough for proper display update. Two options are considered effective: (1) modify the storage device in some way in order to increase available transfer rate; or (2) reduce bandwidth requirements by *compressing* frame data prior to recording. In the latter case a key function of the system control block of Figure 2.1 must be the restoration of frames to their original form as they are recovered from the storage device. This must be performed at frame rates (i.e., one frame every 40ms, typically). In the following subsections examples of both these approaches are given.

### Strategy 1: Modify Storage Device

The two most promising candidate classes of non-video-rate device under consideration here are magnetic and optical (i.e., digital) disc drives. As already mentioned, the most effective way of increasing transfer rate is to employ parallel transfer across multiple data heads (which may or may not imply multiple disc drives). Other techniques, such as increasing areal packing densities or spindle speed, can also be considered, but these are harder to implement and the gains are small by comparison.

#### The BBC Television Animation Store

Much of the still and animated graphics seen on television is produced by a device called a *film rostrum camera* or *animation stand.* Conceptually this is a simple piece of equipment consisting of a film camera mounted over a flat table surface upon which artwork is placed. The table can be moved horizontally in $x$ and $y$ and rotated in a horizontal plane about a vertical axis through its centre, and the camera can be raised or lowered on its vertical mounting column. The camera can operate in single-step mode, allowing an opportunity to modify the artwork between exposures, or at normal speed. With this system animation of considerable complexity can be achieved using sophisticated multi-plane and multi-exposure techniques, very much akin to those used for 2-d cartoon cel animation in the film industry.

The use of film in this process has a number of disadvantages. In particular processing of the film is necessary before the animation can be viewed. This makes it difficult during shooting to gain an idea of how the final animation will look or to correct any mistakes. Thus, putting a sequence together can be a time consuming process requiring many shoot/process/view cycles. Also the use of film inserts in a programme which is otherwise live or videotaped introduces a noticeable change in image quality[8] as well as requiring extra (telecine) facilities.

[8]This is due to a significant change in perceived flicker as the switch is made between the 24 frames per second of film and the 50 fields per second of video.

It was for these reasons that, as early as 1976, the BBC considered replacing the film camera with a TV camera as a way of producing video sequences directly. The initial requirements of a new system were that it should have at least the power and flexibility of its film based predecessor. In the event the resultant system, the Television Animation Store [Wolf82,Dure84,Kirb88], easily met and exceeded these requirements with considerable improvement in the speed of production and range of effects which could be achieved.

For the new system a number of requirements were identified. Firstly, a storage device was required which would allow frames to be written at both full rate and frame-by-frame (i.e., single captured frames). This was to be possible without recourse to expensive videotape editing facilities. Secondly, it should be possible to preview and edit sequences. Again this requires access to the storage device on a single frame basis, as well as full rate replay. Thirdly, a digital frame representation was desired in order to minimise image quality degradation over multiple edit cycles. Finally, a reasonable sequence length was to be possible in order to minimise the amount of post-processing required for the production of the final sequence. These requirements led to the adoption of a magnetic disc drive for frame storage.

The drive used was an Ampex DM331 which has an eleven platter removable disc pack. The modifications made to the drive in order to meet the required performance have been described by Durey ([Dure84]) and are summarised below. Frames in the system are represented in digital YUV component form (sampled at Y=12MHz, U,V=4MHz, 8 bits per sample[9]). This representation requires a data rate of 160Mb/s. The Ampex drive has 18 data surfaces, each with its own record/replay head. The 18 heads were split into two groups of 9 each and nine sets of record/replay logic were provided. This allowed data to be read and written in parallel on nine channels using one or other of the head groups. Also, the data rate per channel was increased by 50% to 15Mb/s, giving an overall transfer rate of 135Mb/s. In order to carry 160Mb/s frame data at this rate two bandwidth reduction techniques were applied. Firstly, data are not recorded for 11$\mu$s out of the 12$\mu$s *line* blanking periods, giving a reduction to 132.5Mb/s. During replay this information is reconstructed externally. Note that field blanking periods cannot be removed since these are required to maintain synchronisation during head/track switches at the end of a field. Secondly, frames are recorded at $7\frac{1}{2}$ bits/sample (i.e., alternate 7 and 8 bit samples) using a technique known as error feedback to account for the lost least significant bit on alternate samples. This gives a further reduction to 124.22 Mb/s and leaves 8% of the disc bandwidth available for error correction data.

A side effect of increasing the data rate per channel was that the raw bit error rate was also increased to between $10^{-6}$ and $10^{-5}$. The required rate for the application was $10^{-9}$ or better. To account for this a combined error detection/correction and concealment scheme was used. For detection/correction a Hamming double detecting/single correcting code was used. The basis of this scheme is to split data

[9]These sampling rates do not conform with the CCIR proposed standard (see Section 2.2) as this work preceded the publication of Recommendation 601.

into blocks and add protection bits per block. To save bandwidth only the top four bits of each sample were protected in this way as a fair degree of corruption of the lower order bits can be tolerated without significant image degradation. Uncorrectable samples were concealed by replacing them with values interpolated from their neighbours. The organisation of the disc is such that these neighbouring samples are always on different surfaces to the erroneous one, reducing the chance that they too will be corrupted.

The rotational speed of the disc was also altered and locked to the *field* rate so that one complete field could be transferred per revolution. Head switches and/or seeks to the next cylinder were made during the field blanking interval. Pairs of tracks are used to store frames. Each pair can be viewed as a fixed sized "frame slot" which is either allocated or free. This arrangement considerably simplifies the process of editing frame sequences. With the configuration described above, the disc had a total capacity of 815 frames (1630 fields) which gave a total replay time of just over 30 seconds. This was considered adequate for the intended application of sequence assembly and viewing.

The modified Ampex drive forms the frame storage sub-system of the Television Animation Store. In the complete system facilities are required for loading and viewing frames, video mixing and system control. Of particular interest is the control system which manages the overall operation of the Animation Store to allow frame input, editing, previewing, etc. The operation of the control system has been described by Kirby [Kirb88].

Overall control is provided by an LSI 11/23 processor. The system is menu driven using a specially designed control panel for function selection. There are five main stages required for the production of any piece of animation: (1) data entry; (2) disc space allocation; (3) frame loading; (4) animation assembly; and (5) sequence replay. During the data entry stage a specification is given of the required animation. This gives details of the required frame ordering, the display duration for each frame in the sequence, mixing effects, etc. From this specification the required amount of disc space can be allocated. It is a requirement that a sequence be held as a contiguous block of cylinders because the 1.5ms track to track seek time means that only one track may be crossed during field flyback periods. Since a number of sequences may be held on the disc at any one time, a first-fit strategy is used to find a sufficiently large free block (re-arranging if necessary to reduce fragmentation). During stage (3) frame data are written to the disc from the video rostrum camera (or other source). Because the disc only has one replay channel, any effects involving video mixing, matting, etc., from multiple frame sources must be calculated as a pre-processing stage prior to replay. This happens during stage (4) and the final sequence is written to the previously allocated block. Finally, the sequence may be previewed and, if satisfactory, recorded to videotape. To edit sequences, changes are made to the specification in stage (1) and the process of disc allocation and sequence assembly are automatically repeated. It should be noted that to save disc space only single copies of repeated frames and sub-sequences are held on disc. Frame repetition is achieved using frame buffers. In the case of sub-sequences the first and last frames are buffered, allowing time for seeking back at the end of each repetition and for skipping to the continuation point at the end of the repetition.

If the Television Animation Store was being redesigned today an obvious alternative approach would be to employ a commercially available parallel transfer drive (such as the Fujitsu drive discussed in Section 2.3.5) or multiple Winchester drives. The principal difficulty with modifying the disc drive is that it is not a trivial process, possibly requiring the assistance of its original manufacturer (as was the case for the Animation Store, for example). The advantages include improved performance and a potentially simplified system design. An example of the latter is where the disc rotational speed and bit transfer rate can be modified to introduce a fixed relationship between frame data and disc layout (e.g., one field per rotation).

#### Magneto-Optical Disc Video Recorder

An experimental system along similar lines to the BBC system has been described by Nomura *et al* [Nomu87] but employing a pair of 12 inch magneto-optical disc drives instead of a magnetic drive. Again the characteristics of the drives and media have been carefully studied and the drives modified to improve their performance and tailor them to the particular application. Similarly, YUV encoding was used, but at a slightly lower sampling frequency. The single recording surface on each of the two drives is read by two heads, one reading the inner tracks and the other the outer tracks. The transfer rate is 22Mb/s from the inner heads and 33Mb/s from the outer heads. Thus, 55Mb/s is available per drive giving a total transfer rate of 110Mb/s over all four channels. The raw bit error rate also increases as a result of modification ($10^{-5}$ is quoted at 10Mb/s transfer rate) requiring the use of Reed-Solomon error correction codes. The overall capacity of the system is claimed to be 6.5GBytes or 18,000 frames, corresponding to a replay time of 10 minutes at 30 fps.

## Strategy Two: Compress Frame Data

### (a) Software Packages

### ANIMA II and ANNTS

ANIMA II [Hack77,Magn85*a*] was developed at Ohio State University in the mid-1970's and is a good example of an early 3-D animation system. The system provides an environment for: (1) creating and manipulating (editing) polyhedral models—the objects that are animated; (2) producing a script for describing the required animation in terms of translations, rotations, etc.; (3) production of the animated sequence from the model description and script; and (4) replay of completed sequences in real-time in order that they may be viewed (or recorded on an ordinary videotape recorder). It is this final stage which is of particular interest here. The system is implemented on a PDP 11/45 minicomputer and written in assembler code.

Image files produced by stage (3) above are converted to a runlength format (see Chapter 3) and stored on a 44Mword (16 bit words) disc drive connected to the PDP 11/45. Frames are stored contiguously as this is a requirement for playback. For replay runlengths are fetched from disc over the system bus (UNIBUS) and fed to a decoder unit, via buffers in the 11/45's main memory. Resource conflicts during transfer are automatically handled by the UNIBUS priority arbitration scheme. Simple hardware in the decoder unit decodes the runlengths and produces a broadcast resolution (NTSC) signal which may be used directly or recorded.

The main problem with the replay mechanism is its limited performance. The available disc transfer rate of around 5Mb/s is a bottleneck that limits the number of runs which can be transferred per frame and hence the image complexity that can be obtained. All images are composed of constant shaded planar polygons; no other modelling primitives are available. There is no smooth shading, transparency or texturing, and aliasing was reported to be a problem. Also, images cited as being "typical", and used to evaluate system performance, occupy only about 1/4 of the available display area, with the background set to black.

Some of the shortcomings of ANIMA II were addressed in a later system called ANTTS [Csur79]. This aimed to support images of much higher visual complexity and offered a much richer set of modelling primitives: polygons, points and patches, as well as procedurally defined objects. A similar playback scheme was used, that is, generation of NTSC video from runlengths held on disc. For the more complex images, which exceeded the DMA transfer capability of the (proposed) VAX 11/780 host, runs could be transferred from an external core memory. Such sequences had to fit into the available memory capacity of around 6Mbytes.

### The Differential Compiler

A different image compression approach is used in a more recent system described by Denber and Turner [Denb86]. In this system a technique called *Frame Replenishment Coding* [Prat78] is used that achieves compression by exploiting *temporal coherence* within animated sequences. Temporal coherence is a property of many sequences that for any given frame the following frame is likely to be substantially the same; indeed, as pointed out in [Denb86], such a property is a requisite for smooth animation. Frame replenishment coding works by recording *differences* between frames rather than the frames themselves in the expectation that the former will require less storage.

The system is implemented in Interlisp and runs on a Xerox 1108 workstation. Facilities are provided for entering and editing of frame data, but these are not relevant to this discussion. More interesting are the elements which process the frame data—the *differential compiler* and the *real-time interpreter.*

The 1108 workstation has a bitmap display which means that all animation is restricted to binary frame data. Frames can be any size up to a maximum determined by the resolution of the display (1024×808 bits). Encoding of frame data is performed by the differential compiler which operates in two phases. In the first phase the differences between each pair of frames are found. This comparison is done in terms of *primitive blocks* which are 16×16 sub-arrays of bits. For each pair of corresponding blocks if any difference is found then the block from the later frame is saved, along with its location, on a list of changes for that frame. At the end of the pass the original frames are discarded as the change list is sufficient to enable a frame to be created from its predecessor (which in turn can be created from its predecessor, and so on). The change list may contain hundreds of primitive blocks and for so for efficiency a second phase is invoked in which adjacent blocks are merged to form larger blocks. This merging process has two stages: *primary* and *secondary merging*. The result of this phase is a smaller set of rectangular update blocks which is known to be sufficiently small to allow all the updates to be applied within the one frame time available.

The real-time interpreter facilitates replay of completed sequences. The change list is held in memory and the 1108's bitblt operation used to apply the set of updates for each frame. An artificial delay is added when a frame can be updated in less than one frame time in order that the frame rate is kept constant. The delay can be specified on a per-frame basis to allow for a variable frame rate if required (this can be used to show title frames, for example). Note that the real-time interpreter is very simple because all the hard work has been done during the compilation stage. This is a general design principle for a real-time playback system: if there is to be a disparity between the coding and decoding stages then effort should be concentrated in the coding stage in order to simplify the (time-critical) decoding stage.

Performance restrictions are imposed on this animation system by limitations associated with the 1108 workstation, and in particular its bitmapped screen, bitblt unit and the memory size. The 1-bit deep screen restricts the type of image to black and white drawing. The bitblt operation has a limited bandwidth which limits the number of changes that can be made per frame and hence the complexity of the animation. The amount of real memory (3.5MB) places a constraint on overall sequence length. This can be extended by including virtual memory, but to do so affects the achievable frame rate because of paging overheads.

### (b) Hardware Approaches

Many of the performance limitations of the two approaches just outlined can be overcome with the provision of custom hardware for critical stages in the replay process. In particular, more powerful image compression techniques can be implemented using custom frame decoding hardware. This realisation was one of the motivations for the design of a new hardware based animation system; it is this system which is discussed and analysed in the remainder of this dissertation. In Section 2.5 the aims of this research are set out. Recently a system based upon similar lines, but with different objectives, has been announced, and this system is discussed first.

### Digital Video Interactive

Digital Video Interactive (DVI) was developed by RCA (prior to its takeover by General Electric) at its David Sarnoff Research Laboratory in Princeton, New Jersey. The aim was to produce a system which could replay video from a CD-ROM, that is, a 5¼ inch digital read-only optical disc drive. There is a considerable market for a system capable of producing long-play, high quality video from a compact, inexpensive and mass-reproducible medium. The principle application areas targeted by General Electric, along with its new collaborators (Microsoft, Intel and Lotus), are those of education, training, entertainment and retail sales. A similar system, known as CD-I, is under joint development by Sony and Philips, although this is intended to be purely a consumer product.

Storing and retrieving video rate data from a CD-ROM represents a considerable research challenge because of the relatively low transfer rate available (typically 150KB/s). A high degree of compression is also required to achieve a long playing time. In DVI a very complex *coding* algorithm is used in order that decoding may be done relatively easily and therefore quickly. The complexity of the coding algorithm is such that a high performance parallel processor (a Meiko Computing Surface) has to be used to achieve compression of sequences in a reasonable time. The details of the coding algorithm have not been published, although it is believed to be a form of Adaptive Differential PCM (ADPCM) (see [Netr80] or [Jain81]). The scheme makes considerable use of inter-frame (temporal) coherence. The claimed compression ratio is 120:1 for motion video, which represents an incredibly high degree of reduction. This must be placed in context, however. In particular, the figure quoted assumes source frame of 512×400 pixels at 24 bits/pixel and 30 fps. But a number of bandwidth techniques can easily be applied prior to compression which would give considerable savings without significantly affecting image quality. For example, DVI uses YUV component coding for pixel values. The colour components (U,V), however, are sub-sampled by a factor of four which gives a reduction to 12 bits/pixel with negligible loss in image quality. Another technique, used during replay, is to generate both fields of a frame from the *same* data. This yields a further halving of bandwidth requirements.

The video decoding hardware is based around a two chip set known as the Video Display Processor (VDP). One of these (VDP1) is a 12.5 MIPS microcoded processor which implements the decompression algorithm. Decoded frames are written to a video RAM (VRAM). The second chip (VDP2) reads this memory and refreshes the screen. Host access to the VRAM is available via VDP1 for the purposes of overlaying text and graphics onto the video data.

One disadvantage of the system is that it is effectively read-only, even though writable media (such as magnetic disc) could be substituted for the CD-ROM. This is because the very high complexity of the compression algorithm precludes individuals coding their own sequences. It is currently proposed that video coding be offered as a service at a number of centres in the world. A tape containing the sequence to be recorded would be sent to one of these centres and a recorded disc returned. Work is under way on a simplified real-time encoding implementation based around the VDP1 processor. However, this will give poorer results than those obtained by the full algorithm, and is intended mainly for previewing prior to final sequence compilation. (Sequences generated in this way are replayed at a lower frame rate and have lower reconstructed image quality.) A second problem is caused by the coding scheme itself due to the stringent demands for high compression placed upon it. In certain situations, such as the rapid motion during a camera zoom, there is a noticeable degree of image degradation introduced by the coding algorithm.

## 2.5 Motivations for a New Animation System

The shortcomings of previous stored-frame animation systems together with a number of other observations formed the motivation for the design and construction of a new hardware animation system. These observations can be summarised as follows:

1. Many previous animation systems suffer severe performance limitations due to their having been implemented on top of an existing machine architecture, often a general purpose processor. Further, the majority of previous systems have been implemented entirely in software, with no specialised hardware support whatsoever. These observations suggest that considerable performance benefits may be gained from the judicial use of custom hardware within any new animation system design, a view attested to by the performance of the BBC Television Animation Store.

2. No previous system had made a significant attempt at improving system performance through the use of a hardware-based implementation of a compression algorithm as a central architectural feature[10]. Such an approach contrasts with the alternative strategy of storage device modification, examples of which already existed (see Section 2.4.3).

3. Real-time playback is not a new technique, but from time to time it is valuable to re-evaluate ideas in the light of technological improvements. Since the design of many early systems considerable advances have been made in display processor architectures, semiconductor technologies and memory systems (both primary and secondary). With secondary storage in particular, a number of new and promising technologies have emerged over recent years—optical storage is a good example. The observation was made that the capacities and transfer rates of current-generation devices are such that an implementation of an animation system becomes feasible with only a small to moderate degree of image compression.

[10]This was the case at the time of the new system's inception. It should be emphasised that Digital Video Interactive (DVI) was not announced until 1987, two years after the start of work on the new system.

4. A new and powerful display processor called Garland [King88] was under development within the graphics group at the University of Cambridge Computer Laboratory. Garland is designed to form a graphical back-end to a larger system and provides support, in the form of video-rate communication links, for a variety of front-end processors. A hardware stored-frame animation system was an obvious candidate for one such front-end processor.

5. The observation was made that whilst graphics workstations are universally employed for the generation of frame sequences for computer animation, such systems are unsuited to frame replay. Difficulties have been experienced by many people whilst attempting to produce animation in such an environment. A number of examples can be cited from amongst the author's own colleagues including those working upon the development of new animation techniques themselves (e.g. the work of Pullen [Pull87]) as well as others interested in the animation of the results of more general work on image synthesis techniques. The traditional approach to the frame-by-frame animation, through the use of film or videotape, has a number of drawbacks for the production of computer animation in a workstation-based environment:

   (a) It requires expensive additional facilities, namely a film or stop-frame video recorder. These may not be available at all or, when they are, often represent shared central resources with limited accessibility.

   (b) The recording and replay environment is disjoint from the development environment. Whilst this is fine for final sequence production it is less acceptable for the interactive development of animation. With film, which requires developing and printing, the turn-around time is excessive, and the general drawbacks of both media—incompatible (non-digital) representation, difficulties with editing, etc—have already been discussed.

These observations led to the design of a new system with the following framework and objectives:

1. To investigate the extent to which performance limitations associated with many previous animation systems can be overcome by the development of a custom real-time playback architecture. The target performance is the replay of medium to high resolution multiplane (colour/greyscale) raster frames at full frame rate (25 fps).

2. To employ an unmodified, non-video-rate storage device in conjunction with a hardware based implementation of a frame compression algorithm. Such an approach has previously received little attention yet offers a number of potential advantages over alternative strategies such as storage device modification.

3. To exploit the functionality and graphical processing power of Garland by designing the system in the form of a Garland front-end processor.

4. To develop a workstation-based system that provides an integrated environment for the interactive development of animated sequences. This contrasts both with current approaches (i.e., film/videotape) that only allow the production of final sequences and with systems designed for mass distribution of material (e.g. videodiscs). An interactive system allows a newly produced sequence to be viewed immediately, then modified if desired and replayed again to check the changes. This edit-view cycle is repeated for as long as it takes to produce the desired animation. Such an approach is not possible in systems for which the storage medium requires either post-processing (e.g., some forms of optical disc) or a mastering process (e.g., videodisc) before the recorded sequence can be viewed. Similarly, interactive working is not possible with the DVI system because the complexity of the coding algorithm precludes users compressing their own sequences locally in a reasonable time.

## 2.6 The Basis for a New System

### 2.6.1 Storage Device

A Winchester disc drive was selected as the most appropriate form of storage device for the proposed system. This was for three reasons. Firstly, of all the potential frame storage devices considered in Section 2.3, magnetic storage most closely matched the requirements of the new system. This can be seen by comparing the objectives above with the properties of the candidate devices as discussed in Section 2.3. The advantages of magnetic storage are: (1) re-usability; (2) immediate play-back (no post-processing); (3) locally writable; (4) supports frame-by-frame operation; (5) random access; and (6) direct digital representation. Secondly, the use of an unmodified serial magnetic disc for video-rate frame replay has previously received little attention and was therefore felt to justify further investigation. Thirdly, the provision of magnetic (Winchester) storage within current workstation systems is commonplace. Consequently, a suitable drive may already be available in such an environment without the need to supply an additional frame storage device.

The principal disadvantages of a magnetic disc for frame storage are capacity and transfer rate; these are both addressed by the use of image compression.

### 2.6.2 Image Compression Strategy

The use of image compression was adopted for two reasons. Firstly, it is *required* for the chosen frame-storage device in order to overcome limited transfer bandwidth and storage capacity. Secondly, it represents a largely unexplored approach to animation system design. It is always advantageous to apply image compression to a sequence prior to recording, even when it isn't necessary for increasing the effective transfer rate. This is because of the reduction in storage requirements and corresponding increase in maximum sequence length. A modest 2:1 compression, for example, doubles the replay time. When the required bandwidth is not available, compression represents a less expensive approach to that of device modification. The BBC is known to have considered replicating its Television Animation Store but was unable to do so because of the cost involved. The cost of a commercial PTD system (approximately £27,000 in 1988) is considerably higher than the total cost of an equivalent capacity sequential disc plus frame decoding logic.

The choice of a compression technique must be based upon the requirements of the application and this is the topic of discussion for the next chapter.

# Chapter 3

# Image Compression Considerations

## 3.1 Introduction

Development of techniques for image compression began in the early 1950s and the field has received continuous attention through to the present day. Recently there has been a heightening of interest as new applications have come to the fore. Examples of these include the provision of integrated services (voice, video and data) on local and wide area networks, remote sensing via satellite and work on a broadcast high definition television (HDTV) service. All of these require the use of image compression to reduce bandwidth requirements. Also, in recent years a number of new and interesting "second generation" compression techniques have been investigated which have hold considerable promise for substantially improved compression performance in the future.

Image compression is encompassed by the more general field of data compression and is commonly treated from an engineering point of view using techniques from the signal processing and communications fields. These fields have their mathematical basis in *Information Theory*, whose foundations can be traced back to the classical 1948 paper by Shannon [Shan48]. The discussion in this chapter is concerned only with image compression and does not consider the numerous other forms of image processing operation which are in common use. Such operations include edge detection, region finding and feature extraction, and are employed in applications as diverse as artificial intelligence (AI) and automated cartography. These techniques are covered in depth in the many texts devoted to the subject; for example, [Gonz77,Prat78,Prat79,Rose82] are "standard" works.

In the Animation Server image compression was adopted as a means of overcoming the storage and data transfer limitations associated with the chosen frame storage device. In selecting a compression scheme it is necessary to give thought to such questions as how much image compression per frame might be reasonably expected (i.e., what theoretical and practical limits are there), how complex do algorithms need to be to achieve a given degree of compression and what special or unique features of this application may be exploited. In the space of a single chapter it is impossible to do justice to the vast field of image compression and the multitude of techniques which have been developed (one author reports that a thousand papers a year are published on the subject). Rather, the aim is to present a general overview of the field with the hope of providing sufficient context for the description of the compression strategy actually adopted (see Chapter 4). In the next section a brief description of the relevant information theory is presented, followed by some general considerations for an image compression scheme and mention of the main approaches which are currently used or under development. Note that in the following the terms "compression" and "coding" are used interchangeably.

## 3.2 Information Theory and its Application to Image Compression

Given an image $F$ of resolution $X$ by $Y$ pixels by $D$ bits/pixel, it would appear at first glance that exactly $X \times Y \times D$ bits are required to represent the image. For an image composed of bits selected completely at random this would be true. However, most images are not random and consequently there is usually some structure and inter-relationship between bits. It is the exploitation of this structure that provides the foundations of image compression.

In information theory terms the image $F$ is an example of an *information source* (it is the source for later processing involving compression, transmission, etc.). Consider another information source, namely ordinary English text. Again, the source is not random and without structure. This can be seen by looking at any typical document. Firstly, the letters do not occur with equal probability; "a" is much more likely than "q" and "the" more likely than "eht", for example. Secondly, elements are not unrelated; given a "q" there is a very high chance that the following character will be a "u" (a property of the English language). As for images, this structure can be exploited to achieve compression.

In general terms an information source is modelled as a sequence of *symbols* drawn from a finite set $\mathcal{S}$ with elements denoted by $s$. In practice symbols are typically things such as letters from an alphabet or pixels in an image. The sequence produced represents successive values of a random variable $S$, and associated with each symbol $s$ is a probability $P(s)$ which says how likely it is that $S$ will take the value $s$; that is, $P(s) = Prob.\{S = s\}, s \in \mathcal{S}$. The arrival of a symbol $s$ contributes an amount of *information* related to its probability $P(s)$. Unlikely symbols carry more information than more common ones since their arrival comes as a greater "surprise". In information theory the term *information* has a specific and precisely defined meaning (it is a measurable quantity) and the concept of information is one of fundamental importance. The information contributed by the arrival of a symbol $s$ is defined as[1]:

[1]The use of base 2 in the log function of this and later equations gives the information rate measure in *bits of information*. Other bases are possible by multiplying by the appropriate constant conversion factor.

$$I(s) = \log_2\left(\frac{1}{P(s)}\right)$$

The amount of information contributed on *average* by a symbol $s$ is given by the quantity $P(s)I(s)$. Taking the average over the entire set of symbols $\mathcal{S}$ gives the average information rate or *entropy*:

$$H(S) = \sum_{s\in\mathcal{S}} P(s)\log_2\left(\frac{1}{P(s)}\right)$$

$H(S)$ is called the *zeroth order* entropy of the random variable $S$; it is a measure of the average information given by a sample $S = s$ without consideration of the contributions from other samples due to statistical dependence. For a more accurate measure of entropy it is necessary to consider more than one symbol at a time. Consider two random variables $S_1$, $S_2$, where:

$$P(s_1, s_2) = Pr\{S_1 = s_1, S_2 = s_2\}, s_1, s_2 \in \mathcal{S}.$$

The *joint* entropy of $S_1$, $S_2$ is then given by:

$$H(S_1, S_2) = \sum_{s_1,s_2\in\mathcal{S}} P(s_1, s_2)\log_2\left(\frac{1}{P(s_1, s_2)}\right)$$

This is easily generalised for a finite sequence of random variables $\{S_k\}_{k=1}^{N}$, denoted $H(S_1, S_2, S_3, \ldots, S_N)$. The entropy of the source, $\mathrm{H}_s$ is then defined by:

$$H_s = \lim_{N\to\infty} \frac{1}{N} H(S_1, S_2, S_3, \ldots, S_N)$$

Entropy is an important concept because it allows the information content of a source (e.g., an image) to be determined and it is this which defines the amount of compression which can be obtained. More precisely, ***Shannon's Noiseless Coding Theorem*** states:

$$H_s \leq \bar{L} < H_s + \frac{1}{n}$$

This says that for reversible coding the average number of bits per source symbol required for the coding ($\bar{L}$) cannot be less than the entropy of the source, but for sufficiently large order ($n$) it is possible to come arbitrarily close to that value. Thus, the theorem places a fundamental limit on the number of bits required to represent a source. Unfortunately, it does not say how to reach this limit in practice.

Figure 3.1 shows the situation described so far with a source coder which compresses the source to $H_s$ (in the ideal case) and a corresponding decoder which reconstructs the original source from the compressed form. Compression is achieved because $\bar{L} < D$. In practice the entropy of real-world cases is difficult to measure

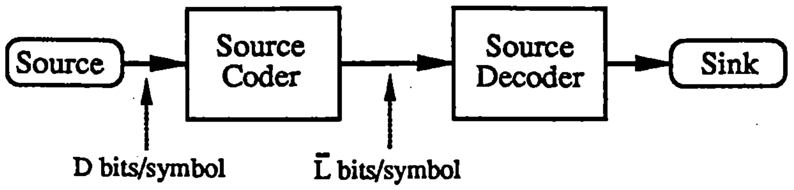


Figure 3.1: Source coding/decoding model

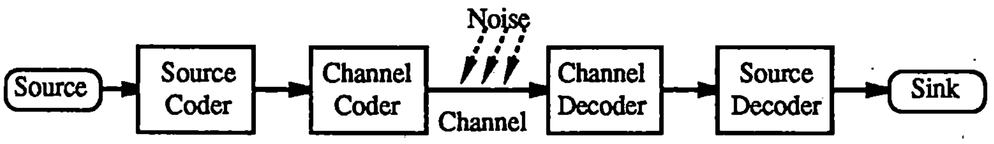


Figure 3.2: Source coding/decoding over a noisy channel

for anything more than the very lowest orders of entropy because of the difficulty of finding an accurate source model. The entropy of English has been estimated at between one and two bits per letter [Wyne81] (c.f. at least five bits which are required to represent the 26 letters directly), and the zeroth, first and second order entropies of 6-bit/pixel monochrome images have been estimated at 4.4, 1.9 and 1.5 bits/pixel, respectively [Jain81].

It is possible to code a source with $R$ bits/symbol where $R < H_s$, but not without introducing distortion into the reconstructed stream. In this case *Rate Distortion Theory* provides a mechanism for reasoning about the distortion produced and how it can be minimised for a given rate $R$. (Note that the previous discussion about error-free coding is just a special case where $R \geq H_s$). Wyner discusses this generalisation of the source coding problem in more detail in [Wyne81].

Compression is achieved by the source coder of Figure 3.1 by exploiting statistical knowledge of the source in order to remove *redundancy*. The model of Figure 3.1 assumes that the symbols sent from the coder arrive unchanged at the decoder; that is, that there is a *noiseless channel* connecting the encoder and decoder. This is not usually the case in practice and a certain proportion of symbols will be corrupted during transit due to channel noise. Detection and correction of such corruption is difficult or impossible for a stream containing little or no redundancy. For this reason it is usual to add some redundancy back into the coded stream in a controlled way for the purposes of transmission or storage of coded streams. This leads to the idea of channel coding as illustrated in Figure 3.2. Clearly the amount of redundancy added by the channel coder must be less than that removed by the source coder if a net compression is to be achieved. The theoretical basis for this is provided by Shannon's main theorem which states the requirements for reliable transmission over a noisy channel at rates arbitrarily close to the channel capacity. The details of this are beyond the scope of the current discussion and for more information the reader is referred to Wyner [Wyne81] or one of the many texts on information theory (e.g., [Hamm80]).

## 3.3 Coding Framework and Considerations

In examining the gamut of current and experimental compression schemes a number of broad differences emerge. These differences, together with a number of general observations, are outlined below and provide a useful framework for the classification and discussion which follows.

### 3.3.1 General Differences

1. *Spatial (Causal) versus Transform (Non-causal) Domain Coding:* A distinction can be made on the space in which a method is applied. Spatial coding methods operate on the original pixel data whereas transform coding methods operate on a set of transform coefficients derived from this data. A third class of algorithms uses a combination of both and is generally referred to as *hybrid* coding.

2. *Exact (lossless) versus Approximate Coding:* In exact coding the image reconstructed by the decoder is identical to the original image seen by the coder. A higher degree of compression can be obtained when this constraint of exactness can be relaxed and approximate coding used. Here the decoder is incapable of reconstructing an exact copy of the original because some image information is discarded by the coder during the compression process. Generally the aim is to minimise the distortion produced in the reconstructed image by the loss of this information. Unfortunately, there is no really effective distortion measure when, as is often the case, the image receiver is a human observer (as opposed to some form of automatic image analysis system). In this situation, objective measures such as mean-square error[2] are of little use because they fail to account properly for the properties of the HVS. Compression schemes which exploit properties of the HVS can yield considerable improvement in the degree of compression obtained for little subjective loss of image quality. This is achieved by preserving image features which are important in the recognition process at the expense of other information. There has been considerable interest in schemes of this type during recent years. Many properties of the HVS are now better understood and those relevant to image compression are described by several authors, for example [Schr67,Conn72,Netr80,Hask81,Kunt85]. Unfortunately, however, there is no simple concise HVS model which allows the performance of these schemes to be evaluated objectively and it is usually necessary to resort to subjective testing. Generally, this involves asking subjects to judge the quality of images and the effects of distortions under carefully controlled conditions of lighting, viewing distance, etc. Typical impairment scales used, which can be absolute or comparative, are illustrated in Table 3.1.

[2]The average mean-square error between two $N \times M$ images, $\{u_{i,j}\}$ and $\{u\prime_{i,j}\}$, is determined experimentally [Jain81] as follows: $e^2_{ms} \simeq \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}(u_{i,j} - u\prime_{i,j})^2$.

| | | | | | |
|---|---|---|---|---|---|
| | | | | 3 | Much better |
| 5 | Excellent | 5 | Imperceptible | 2 | Better |
| 4 | Good | 4 | Perceptible but not annoying | 1 | Slightly better |
| 3 | Fair | 3 | Slightly annoying | 0 | Same |
| 2 | Poor | 2 | Annoying | -1 | Slightly worse |
| 1 | Bad | 1 | Very Annoying | -2 | Worse |
| | | | | -3 | Much worse |

Table 3.1: Typical Impairment Scales used for Subjective Image Testing

3. *Non-adaptive versus Adaptive Coding:* Improvements in compression performance can generally be obtained by making a scheme sensitive to changes in source characteristics as coding progresses. This improvement is generally obtained at the expense of increased coder and/or decoder complexity. Adaptive coding is commonly used for schemes which are only able to make a single pass over the source, as is the case for many real-time compression schemes, for example. In this situation it is difficult to have additional passes for the purposes of gathering statistics about the properties of the source.

4. *Intra-frame (spatial) versus Inter-frame (temporal/spatiotemporal):* A common image coding situation is that of processing images which are not independent but represent individual frames from a longer time-varying sequence. Typically in such sequences there is considerable *temporal coherence* between successive frames, especially at normal frame rates (e.g., 25 fps). Inter-frame techniques aim to exploit this redundancy. This contrasts with intra-frame techniques which exploit the spatial coherence within individual frames.

## 3.3.2 Other Considerations

Jain [Jain81] considers that three factors determine the efficiency of a compression algorithm: its data compression ability, the level of distortion produced and its implementation complexity.

In this dissertation compression ability is expressed solely in terms of a *compression ratio.* This defines the ratio of the uncompressed and compressed images. For example, a compression ratio of 2:1 indicates that the image data requirements are halved by compression.

Distortion has already been discussed in relation to approximate coding.

Implementation complexity is of particular relevance to hardware realisations of compression schemes. In addition to the raw computational cost of any algorithm it is also necessary to consider practical factors. For example, it is generally preferable if the compressed form of an image is *instantly decodable*; that is, pixels can be decoded sequentially as the stream is read, without first (potentially) having to scan the entire coded image[3]. Similarly, it might be desirable that the decoding

[3]This is a more rigorous form of *unique decodability*. Unique decodability states that there must be a single, unambiguous decoded form of the coded stream [Hamm80].

of the image occurs in scanline order. The effects of such factors can have a considerable influence on the implementational complexity (or even the feasibility) of a (de-)coder.

### 3.3.3 Application Tailoring

Further improvement in coding efficiency can be obtained if it is possible to exploit known properties of the particular application. Examples of such properties include the following:

- *Image source:* Images to be compressed may be generated from a number of sources, including scanners, video cameras and via image synthesis. A common situation is for the application to be interested in images from only one source and in such cases the properties of that source can be exploited by the coding scheme. The properties of sources vary considerably. For example, a video source introduces a slight blurring for moving objects due to an optical integration effect within the television camera whereas such blurring is not normally present in a computer synthesised view of the same scene. Similarly, a video source introduces noise into the image which is not present in the computer synthesised form. The optimisation of a coding algorithm to the properties of a particular source can yield considerable savings.

- *Image type:* Similarly the source might be confined to one particular type of scene. For example, in a video-conferencing application the image is normally restricted to that of a head and shoulders view of a person. Both the camera and background remain static. In such a scene rapid movement or sudden change is rare and when it does occur it is usually acceptable for this to be transmitted at a lower quality than that for slower change.

  With computer synthesised scenes images may be restricted to one type of modelling primitive or rendering technique (e.g. only flat shaded polygonal models).

- *Operational considerations:* Different applications impose different constraints upon the coding and decoding processes. For example, with real-time playback systems it is the replay (i.e., decoding) which is the time-critical operation. In such cases it is common to employ a non-symmetrical coding/decoding arrangement whereby more complexity is placed in the coder in order to simplify the decoder. In a symmetrical application (such as video-conferencing) such an arrangement is not possible.

## 3.4 Coding Approaches

The space available in this section allows only the briefest outline of the main techniques which are used for image compression. Further details can be found in the extensive literature for this field, references to which are given throughout the following discussion. In addition, there have been several surveys published which chronical the work done during the last four decades. The best known of these are by Netravali and Limb [Netr80], Jain [Jain81] and Musmann *et al* [Musm85]. The field is also covered by many of the standard texts on image processing (e.g. Rosenfeld and Kak [Rose82]).

### 3.4.1 Direct Methods

Substantial compression can often be achieved directly by obvious and simple means such as pixel quantisation (dropping some bits from each pixel) or spatial and/or temporal sub-sampling (either dropping pixels from within individual frames or between successive frames). This constitutes a reduction in the resolution of the source. In many cases the source representation is at a higher resolution than is actually required for an effective representation. For example, in graphics workstations it is convenient (and common) to employ a 24 bit image representation, with 8 bits for each of the three primaries red, green and blue (RGB). However, it turns out that a total of 8 bits is sufficient for a reasonable quality representation of colour images [Cowl84]. Similarly, 8 bits are often used for monochrome images, whereas 6 bits can be used with little or no loss in subjective image quality. Both of these values depend upon viewing distance and the resolution of the display [Cowl84].

In a similar way the spatial or temporal resolution may be greater than that needed for the final representation and can be reduced directly by some form of sub-sampling. In the temporal domain, for example, frame repeating may be used to yield an immediate halving of bandwidth requirements; an alternative is scanline interlacing (as used in broadcast television) which achieves the same result and is subjectively more acceptable. Of course any compression achieved in this way must take into account factors such as image flicker and refresh rates (as discussed in Section 1.4). These direct sub-sampling methods can be extended in a number of ways and form the basis for interpolative/extrapolative coding discussed below.

Generally, such direct reductions are not implemented in an *ad hoc* way but take some consideration of the properties of the source and receiver (typically the HVS). For example, to quantise from 24 to 8 bits a technique such as Heckbert's *median cut* [Heck82] might be used. This takes account of the distribution of colours within the original $2^{24}$ point colour space to select the best $2^8$ colours. The HVS is more sensitive to green than it is to either blue or red and it is usual to reflect this fact in the allocation of bits to the primaries. A typical assignment for 8 bits is 3:3:2 for R:G:B. Many other properties of the HVS can be exploited to achieve compression. For example, the HVS is more sensitive to luminance information than it is to colour. This knowledge is exploited in CCIR Recommendation 601 which defines a digital representation for video [Kret85]. Here the colour information is typically sampled at half the spatial resolution of the luminance component, with little or no perceived loss of quality.

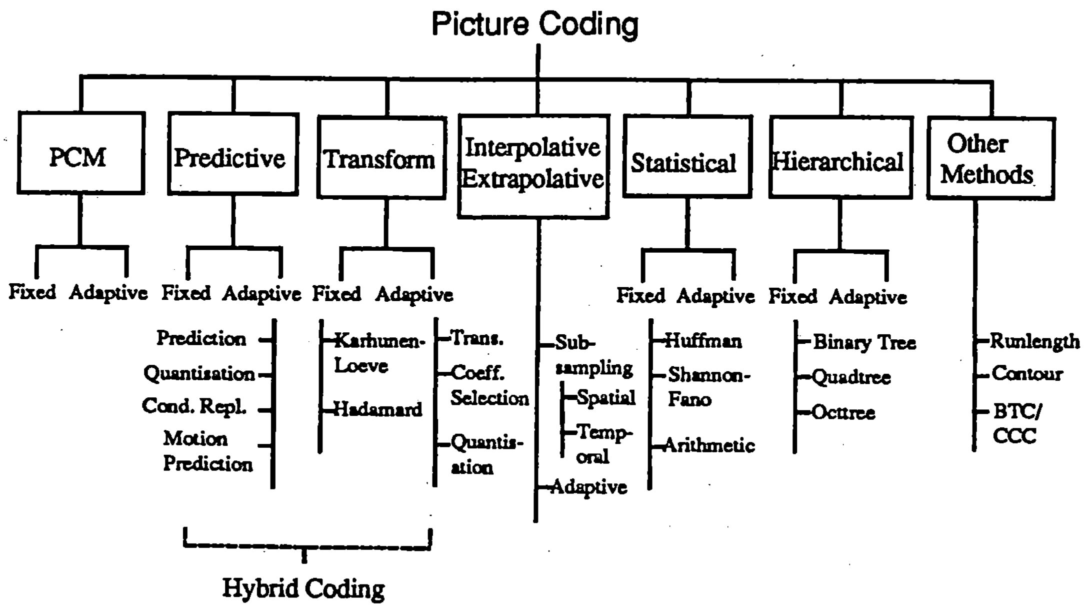


Figure 3.3: A Classification of image coding techniques

## 3.4.2 Principal Approaches

The main coding approaches which have been developed are shown in the Figure 3.3. This classification is based on that used by Netravali and Limb [Netr80] and represent what Kunt *et al* term "first generation" coding techniques. These methods are outlined briefly below (and for further information the reader is referred to the given individual references).

1. *Pulse Code Modulation (PCM):* PCM coding refers simply to the process of producing a digital representation of an analogue waveform. PCM techniques were developed before the advent of digital processing and have been widely applied to the coding of television signals.

2. *Predictive Coding:* [Netr80,Jain81] In predictive coding schemes the coder scans the image and attempts to predict the value of upcoming pixels by using information derived from pixels already seen. The difference between each prediction and the true pixel value is taken (to give an error term) and coded for transmission to the decoder. The decoder employs an identical prediction function and adds each transmitted error term to the corresponding prediction in order recover original pixel values. The most common form of predictive coding is known as Differential PCM (DPCM).

3. *Transform Coding:* [Wint72] In transform coding the image is divided into a number of small blocks of $n \times n$ pixels and each block is independently transformed into a set of $n^2$ transform coefficients. The purpose of the transform is to redistribute the information content of the block such that the maximum

amount of "energy" is packed into the smallest number of coefficients. Compression is obtained by discarding those coefficients with the least energy and by representing the remaining coefficients with an accuracy proportional to their contribution to the information content of the block. Decoding is performed by applying the inverse transformation to coefficients received from the coder. Any transform which is used must therefore be reversible. An optimal transform, known as the Hotelling or Karhunen-Loeve transform, does exist, but for various reasons this is difficult to implement. A number of other transforms have been developed, the best known being the discrete Fourier, discrete cosine, Hadamard and Haar transforms.

4. *Interpolative Coding:* [Hask72] In interpolative coding only a proportion of the pixels in the source is transmitted by the coder and during decoding interpolation is used to generate (approximations to) the missing pixels from neighbouring transmitted pixels. Transmitted pixels are selected using some form of spatial and/or temporal sub-sampling.

5. *Statistical Coding:* Statistical coding techniques obtain compression by directly utilising knowledge about the statistics of a source. Many of the methods used are familiar from their application to non-image sources. Perhaps the best known of these is the simple and elegant scheme developed by D.A Huffman and described in his now classical 1952 paper [Huff52]. A lesser known, but slightly more efficient, technique is that of arithmetic coding. This technique has been described by various authors, including, for example, [Lang84,Witt87,Corm87].

6. *Hierarchical Coding:* [Same84,Cohe85] Hierarchical coding is used here to refer to the application of hierarchical data structures, notably binary trees, quadtrees and octtrees, to the task of image compression. Compression is obtained by a recursive decomposition which directly exploits 2-d (spatial) or 3-d (spatio-temporal) coherence within the source image data. Of these structures, the quadtree is the most widely used for the purposes of image compression. For effective compression a *semi-pointered* [Stew86,Will88] or *pointerless* representation [Garg82,Oliv83] must be used since any pointer-based representation is too cumbersome. It is also possible to consider optimisations of these codes [Same85,Wood84,Aned88].

7. *Other Coding Approaches:* Numerous other coding schemes have been developed which do not fit neatly into any of the above categories. Mention here is made of only one, *runlength coding*, perhaps the simplest and best known of all coding methods. With this technique an image is scanned in some order (not necessarily scanline order [Cole87]) and repeated successive occurrences of a pixel value replaced by a single instance of that value and a counter to indicate the number of pixels in the run.

### 3.4.3 Coding Optimisations

Considerable improvement in coding efficiency can be gained through extensions to the basic coding schemes described above. Some of the approaches which have been tried are discussed by Musmann *et al* in [Musm85]. Two techniques which have received particular attention in recent years are adaptive coding and motion compensation.

#### Adaptive Coding

Adaptation can be introduced into any particular coding scheme in a number of ways. For example, with predictive coding both the prediction and the quantiser functions can be made adaptive. A number of adaptive schemes are discussed in [Musm85]. Quantiser designs have also been improved by optimising them subjectively rather than statistically [Netr80,Musm85]. Similar changes can be applied to other coding techniques. For example, Knee *et al* [Knee88] discuss some adaptive sub-sampling strategies for use with interpolative coding.

#### Motion Compensation

In motion compensated compression schemes an attempt is made to track movement within a frame sequence and to use this information to guide the coding process. If this is done well considerable gains in compression efficiency can be achieved. Unfortunately, the problem of accurately tracking motion is generally difficult. Even so, much effort has been invested in investigating the application of motion compensation to coding schemes, most notably for predictive coding. In [Musm85] mathematical models for dispacement estimation are discussed and two classes of estimation algorithm, known as block matching and pixel recursive, are examined.

### 3.4.4 Second Generation Techniques

For any image the degree of lossless coding which can be achieved is fundamentally limited by its entropy as discussed at the beginning of this chapter. Compression beyond this point can only be achieved with the use of approximate coding techniques which necessarily introduce some distortion into the reconstructed image. The aim of any approximating method is to minimise this distortion to an acceptable level. Kunt *et al* [Kunt85] argue that this can only be achieved by using compression techniques which are based upon known properties of the HVS (i.e., the most usual receiver) rather than on information theory. They term techniques operating in this way "second generation". In [Kunt85] two main classes of second generation technique are described: local operator based techniques and colour-texture oriented techniques. In the latter class, for example, edges within an image are coded separately and more accurately from the regions they enclose (reflecting the fact that the HVS attaches more importance to edge information). With such approaches very much higher compression rates are claimed.

Another way in which coding improvements are likely to be obtained in the future is with feature coding. This is based upon the idea that an image which is difficult to describe at the low (pixel) level may have a simple and compact higher level representation. If a high level description can be extracted from the low level image data then the potential exists for a much more efficient coding. For example, for textual images (as processed by FAX machines, for example) optical character recognition represents the ultimate coding scheme. Parameter coding is discussed in [Hask81]. An example of a coding scheme for image data is IFS coding [Barn88] which is designed to allow very efficient coding of certain fractal shapes. At the low level the complexity of such shapes is difficult to describe. Unfortunately, the required process of mapping low level constructs into high level descriptions is not well understood (it is an AI problem of pattern recognition) and there are no readily available solutions.

## 3.5 Compression Results

Following such a general treatment of image compression techniques it would be meaningless to attempt to cite specific compression factors for individual schemes. The preceding coding scheme descriptions are illustrative only and within each category a continuous range of implementations is possible, ranging from the very simple to the very sophisticated. Consequently, any compression figures cited are only meaningful when considered in relation to the framework in which they were obtained. This framework includes factors such as the test images employed, the subjective/objective measures of image quality used, the source models adopted, and so on. Unfortunately, no standard framework exists for this and consequently a meaningful comparison of the results quoted in the multitude of published material is virtually impossible. For example, there isn't even a commonly agreed set of test images that can be used[4]. Also any choice of compression scheme must take into consideration the intended application—it is insufficient to consider compression ratios alone. For example, many algorithms give excellent results for static images (very high compression ratios for good image quality) but are useless for coding image sequences because coding artifacts become visible during replay. Artifacts include such things as unexpected and distracting changes of colour or shade. This is particularly true of approximate coding schemes.

Nevertheless, it is possible to indicate the range of typical performances obtained. Generally speaking, compression factors achieved vary between 1:1 to around 50:1. The value of 10:1 can be regarded as a watershed, with the majority of schemes achieving results below this level. Compression factors above 10:1 are generally only achieved with the use of the most sophisticated coding schemes or the second generation coding methods discussed by Kunt. From this it can be seen that some of the compression results cited, such as the 10,000:1 claimed by Barnsley and Sloan [Barn88], are clearly unreasonable when talking about general

[4]An exception to this is the standard set of "typical" documents developed for document processing applications (e.g. FAX coding) by CCITT Study Group XIV.

purpose image compression schemes. Again, it must be emphasised that consideration of any compression ratio must take into account any assumptions or compromises made.

## 3.6 Compression Requirements for the Animation Server

This chapter concludes with consideration of the factors affecting the choice of compression algorithm for the Animation Server. The principal requirements for the Animation Server coding scheme are as follows:

- *Decoding* must be possible at frame rates.
- *Decoding* should be as simple as possible since the decoder has to be implemented in hardware.
- Any asymmetry between the complexity of the encoder and decoder should favour the latter. Coding is performed off-line by software, whereas decoding must be performed in real-time by hardware. However, coding cannot be arbitrarily complex because a reasonable coding rate is required to achieve acceptable turnaround times for interactive operation of the Server.
- The scheme should be image preserving. The Animation Server operates at the raster level and does not make *a priori* assumptions about the frame sequence to be processed. It therefore has no high level knowledge of the contents of the frame being coded. As a result it is impossible to design an approximating coding scheme which can guarantee to identify and preserve the most subjectively important image features in each frame. This might result in the loss of, for example, carefully constructed anti-aliasing information. In addition the problems of coding artifacts introduced into frame sequences by many approximating schemes have already been mentioned (Section 3.5).
- The scheme should exhibit graceful degradation in cases where the required degree of compression cannot be attained. One way to achieve this is by the progressive transmission of frames whereby coarse image detail appears first followed by the finer detail; then when approximation is required it is the latter which is lost first. Note that this requirement conflicts with the previous point because the only way to meet it is to introduce approximations to the required frames.

In designing a compression scheme which meets the above criteria, a number of approaches have to be rejected, and for a variety of reasons. For example, techniques such as transform coding or methods based upon motion compensation are unsuitable for this application because of their inherent complexity. The coding scheme developed for the Animation Server, which meets all the above requirements, is introduced in Chapter 4.

# Chapter 4

# An Overview of the Animation Server Architecture

## 4.1 Introduction

In this chapter the main architectural features of the Animation Server are introduced and an overview of the system's operation given. This discussion marks the start of the actual research performed by the author and many of the ideas introduced in here are amplified in the remaining chapters of the dissertation.

## 4.2 Rainbow II

The Animation Server is designed to integrate with an experimental display processor known as Rainbow II. Rainbow II was developed at the University of Cambridge Computer Laboratory and the choice of name reflects the influence of an earlier system known as the Rainbow Workstation.

### 4.2.1 The Rainbow Workstation

The Rainbow Workstation is an experimental graphics system designed at the University of Cambridge Computer Laboratory to investigate hardware support for screen windows [Wilk84,Styn85]. In the usual approach to window management on conventional graphics systems the layout of windows on the screen is a direct reflection of the layout of rasters within the frame buffer. Thus, to update the screen layout it is necessary rearrange the contents of the frame buffer. This is often a time consuming operation which reduces the performance of the system, even with special purpose support hardware for frame buffer memory management (such as a BitBlt operation, for example). This problem is compounded by the fact that windows are normally allowed to overlap each other: the movement of a window will often expose parts of some windows and obscure parts of others. Consequently it is necessary to record the inter-relationships between windows and take action to ensure that correct visibility is maintained at all times.

The Rainbow Workstation overcomes these problems through its use of *dynamic mapping* between images held in the frame buffer and their positions on the screen. There is no fixed relationship between a location in the frame buffer and a location on the screen. Instead, the workstation has intelligent video refresh logic which dynamically assembles images "on the fly" from different parts of the frame buffer as the image is being built up in scanline order on the display. To move a window all that is necessary is to instruct the video refresh logic to read the corresponding memory image at a different time in the refresh cycle; no rearrangement of frame buffer memory is necessary.

In later experiments this idea was extended to make use of look-up tables for real-time image blending operations. This provided support for a number of new effects including transparency, clipping against arbitrary shapes and dynamic anti-aliasing [Glau85*a*,Glau85*b*].

In many respects the Rainbow Workstation was highly successful as an experimental system. In particular, the ideas of hardware windowing and real-time image combination were found to be powerful and worthy of further investigation. It also had a number of limitations, in particular:

- The architecture used was found to be inflexible with the result that experimentation with different ideas or approaches is difficult or impossible. Much of the hardware is very specialised and not amenable to general purpose programming.

- There is an unequal distribution of processing power. For example, manipulating windows on the screen is rapid and easy whereas the initial loading of image data into the frame buffer is slow.

- The effects which can be achieved with image blending are limited, mainly because the use of the look-up table for this purpose was not envisaged at the time of the original design.

These observations formed the motivation for the design of a new workstation known as Rainbow II. This system is intended to overcome the limitations listed above whilst building on the strengths and successful elements of the original design.

### 4.2.2 Rainbow II

In the Rainbow II processing model an application is distributed across a pipeline of elements connected together by private high-speed unidirectional links. Each element handles a logical component of the computation. The name "Rainbow II" refers not to a specific and rigorously defined system but rather to a framework or platform that is designed to encourage experimentation with different hardware configurations. The architectural foundation of this framework is provided by a powerful graphics processing and display sub-system known as Garland, together with a communication system for rapid transmission of graphical information and

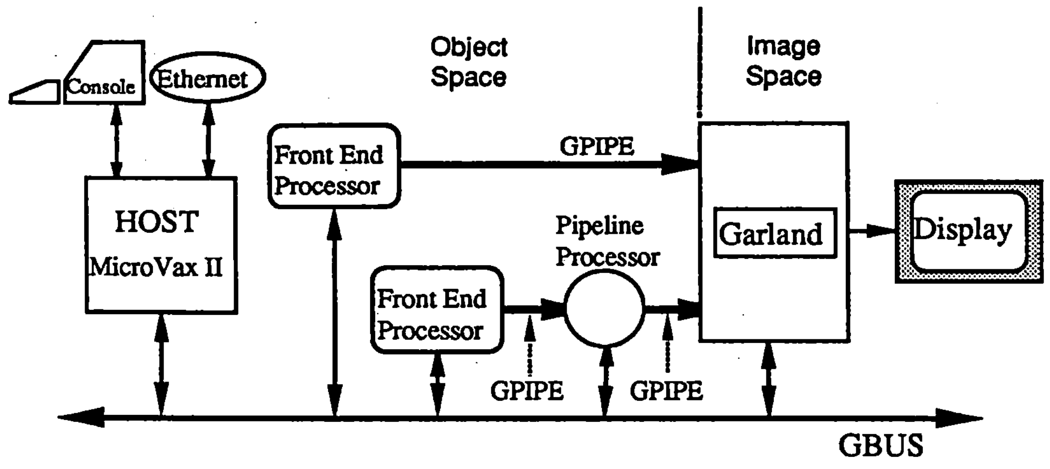


Figure 4.1: A typical Rainbow II system

for overall system control. Garland forms the sink stage of a typical pipeline and provides facilities for image buffering and support for hardware windowing, image processing and image combination. Other processors "upstream" in the pipeline act as either data sources or filter stages[1]. A three stage pipeline for object modelling, for example, might be composed of a 3-d object modeller as the source stage, a smooth shader as a filter stage and Garland as the sink stage. An outline of a typical Rainbow II arrangement is shown in Figure 4.1. Two pipelines are illustrated in this example. The processing elements of a pipeline are linked by fast asynchronous unidirectional links known as GPIPEs [King85*a*]. A GPIPE has a 32 bit wide data path and supports video-rate data up to a maximum (instantaneous) rate of 160MBytes/s. GBus [King85*b*] is a medium speed global communication bus which allows the individual pipeline elements to be initialised and controlled, and the overall operation of the system to be synchronised. There is exactly one such bus in a Rainbow II system. The top level control of the system is the task of the host processor (e.g., a Microvax II as shown in Figure 4.1). This manages the user interface to the system as well as providing support for system development, initialisation of processing elements (e.g., loading software), and so on.

The internal organisation of Garland is shown in Figure 4.2 and consists of the *Image Space* and *Video Processor*. The operation of the latter is straightforward: it generates a video signal output from data read off a selected GPIPE input. The image space is comprised of a homogeneous array of three *Image Planes* which are interconnected in the way shown by GPIPE links. The main data path on each Image Plane is centered around the *Blending Memory*, a conceptual view of which is shown in Figure 4.3. The Graphics Memory allows frame storage up to a maximum image size of 1024 × 1024 pixels, 8 bits/pixel. The look-up memory allows an arbitrary function to be performed between two 8 bit input values to

[1]A filter stage is an element which copies its input to its output processing each data item *en route*.

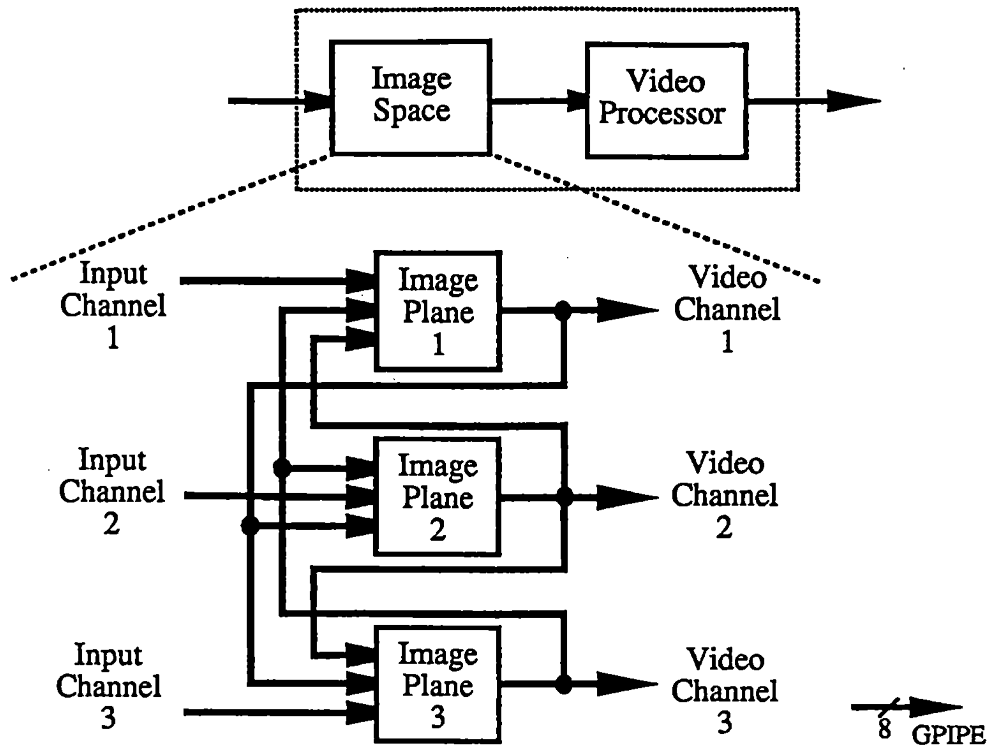


Figure 4.2: The Garland Image Space and Video Processor

yield an 8 bit result. This is achieved through the use of a 64K element look-up table. The close coupling of the blending function to the graphics memory and the generality of the data routing (as determined by the selectors $S_1$–$S_3$) were key aspects in the achievement of flexibility and power in the design. Each Image Plane has a programmable architecture and significant processing power in its own right. A hierarchy of three processors is used to control its operation; these are the Image Processor, the Task Control Unit (TCU) and the Task Execution Unit (TEU). The TCU and TEU are microcoded processors that together form the *Flow Processor* which is responsible for sequencing of the Image Plane's operation The Image Processor is used for more general control and processing functions. The interconnection of Image Planes of Figure 4.2 in conjunction with the Image Plane design itself affords considerable flexibility in the range of image manipulation functions which can be supported by the Garland architecture. With this arrangement up to three GPIPE links are available for connection of front-end processors.

The initial concept of Rainbow II and the design and development of Garland are due to T.R. King. The motivation for this work came directly from an involvement with the Rainbow Workstation and a desire to build on the experience gained. The principal aims of Rainbow II can be summarised as follows:

1. To produce a modular and extensible system which provides for future experimentation and expansion.

2. To produce a powerful and flexible *image manipulation* architecture. The

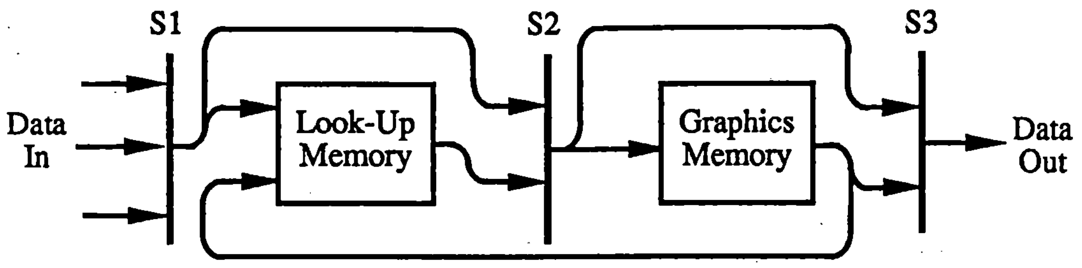


Figure 4.3: Conceptual overview of Garland image blending memory

aim was to produce a system which was equally suited to the (normally separate) fields of computer graphics and image processing.

3. To exploit coarse-grained parallelism for improving system performance. This is manifested in the adoption of a pipeline model of processing without centralised control.

Full details of Rainbow II and Garland are described in [King88].

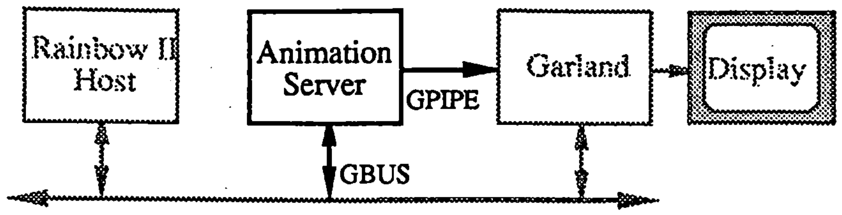
## 4.3 The Animation Server

The Animation Server is designed to fit into the above framework and takes the form of a front-end processor that appears to Garland as a frame-rate image source (Figure 4.4 (a)). Frame sequences are recovered from disc in compressed form, decoded in real-time, and the resulting data stream transmitted to Garland via a GPIPE link. Some of the Server's functionality is distributed to Garland in order to exploit the powerful image storage and manipulation functions offered. Garland is treated as a logical extension to the Server and the resulting integrated system avoids duplication of functionality.

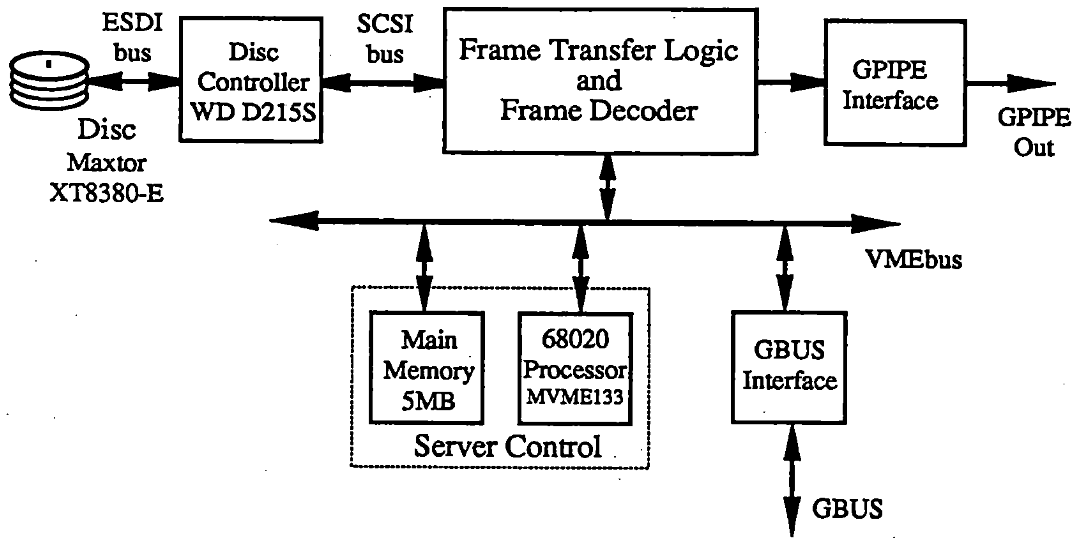
### 4.3.1 Server Architecture

An overview of the Animation Server architecture is presented in Figure 4.4 (b). The frame storage device is a Maxtor XT-8380E 5¼ inch Winchester disc drive which has an unformated capacity of 408MB and a transfer rate of 15Mb/s. Frame data are transferred via a SCSI bus[2]. The Frame Transfer Logic (FTL) and Frame Decoder together form the main Server logic that contains all the custom logic responsible for frame processing. Compressed frames read from the disc are buffered and decoded by this logic and the resulting video-rate data stream is transmitted via a GPIPE link to Garland for display. The disc sub-system, FTL/Frame Decoder and GPIPE interface together form the principal high speed path for frame

[2]The Small Computer Systems Interface (SCSI) is a widely used interface bus for the control of block structured peripheral devices (magnetic/optical discs, tape drives, etc). SCSI evolved from the SASI bus (a proprietary bus of Shugart Associates and *de facto* standard) and has been standardised by the X3T9.2 sub-committee of the American National Standards Institute [SCS84].



(a)



(b)

Figure 4.4: Overview of the Animation Server architecture

data transfer during replay. The control of this path is carried out by the Server Controller via a VMEbus acting as the system control bus. The Server Controller is implemented with a commercially available 32-bit 68020-based processor card. Software running on this processor is responsible for the higher level control and synchronisation of the Animation Server operation.

Further details of the FTL/Frame Decoder block (Figure 4.4) are shown in Figure 4.5. There is a clean functional separation of this logic into two main parts, separated by the FDEC_queue, with each half having a clearly defined role as follows:

1. *Frame Transfer Logic (FTL):* The FTL is responsible for the routing of frame data and control information between the disc, Server Controller and Frame Decoder, according to the selected configuration and transient operational status of the Server. A central element in the main frame data path is the Compressed Frame Buffer (CFB). This is a large circular buffer that is used for the the temporary storage of frames as they are read from disc (or, in the case of frame loading, before they are written to disc). Ordinarily several

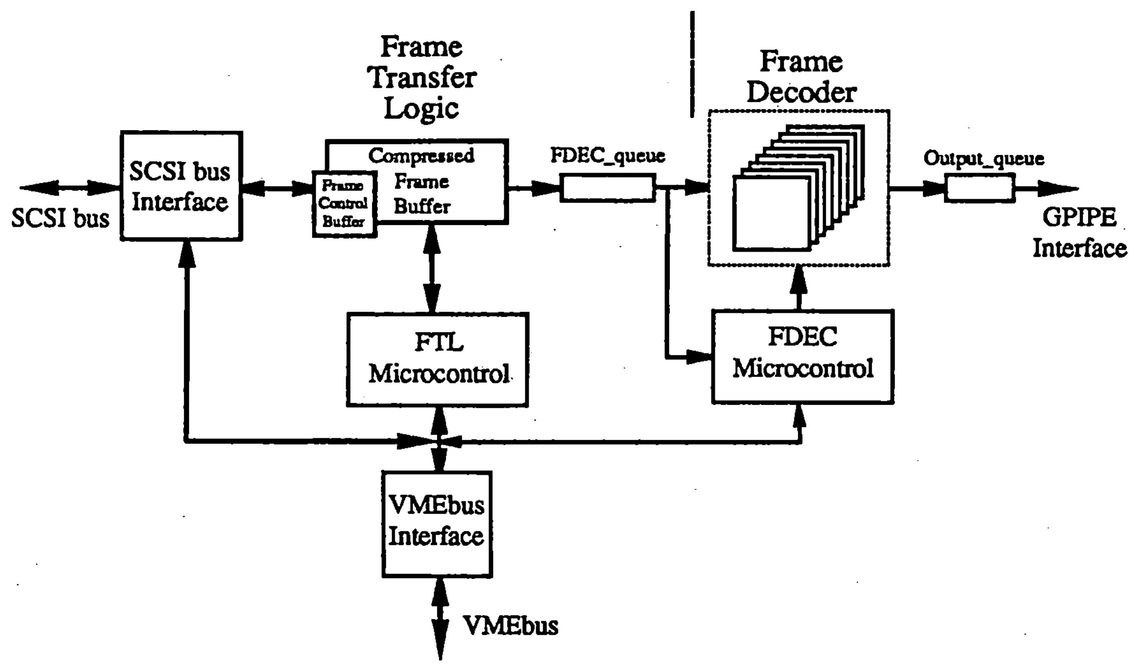


Figure 4.5: Main elements of the Frame Transfer Logic and Frame Decoder

compressed frames are buffered simultaneously. The CFB supports a number of functions related to frame flow control and error recovery. Its main role is during replay where it guarantees a head of compressed frame data to the Frame Decoder, irrespective of any loss of disc transfer caused by control, frame-sequencing or error recovery operations which the Server Controller may have instigated.

The Frame Control Buffer (FCB) is a smaller buffer that stores control information related to frames arriving from disc. This information is recorded interleaved with the frame data (See Section 4.7 for details of frame formats) and is automatically extracted by the FTL as the transfer progresses. Control information placed in the FCB is transferred to the Server Controller for further consideration and processing.

The operation of the FTL is controlled by a microprogrammed sequencer and access from the Server Controller is effected via a 16 register interface. The FTL is discussed in more detail in Chapter 5.

2. *The Frame Decoder:* Restoration of original frame data from the compressed form is carried out by the Frame Decoder. The Frame Decoder is implemented as 8 *Plane Decoder* units operating in parallel, each responsible for decoding one bit of each 8 bit pixel. A second microcontroller controls the operation of the decoder, driven both by command information embedded within the frame format (See Section 4.7) and by the Server Controller. The detailed operation of the Frame Decoder is discussed in Chapter 6.

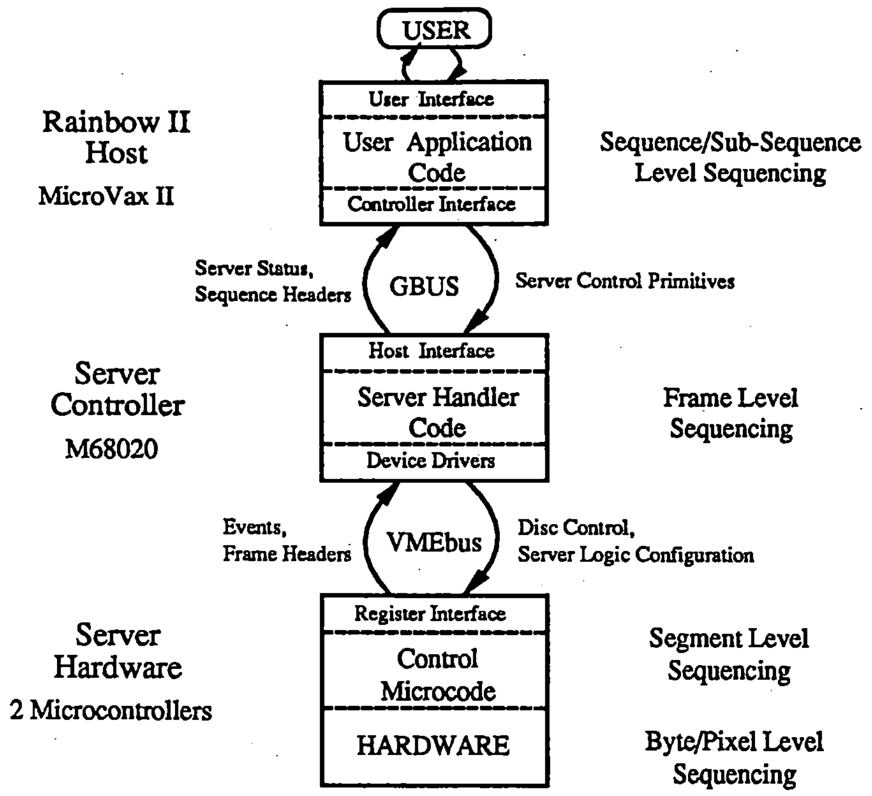


Figure 4.6: Control flow within the Animation Server (excluding Garland)

## 4.3.2 The Animation Server Control Hierarchy

Operational control of the Animation Server is achieved using a three level processor hierarchy consisting of: (1) Microcontrol within the FTL/Frame Decoder; (2) the Server Controller; and (3) the Rainbow II host. Each level of the hierarchy (Figure 4.6) has a clearly defined role as follows:

### Microcontrol

The two microcontrollers represent the lowest level of programmable control and are responsible for sequencing the main Server logic at the system clock rate. This includes the low level buffer and queue management, resolution of resource conflicts and all aspects of the Frame Decoder operation. The two microcontrollers together are designed to be sufficiently powerful to allow the operation of the main logic to be "free running" with respect to the Server Controller. That is, once initialised the FTL and Frame Decoder are capable of handling all aspects of frame transfer and processing without intervention from the Server Controller. Such intervention is only necessary when the Controller wishes to override the selected behaviour of the hardware or when recovery action is required after an error condition. Certain key *events* related to the status of the FTL/Frame Decoder and disc are reported by the microcontrollers to the Server Controller for the purposes of system control (see below). Control of the FTL/Frame Decoder and the disc interface from the

Server Controller is achieved via a register interface mapped into the VMEbus address space. This interface also provides access for initialisation, frame loading (i.e., recording) and diagnostic purposes.

### Server Controller

The 68020-based Controller is responsible for controlling and synchronising the operation of the disc drive, the main Server hardware and Garland. High-level requests sent by the Rainbow II host are interpreted by the Controller and translated into the corresponding lower-level sequence of primitive actions that are required to execute the requested task. The Controller executes code written in a mixture of 'C' and 68020 assembler.

The Controller operates at the frame level and is unaware of the internal representation used for frame data; from its point of view a frame is an atomic entity. All the information required for steering a frame through the Animation Server is contained within a frame header prepended to each frame and recorded on the disc along with the frame data. This header includes information which identifies the frame, gives its (compressed) size, its position on the disc, and so on. During replay as each new frame is fetched from disc the header is extracted by the FTL microcontroller and placed in the FCB. The arrival of a new frame causes the main logic to send a "frame_transfer_complete" event to the Controller and the Controller to subsequently retrieve the header from the FCB. The receipt of "frame_transfer_complete" and other key events, in conjunction with the frame header information, enables the Controller to maintain a model of the current status of the Animation Server. This model includes information such as the frame currently being transferred from disc, the frame currently being processed by the Frame Decoder and the number and locations of frames currently held in the CFB. In the most straightforward operation, that of continuous forward replay at a predetermined frame rate, the Controller adopts a largely supervisory role over the operation of the main frame data path; all the fine-grain control and sequencing is performed by the main logic microcontrollers. In other cases, such as interactive operation for example, the controller assumes a more active role. Here commands must be issued to the disc, main Server logic and Garland to ensure that the required primitive operations occur at the right time and in the correct sequence. Similarly, the Controller must intervene to restore the correct sequencing after an error condition has been detected.

The operation of the disc and Frame Decoder are mutually asynchronous, with the indirect coupling of the two facilitated by the CFB. With this arrangement the disc spindle speed does not have to be synchronised to the frame or field rate, unlike the BBC Television Animation Store, allowing the use of an unmodified drive. Data transfer from the disc proceeds whenever there is space in the CFB, and transfer of each new frame starts immediately on completion of the transfer of the last, independent of the operational status of the Frame Decoder and without reference to frame periods. Frame transfer from disc leads frame decoding by several frames. Frames are mapped to the disc without reference to the disc's physical format; there is no requirement for a compressed frame to occupy a given number of disc blocks (such as one frame per track, for example) or even an integral number of blocks. In fact, such a requirement would be impossible to meet since the size of compressed frames is variable—unless of course a fixed compressed frame size was set and all smaller frames wastefully padded to meet this size. However, frames are generally placed contiguously on the disc because the greatest system performance is obtained for this case. Each discontinuity in a frame sequence requires a further disc command to be issued followed by a subsequent seek and/or rotational latency before the flow can resume. This degrades the system's overall performance. The Animation Server design does support the use of discontinuous frame sequences (this is one of the functions of the CFB—See Chapter 5) and this is used to increase the flexibility of the system. However, such a mechanism has inherent limitations due to the nature of the disc operation.

In the development version of the Animation Server the GBus interface (Figure 4.4) is replaced by a commercial interface card which allows access to a Cambridge Fast Ring (CFR) and distributed system [Hopp88]. This allows loading of software from central fileservers and is a convenient means for system development independent of the Rainbow II environment and in particular in the absence of GBus access to the Rainbow II host processor.

### Rainbow II Host

The Rainbow II host runs user level application code that interacts with the Server Controller via GBus. Commands issued by the host to the controller take the form of high-level control primitives which initiate actions within the Animation Server; the host is unconcerned with the frame to frame operation of the Server being handled by lower levels in the control hierarchy. For example, the Rainbow II host identifies frames only in terms of a frame number and knows nothing of frame sizes or of their locations within the disc or CFB. This control organisation fits in well with the GBus process model [King88] in which simple commands initiate a sequence of more complex actions. The set of primitives allows host control of frame replay, loading etc., some examples are given in Table 4.1. This set is easily extended by providing the appropriate code in the Server Controller. A handler task interprets each command and splits it up into a corresponding sequence of more primitive operations. Commands are examined upon receipt and normally placed on an work queue to await completion of preceding commands. Queuing allows a sequence of control primitives to be transmitted in succession, effectively forming a composite higher level command. In this way more complex control over the Server can be implemented.

User-level programmed control of the Server is achieved via a library interface providing access to the control primitives. More commonly, however, a higher level graphical interface would be used. Interactive operation of the Server is supported by the use of a simulated control panel that provides "buttons" for all the required functions (PLAY, PAUSE, REWIND, etc) together with feedback as to the current frame number, position in sequence, and so on. Such an interface is easily

| Command | Meaning |
|---|---|
| READ_SEQUENCE_HEADER *name* | Returns host-specific portion of named sequence header |
| MOVE *n* | Move to start of frame *n* |
| PLAY *n* | Replay next *n* frames (*n*=1 gives single stepping) |
| PAUSE *t* | Hold current frame for *t* frame times |
| STOP | Halt replay at end of current frame |
| SET_REPLAY_SPEED *f* | Set replay speed (frame rate) to *f* fps |
| READ_CURRENT_FRAME_NUMBER | Returns number of frame currently being displayed |
| READ_SERVER_STATUS | Returns current Server status (Server initialised, active etc) |

Table 4.1: Example host control primitives

implemented under current windowing systems (e.g. X11.3 which is supported on the Rainbow II host Microvax).

## 4.4 Image Coding Strategy

### 4.4.1 Introduction

Some general image coding requirements for the Animation Server were outlined in Section 3.6. A number of constraints are imposed upon the operation of any coding scheme by the demands of real-time playback. In particular, three parameters have a major bearing upon the overall achievable system performance: the disc transfer rate (i.e., the rate at which compressed frame data can be recovered from disc), the degree of reversible (lossless) frame compression obtainable and the total number of Frame Decoder processing cycles available per frame time. A central aim of the Animation Server is to optimise the use of these limited resources over a wide range of operating conditions and requirements. This is achieved by avoiding the imposition of rigid limits on parameters such as frame size and replay rate. Flexibility in the architecture and coding scheme allow the user to trade off such parameters against one another according to the needs of the particular application.

Within the Animation Server a number of alternative image coding techniques or strategies are implemented. These are supported within a general framework, the basis of which is provided by a *frame segmentation mechanism* (Section 4.6). The core of the coding scheme is provided by an implementation of a new predictive/runlength compression algorithm. The details of this scheme are described first and then the discussion is expanded to cover the general coding strategy.

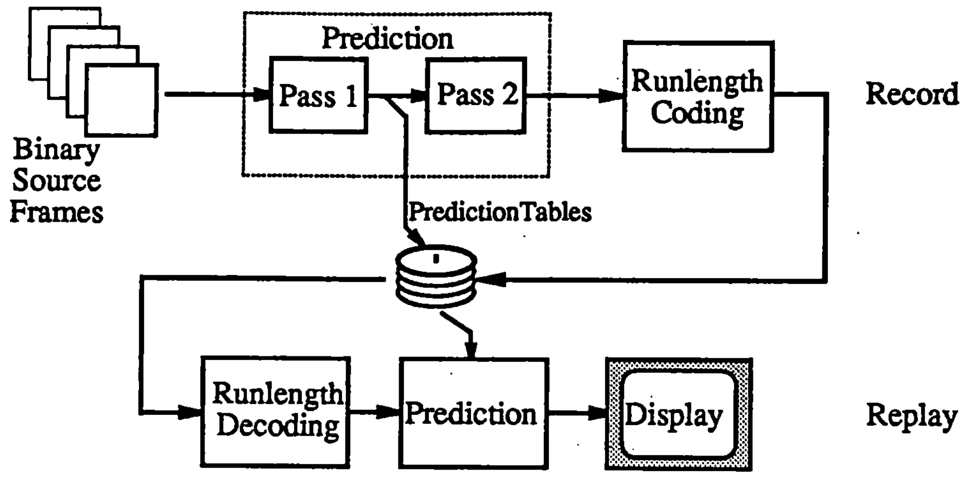


Figure 4.7: Overview of the Animation Server Coding process for a single binary plane

## 4.4.2 Predictive/Runlength Coding

During coding the encoder traverses each frame in scanline order attempting to predict the value of each upcoming pixel from its knowledge of previous pixel values. Each prediction is compared with the corresponding actual pixel to generate an error term and the resulting string of error terms is coded with an optimised runlength coder. This runlength stream represents the compressed form of the frame. The decoder uses an identical prediction function to the coder and consequently makes exactly the same good and bad predictions. Of course the decoder does not have access to the original frame data so corrects bad predictions using the runlength data transmitted from the coder. This scheme is lossless (frames are reconstructed exactly by the decoder) and essentially symmetrical, although the encoder is slightly more complicated because it also has to determine the prediction function. A simplified view of the encoding and decoding processes is illustrated in Figure 4.7. The prediction function employed actually operates on binary data; an 8-bit deep image is processed as 8 separate binary *planes*. During the encoding stage planes are processed sequentially. For decoding, however, speed is critical and planes are decoded in parallel. The coding and decoding stages are described more fully below.

## 4.4.3 Coding of Binary Frames

The coding of a single binary plane will be described, with the extension to multiple plane frame data discussed later.

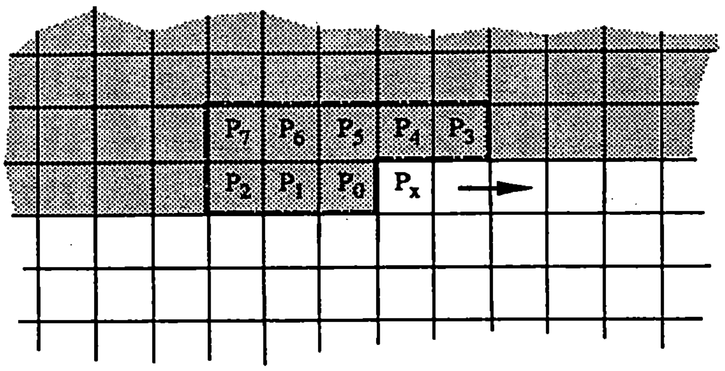


Figure 4.8: Typical prediction template

## Prediction

The prediction scheme developed is based upon a suggestion made by Professor David Wheeler of the University of Cambridge Computer Laboratory[3]. This employs a 2-d prediction template formed from the neighbouring bits of the unknown next pixel $P_x$ (see Figure 4.8). The aim is to determine the most likely value of $P_x$ given the values of the prediction template bits. Note that no template bits ahead of the current position can be used—during decoding the values of such bits would be undefined. Prediction is a two stage process, requiring two passes over the plane:

**First Pass:** The first pass is essentially a pre-processing stage whose purpose is to gather frame statistics for later use. The output of this pass is a *Prediction Table* that is used by both the encoder and decoder. The frame is scanned in scanline order and at each pixel position the bits of the prediction template together with the "unknown" pixel $P_x$ are formed into an $(m+1)$-bit index and used to access a $2^{m+1}$ entry *Occurrence Table* (Figure 4.9). The indexed entry is incremented to record the relationship between that template bit pattern and the value of $P_x$ (0/1).

At the end of the pass each odd/even pair of Occurrence Table entries T0,T1 record the number of times $P_x$ was found to be 0,1 respectively for the template pattern T and the sum of these entries gives the total number of times the pattern T occurred in the plane. Let the values of each pair of entries be denoted by $n_{t_0}$ and $n_{t_1}$ for any m-bit template pattern $t$, $0 \leq t < 2^m$.

The Occurrence Table is used to construct a $2^m$ entry *Prediction Table* which during the second plane pass will be indexed by the $m$-bit prediction template. Each Prediction Table entry is derived from the corresponding pair of Occurrence Table entries (Figure 4.10) and consists of two fields as follows:

1. A predicted pixel (0 or 1) for that prediction template pattern, and

[3]A similar scheme has been proposed by Wholey in an early (1961) paper [Whol61] (a reference found after implementation of the present scheme).

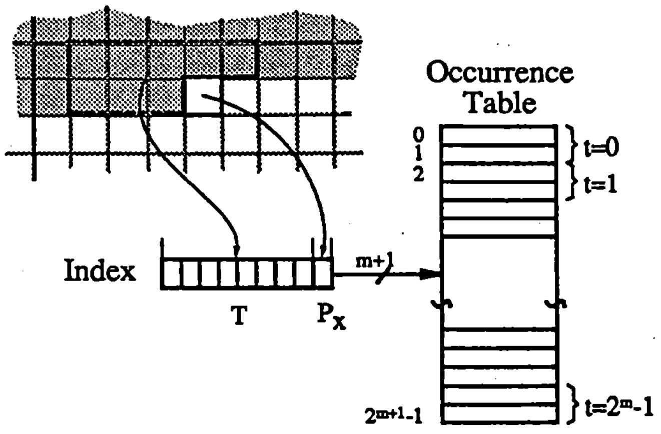


Figure 4.9: Occurrence Table generation

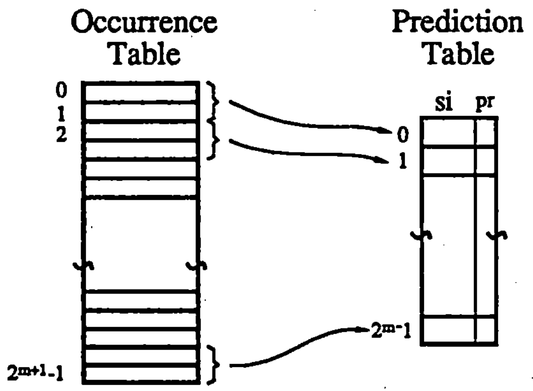


Figure 4.10: Prediction Table production from an Occurrence Table

2. A "quality of prediction" indicator which is used to guide the runlength coding process.

Consider a Prediction Table entry $PT_t$, $0 \leq t < 2^m$. Then, the predicted bit value for the template pattern $t$ is simply that which was found to occur the most frequently in the plane. That is:

$$PT_t.pr = \begin{cases} 0 & \text{if } n_{t_0} > n_{t_1} \\ 1 & \text{otherwise} \end{cases}$$

Let $n_{t_p}$ be the number of occurrences corresponding to the predicted value (that is, $n_{t_p} = \text{MAX}(n_{t_0}, n_{t_1})$). Then of the $(n_{t_0} + n_{t_1})$ predictions which will be made using this prediction table entry, $n_{t_p}$ will be correct. The probability of a successful prediction is therefore given by:

$$\text{Prob.(success)} = \frac{n_{t_p}}{n_{t_0} + n_{t_1}},$$

which ranges between 1/2 and 1. A probability of 1/2 means that the predictor is essentially guessing—'0' and '1' are equally likely ($n_{t_0} = n_{t_1}$). At the other extreme a probability of of 1 means every prediction made will be successful (that is, if $n_{t_p} = n_{t_0}$ then $n_{t_1} = 0$ or *vice versa*). This probability information is used to guide both the runlength encoder and decoder. For this purpose a ***stream index*** is calculated and stored in the Prediction Table. The value of the stream index is given by:

$$PT_t.si = \log_{\frac{1}{2}}\left(\log_{\frac{1}{2}}\left(\frac{n_{t_p}}{n_{t_0} + n_{t_1}}\right)\right)$$

This equation is derived from $p = \frac{1}{2}^{\frac{1}{2}^{s}}$ [Whee86] which relates prediction probability to expected average runlength.

**Second Pass:** During the second pass actual predictions are made using the predictor template to index the Prediction Table. A prediction and stream index are obtained for each pixel in the plane. Each prediction is compared with the corresponding pixel and '0' is output to indicate a successful prediction and '1' to indicate a failure[4]. The result of this pass is thus an error bitstream, the same size as the original frame, together with associated stream indices.

### Runlength Coding

Runlength coding is only efficient if the size of the runlength counter is accurately matched to the average length of run. Failure to do this results either in the use of multiple counters for a single run (when the counter is smaller than the average runlength) or wasted counter bits (when the counter is larger). Efficient coding is achieved in the Animation Server's scheme by employing two optimisations devised by David Wheeler [Whee86]. These exploit knowledge of the operation of the prediction process. Firstly, runs of '0's are coded differently to runs of '1's, with the scheme favouring the former. The assumption is made that a '0', identifying a correct prediction, is more likely than a '1', a wrong prediction—a fair assumption for a reasonable predictor and one verified by experimental observations. Secondly, the incoming source data are split into a number of ***streams*** that are coded separately. Each stream has a counter whose size is determined by the stream index: stream 0 has a 0-bit counter, stream 1 a 1-bit counter, ...and so on up to stream N-1. In practice efficient coding can be obtained with a relatively small number of streams (e.g., N=8 is sufficient). Each incoming bit is encoded in the stream identified by its associated stream index. Recall that the stream index is derived from the probability that the predicted bit will be correct. This probability is directly related to the average expected runlength of successful predictions ('0's)—good predictions are more often correct and hence form longer runs than bad predictions. With multiple streams good predictions can be coded

[4]Note that with binary plane data a single bit ('1') is sufficient to describe a failed prediction; the correct pixel value is obtained simply by inverting the prediction. In the more general case of an n-bit deep frame, it would be necessary to also transmit the actual pixel value (or the difference between the predicted and actual value).

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Source: | 0000 | 001 | 01 | 001 | 0001 | 01 | 0000 | 0000 | 1 |
| Coded: | 0 | 110 | 101 | 110 | 111 | 101 | 0 | 0 | 100 |

Figure 4.11: Example runlength coding for stream 2

separately from bad, and the counter in each stream matched to the expected average runlength; stream 0 codes the worst predictions (probability≈1/2), stream N-1 codes the best (probability≈1). The encoder maintains N separate counters and incoming bits are coded using the following scheme:

```
IF (bit = 0)
{
    $counter_s$ := $counter_s$ + 1;
    IF ($counter_s$ = $2^s$)
    {
        OUTPUT ('0');
        $counter_s$ := 0;
    }
}
ELSE /* bit = 1 */
{
    OUTPUT '1'<$counter_s$>;
    $counter_s$ := 0;
}
```

where $s$ is the stream index associated with the input bit. Thus, in the coded stream a '0' represents $2^s$ '0's, '1'$< n >$ represents $n$ '0's ($0 \leq n \leq 2^s - 1$, where $n$ is an $s$-bit counter) followed by a '1'. An example coding of a stream is illustrated in Figure 4.11 and an example of multiple stream coding for 4 streams is shown in Figure 4.12. Note that the counter for stream 0 requires 0 bits (that is, '0' and '1' stand for themselves). A consequence of this is that the encoded form cannot be longer than the original input data, a property which is far from true for ordinary runlength coding.

On completion of coding, the N streams are merged to form a single output stream for the plane, the merge ordering being determined by the sequence in which streams are referenced (i.e., as defined by the stream-index stream generated by the predictor). When a stream is referenced a completed run is obtained from the appropriate stream coder and placed in the output stream. The length of this run, $n$ say, is noted and the following $n - 1$ references to this stream are ignored by the merger. Then on the next reference a further new run is written to the output stream. In this way the runlength ordering in the composite stream is guaranteed to be consistent with the order in which new runlength data will be demanded by a Runlength Decoder Unit. This process of *stream merging*

```
Uncoded Source:
        32221002010302012213212000023320102320032102302102300222
        00000010000010010010100010110000000011000010010100110101

Code Streams:
        0:   0100100101001011010     →  0100100101001011010
        1:   00110001                →  0 11 10 0 11
        2:   00000010100100000101    →  0 110 101 110 0 100 101
        3:   000000001               →  0 1000

Merge Streams:
(3 2 1 0 0 0 0 0 2   0 1  1  1 2   0 0 0 0 2   0 0 0 0 2 1  0 0 0 3    0 0 2   2 )
 0 0 0 0 1 0 0 1 110 0 11 10 0 101 0 1 0 1 110 0 0 1 0 0 11 1 1 0 1000 1 0 100 101
```

Figure 4.12: Example multiple stream runlength coding

is illustrated in Figure 4.12. The implementation of merging and stream coding as separate stages requires the use of intermediate storage for completed stream runlengths. However, attempting a pipelined parallel implementation of this coding presents major problems: the merger demands completed runs from stream coders (in order to insert them into the composite stream at the correct position) which the stream decoder may be unable to supply as it may not yet have seen sufficient input bits to allow it to determine the length of the current run. Recall that because of the interleaved nature of the stream input to the runlength coder (Figure 4.12), predictor references to a particular stream (and consequently the constituent bits of a particular stream run) may be spread over several hundred or thousand source bits. Similar problems do not arise during decoding and so unmerging and stream decoding can (and do) occur in parallel on a demand basis.

## 4.4.4 Coding of Multiple-plane Frames

Each plane in a $n$-bit deep frame is coded separately using the above scheme. The resulting plane streams are then merged to form a single composite stream for the frame. Incoming bit-streams are split into byte sections and merging is performed at a byte granularity. The Runlength Decoder Units within the Frame Decoder (see Chapter 6) have byte wide input paths and stream data are distributed on a byte-by-byte basis. The order in which the Runlength Decoder Units request new runlength data, and hence the required merge order, cannot be calculated directly; it is a complicated function of plane and frame statistics. The plane statistics determine the distribution of data amongst the runlength streams. The frame statistics determine the distribution of data between the planes; in general this distribution is uneven, with higher order planes (which tend to code more efficiently) demanding less data than lower order planes. The merging is accomplished by effectively simulating the Frame Decoder's operation. More precisely, enough of this operation is considered to be able to determine for each Decoder cycle which Runlength Decoder Units have pending requests for new runlength data. The operation of a Runlength Decoder Unit is discussed more fully in Chapter 6.

The compressed frame data representation therefore consists of a two level interleaving of $N \times P$ separate runlength streams, where $N$ is the number of streams/plane and $P$ is the number of planes. In the Animation Server $N = P = 8$. Each compressed frame is represented by a doubly-interleaved runlength stream together with $P$ prediction tables defining the prediction functions to be used for each plane in that frame.

### 4.4.5 Frame Decoding

Each frame is decoded as separate binary planes using 8 *Plane Decoders* operating in parallel. A Plane Decoder consists of a Runlength Decoder Unit coupled to a Predictor Unit. The incoming data stream is split into bytes which are distributed to the Runlength Decoder Units on a demand basis (recall that the correct byte ordering is determined during coding). The decoding of an individual plane is similar to its coding:

#### Prediction

The Prediction Table within the Predictor Unit is loaded from the transmitted table data to establish the prediction function for the plane. The frame is scanned as for the coder's second pass and at each pixel position a prediction and stream index are obtained via a Prediction Table look-up. The stream index is sent to the runlength decoder which decodes the next bit from the specified stream and returns it to the Predictor Unit. The returned bit indicates a successful or failed prediction ('0' or '1', respectively) and is used to invert the predicted value if wrong.

#### Runlength Decoding

The decoder maintains N counters, one for each stream. On receipt of a stream index $s$ the appropriate counter is examined and '0' or '1' returned to the predictor as appropriate. When a count expires a new run is started for the stream and either 1 or $s+1$ bits (for '0' or '1'$< n >$, respectively) are read from the decoder unit's input. The first bit read in the new coded runlength is viewed as a flag that distinguishes maximal (i.e., $2^s$ '0's) and non-maximal runs ($n$ '0's followed by a '1') and this is stored with each counter. The decoder algorithm is as follows:

```
IF ($counter_s$=0 & $flag_s$=0)
    /* run has expired */
    LOAD ($flag_s$, $counter_s$);
```

```
IF (counter_s > 0)
{
    counter_s := counter_s - 1;
    RETURN '0';
}
ELSE /* counter_s=0, flag_s=1 */
{
    /* non-maximal run has expired */
    flag_s := 0;
    RETURN '1';
}
```

where LOAD takes the form:

```
LOAD(flag_s, counter_s)
{
    flag_s := <next bit in input stream>;
    IF (flag_s = 0)
        counter_s := $2^s$;
    ELSE
        counter_s := <next $s$ bits from input stream>;
}
```

## 4.5 Frame Reconstruction Modes

As mentioned earlier the operation of the Frame Decoder is actually more general than outlined above, with the runlength/predictive coding providing the central core of the overall frame coding strategy. The preceding description of the decoding algorithm represents the most straightforward operation of the Frame Decoder. Extensions to this operation are obtained as a result of three factors. Firstly, values decoded by the Frame Decoder need not necessarily be interpreted directly as pixels. Secondly, the operation of the Frame Decoder is programmable via its microcontroller, allowing control over the sequencing of its internal pipelined stages. Thirdly, the operation of Garland is programmable, allowing control over its image storage and manipulation functions. Together these allow the implementation of a number of frame reconstruction *modes* that greatly enhance the utility and performance of the Animation Server's compression scheme. A mode defines the way in which frame data are reconstructed from the compressed representation. The principal modes are as follows:

1. **Full Pixel:** Decoded values are interpreted directly as 8-bit pixel values, with one pixel decoded per Frame Decoder cycle. This corresponds to the operation of the Frame Decoder described above.

2. **Temporal Difference:** Each decoded value is interpreted as the *difference* ($d$) between the required pixel ($P_{x,y,t}$) and the value of the corresponding pixel

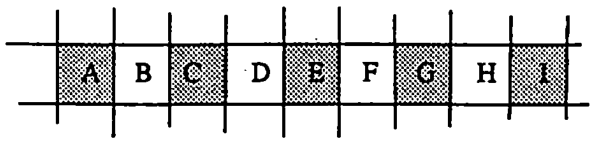


Figure 4.13: The pixel interpolation mode

in the previous frame ($P_{x,y,t-1}$). The required pixel is obtained by adding the difference to the previous-frame pixel: $P_{x,y,t} = P_{x,y,t-1} + d$. This is the principal operating mode of the Animation Server. During coding a set of difference frames is generated by taking the difference between each consecutive pair of source frames. After differencing the source frames are discarded and the set of difference frames coded instead. Differencing exploits temporal coherence within frame sequences. Coding the changes between frames is generally more efficient than coding the frames themselves.

3. **Spatial Difference:** Each decoded value is interpreted as the difference ($d$) between the required pixel ($P_{x,y,t}$) and the previous pixel ($P_{x-1,y,t}$), $P_{x,y,t} = P_{x-1,y,t} + d$. This exploits 1-d spatial coherence within a frame (that is, the property of many images that as a scanline is traversed the change in intensity is both gradual and continuous).

4. **Interpolation:** During compression only half the source frame pixels are examined and coded, and the missing samples are reproduced at the decoder by interpolation of transmitted values. This is a form of interpolative coding (see Chapter 3 and [Hask72]). Figure 4.13 illustrates the operation of this mode along a scanline. The interpolation function used is a simple average of the two horizontal neighbours[5]. Decoded frames approximate the original source data, but for source frames with a moderate degree of spatial coherence the reconstructed frame closely matches the original. The mode approximately halves the compressed frame storage requirement (compared to that required for the same frame coded using full pixel mode), with a corresponding saving in the disc bandwidth required for its transfer. Interpolation thus provides a mechanism for achieving the required degree of compression when this cannot be obtained with one of the lossless modes.

   The Animation Server also supports a *pixel repeating* mode (where B←A, D←C, F←E, etc in Figure 4.13). However, this is less useful than interpolation as it renders a poorer approximation to the source frame.

5. **Plane Masking:** Individual Plane Decoder units within the Frame Decoder can be disabled and the corresponding pixel bit is effectively masked ('0' is output from the unit). More importantly, a disabled unit does not request any compressed input data. Consequently, this can also be used to effect a reduction in bandwidth requirements. In particular, the technique of reducing *amplitude resolution* can be employed (Chapter 3) whereby the number

[5]In fact two different functions are provided, one for greyscale and one for colour pixels.

of bits assigned to each pixel is reduced. This again results in an approximation to the required frame. Very considerable savings can be achieved by masking the low order planes where much of the compression effort is often concentrated. Recall that for natural greyscale images, for example, a reduction of 8 to 6 bits is possible with little loss of subjective image quality.

6. **Variable Rate Decoding**: Failure to meet the required degree of compression results in the Frame Decoder demanding new runlength data at a greater rate than the disc can supply it. In the short term larger compressed frames are handled by buffering within the Frame Transfer Logic (indeed, this is one of the functions of the CFB—see Section 5.3). For a longer term solution the decoding rate of the Frame Decoder can be reduced whilst maintaining full speed operation of the disc and FTL. This allows the frame decoding rate to be matched to the rate at which data are demanded from disc (which in turn is a function of the degree of compression obtained). Note that the decoding rate is distinct from the frame rate and that these two rates can be varied independently of each other (see Section 4.6.1). Reduced Decoder rate operation is achieved by the introduction of idle cycles, the ratio of idle to active cycles determining the rate of reduction. With this mechanism the decoding rate is specified as a fraction of the full decoding speed and selected from a continuous spectrum of possibilities—1/2, 3/4, 2/3, 1/3, 3/5 ... of the full rate, for example.

### Mixing Modes

The above modes are not mutually exclusive and can be combined to create new modes. The principal purpose of combining modes is to increase further the degree of frame compression obtained. So, for example, *approximate* temporal differences can be generated by the combination of plane masking and temporal difference modes. Very considerable reductions in compression requirements are possible by such combinations. For example, the compression obtained by approximate temporal differences could be improved further by combination with interpolation or reduced decoder rate, or both. The implementation of a new mode may or may not involve the provision of new Frame Decoder microcode, depending upon the particular combination in question (see Chapter 6).

## 4.6 Frame Segmentation

Central to the operation of the Animation Server is the *frame segmentation* mechanism which addresses two main issues:

1. It provides flexibility in the operation of the Animation Server and freedom of control over its finite resources by allowing the user to trade off parameters such as frame size and replay rate against one another according to the needs of the particular application.

2. It provides hardware assistance for frame generation in cases where the required degree of compression cannot be achieved.

The basis of this mechanism is the introduction of a sub-frame element known as a *segment*. From the point of view of the disc drive, the FTL/CFB and Server Controller a frame is an indivisible entity. Within the Frame Decoder and Garland, however, the basic processing unit is the segment and a frame is viewed as being composed of a set of (normally non-overlapping) segments. A segment is a rectangular image raster defined by an origin, width and height and some other control information. Segmentation is specified on a per-frame basis and can be changed arbitrarily between frames. No upper limit is placed upon the number of segments per frame. However, this is normally kept relatively small (e.g., under 16) as there is an overhead associated with each segment switch; segmentation is intended to be a coarse-grained mechanism.

The segmentation mechanism provides a framework in which the following are supported:

1. *Frame Reconstruction Modes*: All the modes described in Section 4.5 are associated with segments rather than entire frames. This allows different modes to be freely mixed within individual frames and, therefore, different parts of a frame to be reconstructed in different ways. For example, one part may be generated with full pixels whilst others are differenced, some parts may be generated exactly and other areas approximated, some areas may be decoded at the full frame rate whilst others are decoded at a lower rate, and so on.

2. *Prediction Table Loading*: The loading of prediction tables, which define the prediction function, is also associated with segments rather than entire frames. At each segment boundary the prediction table of each of the Predictor Units may be optionally (re-)loaded (that is, for each segment between 0 and 8 prediction tables are loaded). This dynamic loading facilitates a simple form of adaptive prediction—when frame statistics vary widely over a frame prediction functions can be localised to remove disruptive influences from other areas of the plane.

3. *Conditional Frame Replenishment*: There is no requirement for the set of segments defining a frame to cover the whole frame area. Pixels in uncovered areas retain their values from the previous frame. With this arrangement a technique known as *conditional frame replenishment* [Hask72] can be supported whereby only the areas of the frame which have changed since the previous frame are coded and updated. Conditional frame replenishment represents another means in the Animation Server by which the obtainable degree of frame compression can be increased.

   The level of compression can be increased further by a generalisation of this technique in which changes are only transmitted when they exceed a preset threshold. This is a form of approximate coding, where the degree of approximation is determined by the threshold. This approach has the desirable property that as the threshold is increased it is the fine detail (small changes) which is lost first whilst coarse detail (large changes) is maintained.

4. *Variable Frame Size*: The size (and shape) of frames in an animated sequence is a direct function of the segmentation employed. The smallest frame size supported is that of a single, minimum-sized segment—$4 \times 4$ pixels. An upper frame size limit of $1024 \times 1024$ is imposed by Garland. All Frame Decoder addressing is 10-bit so the maximum segment size is also $1024 \times 1024$[6]. A large frame cannot be fully updated by the Frame Decoder within a single frame time. The notional Animation Server frame size is $512^2$ pixels. More precisely, the Frame Decoder's master clock has a period of $1/(512^2 \times 25)$s ($\approx$152.59ns) and one pixel can be decoded per Decoder cycle. Thus, during each frame time a maximum of 262144 (i.e., $512^2$) pixels can be decoded. Larger frames are handled either by using conditional frame replenishment (when the full frame area is not active in a frame time) or by reducing the frame rate.

5. *Frame Error Recovery* and *Frame Sequencing*: These relate to the adoption of temporal differencing as the principal frame reconstruction mode. With differencing if any pixels are not properly updated, because of a transmission error for example, then that error propagates until the end of the sequence. Similarly, changing the order in which frames are replayed is impossible as each frame is derived from a predecessor that is determined at the time of coding. Re-sequencing is required for error recovery, for interactive operation (e.g. to cycle back interactively through a particular sub-sequence) and to allow replay to start from an arbitrary point in the sequence. These problems can be overcome by the inclusion of full-pixel segment(s) in each frame, as discussed in greater detail in Chapter 7.

An example of frame segmentation is illustrated in Figure 4.14. In a typical coding strategy, temporal differences (together with full pixel segments—see point 5 above) form the default operation mode. Approximation is only introduced when the required degree of compression cannot be achieved at the desired frame rate. The degree of approximation is increased until the required level of compression has been achieved. The use of a segmentation mechanism means that approximation can often be introduced in such a way as to minimise the level of subjective distortion in the sequence. This is done by exploiting known properties of the HVS in the way discussed in Chapter 3. Some of these approaches are included amongst the following examples.

[6]Thus, the Animation Server can define *frames* up to a maximum of $2048 \times 2048$ using a tiling of 4 maximal segments.

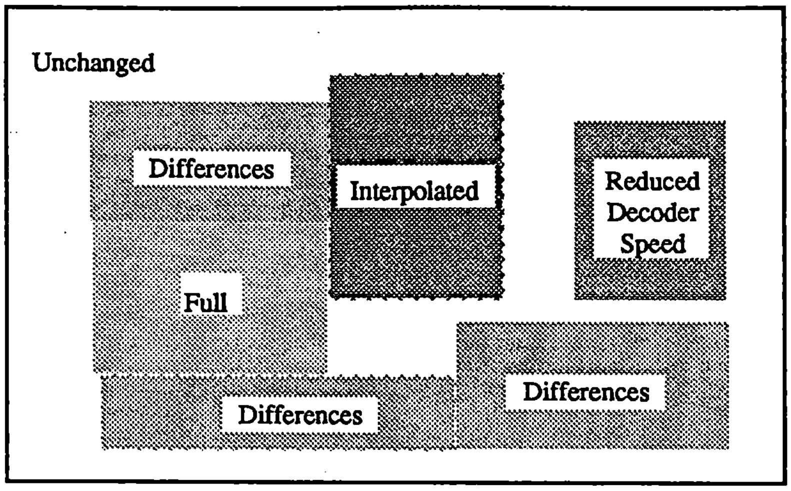


Figure 4.14: An example of frame segmentation

### 4.6.1 Examples of Segmentation Usage

In the remainder of this section a few representative examples are given to illustrate the flexibility of the segmentation mechanism and to show how a number of existing coding schemes or approaches may be implemented within this framework.

**Adaptive Plane Coding** Generally within a frame the higher order planes compress more efficiently than the lower order planes because the latter contain the fine detail and/or noise of the image. To compensate for this more effort can be expended in the coding of the low order planes through the use of dynamic prediction table reloading during the course of the frame-time. Multiple prediction functions allow the localisation of coding effort within these planes. For the high order planes a single prediction function is used for the entire frame by loading the prediction table once at the start of the first segment.

**Variable Frame Size** As already discussed, the frame size of the Animation Server is not fixed and frame sequences other than those of the notional size ($512^2$) can be handled. This includes smaller frames occupying only a part of the screen display area—frames are replayed into a window in Garland and this window treated just like any other (containing text, still graphics, etc). Thus the animation can be moved around on screen, obscured by other windows, and so on. One advantage of this is that as the frame size of a sequence is reduced progressively less compression is required to support its replay (until at a frame size of approximately $256^2$ (or equivalent) *no* compression is required). The bandwidth saved by the use of smaller windows could be used to enable the simultaneous replay of two (or more) sequences into multiple windows. Incidentally, note that the frame size in a sequence can be varied arbitrarily between frames if so desired.

**Exchanging Spatial and Temporal Resolution** A number of compression schemes have been investigated which exploit the observer's varying tolerance to frame distortion according to the rate of image change [Hask72]. In particular it is known that the HVS is less able to discern fine detail in areas of rapid and unpredictable movement than it is in stationary or slowly changing areas. Many of the techniques proposed can be implemented via the Animation Server's segmentation mechanism. For example, one approach is that of exchanging spatial and temporal resolution whereby sub-sampling is used to reduce the spatial resolution in moving areas and the temporal resolution in stationary areas. With the segmentation mechanism moving areas can be reproduced by interpolation (which is a form of spatial sub-sampling) whilst stationary areas are reproduced using conditional frame replenishment (a form of temporal sub-sampling).

**Progressive Image Transmission** The ability to control the degree of approximation in a frame allows for progressive transmission of images. For example, differencing is least effective (and hence compression hardest) at the points in a frame sequence where there is an abrupt change in frame content, as at a scene cut for example. At such points approximation is used to first render a coarse representation of the required frame, followed by progressively finer detail over subsequent frames. Again a known property of the HVS is being exploited, namely that fine detail cannot be absorbed by an observer immediately after such a change. Thus, this coding problem is overcome with little subjective loss of quality in the replayed sequence.

**Trading Frame Size and Frame Rate** The frame size, frame rate, decoding rate and required frame compression ratio can all be traded off against one another according to the demands of the application. For example, the frame rate can be reduced from 25 fps to 12½ fps[7] and a choice made between doubling the frame size, maintaining the same frame size but halving the compression requirements (by halving the decoding rate) or a combination of the two. As an example of the third option, the frame size might be increased by a factor of approximately 1.7 to television resolution ($768 \times 576$) and the decoder rate reduced by a same factor to yield a corresponding reduction in compression requirements. The ability to set the decoding rate on a per-segment basis allows it to be reduced only for the areas of a frame which need it most (i.e. those which compress badly). Reducing the Decoder rate allows insertion of less well compressed data into parts of the frame (and at a decoding rate of approximately 1/4 full speed this corresponds to inserting uncompressed image data into a segment).

## 4.7 Formats

The sequence, frame and segment formats as viewed from the lowest level of the control hierarchy are illustrated in Figure 4.15. A different view is presented at

[7]Still a useful rate—recall that much commercial film animation is shot "on twos" at 12 fps.

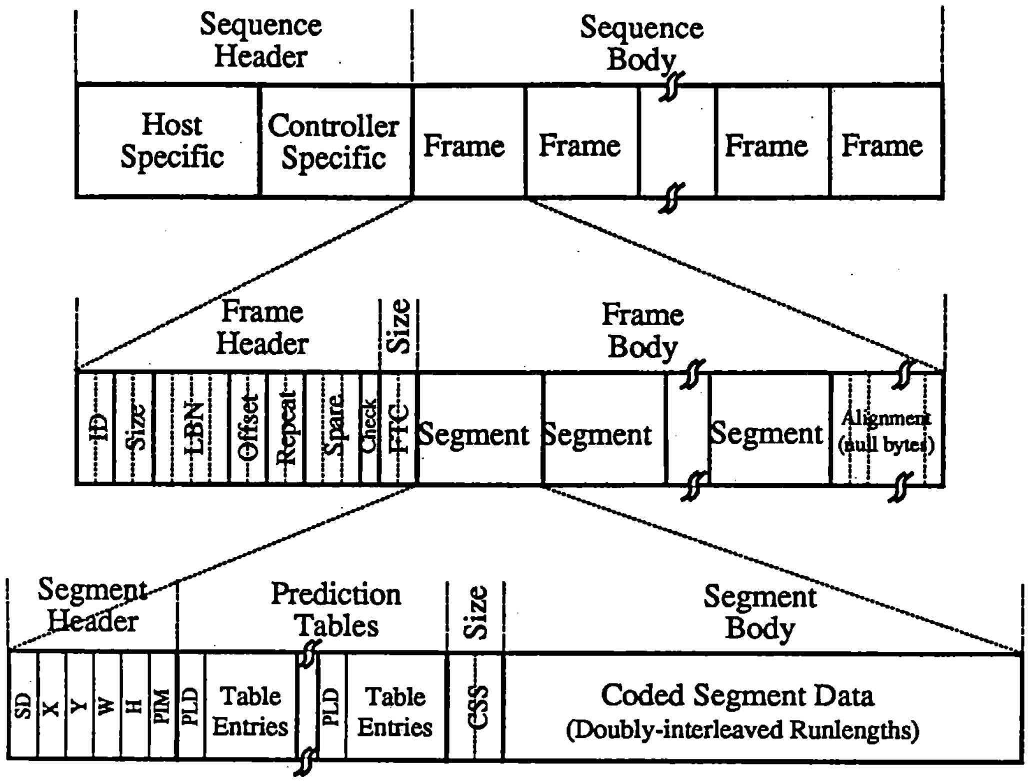


Figure 4.15: Sequence, frame and segment formats

higher levels in the hierarchy. For example, recall that at higher levels frames appear as atomic entities and details of frame segmentation are unknown.

A complete piece of animation is termed a *sequence* and comprises of the constituent frames, forming the sequence body, preceded by a sequence header. The sequence header is divided into two parts, both of which are application specific. The first part is read and interpreted by the Rainbow II host (via the Server Controller) and contains all the information required by the host for processing the sequence. This includes such things as image blending and look-up tables for use within Garland. The Server places no limits upon the size or content of this block; it is simply transferred from disc to the Host prior to sequence replay. The second part contains sequence information that is used by the control code running on the Server Controller. This is global information pertaining to the whole frame sequence (e.g., the total number of frames in the sequence, the default replay frame rate, etc.). This information too is application specific, depending upon the control required over sequence replay. For straightforward replay the information required is minimal but for more complex control further details are required (such as frame address tables identifying frame locations on disc (See Chapter 7)).

Each frame has a 16 byte header containing control information for that frame. It is this header which is stripped off by the FTL logic and placed in the FCB as the frame arrives from disc. This embedded information is used for per-frame control of the sequence replay. The format and content of the header is again application specific, depending upon the level of control required over the replay process; the following discussion illustrates a typical format. The frame is identified by a 16 bit frame number. The next two bytes give the compressed size of the frame, which is used, amongst other things, in the construction of a table identifying the position of each frame held in the CFB (see Chapter 7). The next six bytes identify the disc location of the start of the frame in the form `logical_block_number.byte_offset`[8]. A typical control strategy is for the controller to dynamically retain the disc locations of the previous $n$ frames for the purposes of error recovery (See Section 7.7). The repeat count gives the number of frame times that the frame is to be shown for; only one copy of a repeated frame is physically stored on disc. Finally "check" is a checksum which allows the controller to verify that the header has been recovered properly from the disc. This is part of the error detection mechanism discussed in Section 7.7. The Frame Transfer Count (FTC) is a 16 bit value that is loaded into a hardware counter and allows the FTL microcontroller to detect the end of the frame transfer (and therefore report the arrival of a new frame header to the Controller). The frame body may be followed by a number of null bytes which are used for internal alignment within the Frame Decoder at frame boundaries. The presence of these bytes is again dependent upon the control strategy adopted (Chapter 7).

Each segment is defined by a 6 byte header consisting of a Segment Descriptor (SD), origin (X,Y) and size (W,H), and a Plane Idle Mask (PIM). The origin and size values specify the top 8 bits of 10-bit fields, with the least significant 2 bits reset automatically. Thus the placement and size of segments is restricted to 4-pixel boundaries, not a severe limitation given that segmentation is intended to be a coarse-grained mechanism. The header is optionally followed by prediction table data. Between 0 and 8 tables can be loaded, with no restriction placed upon load order. A prediction table is specified by a load descriptor (PLD) which identifies the table and load size, followed by the table entries. For each table between 4 and 256 entries can be loaded (in power of two steps), corresponding to the size of prediction template chosen (see Section 6.1.2). The compressed segment size (CSS) is a checksum used during decoding to verify that the decoding was completed successfully. The segment header, prediction tables and CSS are all processed by the Frame Decoder microcontroller as discussed in Chapter 6. The segment body represents the actual compressed segment data, contained in up to 64 interleaved runlength streams as described earlier.

[8]A SCSI disc is viewed as a sequence of logical blocks rather than as an arrangement of cylinders, tracks and sectors, and a particular sector is identified by a logical block number in the range 0...NUMBLOCKS-1. The offset records the origin of the frame within the block; recall that a frame does not have to be aligned to the start of any disc structure such as a block or track.

# Chapter 5

# The Frame Transfer Logic

## 5.1 Introduction

In this and the next chapter the hardware implementation of the main Server logic is considered in more detail. Referring back to Figure 4.5, (page 61), recall that this logic divides naturally into two distinct parts: (1) the Frame Decoder and its associated microcontroller; and (2) all the control and interfacing logic prior to the FDEC_queue, collectively known as the Frame Transfer Logic (FTL). The operation of the Frame Decoder is discussed in Chapter 6, whilst in this chapter the purpose and implementation of the FTL are considered in more detail.

The FTL has three main functions. Firstly, it provides a mechanism for defining the *configuration* of the Animation Server, where a configuration describes how frame and control information are to be routed through the Server. The three principal configurations required are: *replay*, in which compressed frame sequences are read from disc and routed through to the Frame Decoder; *record*, in which the Frame Decoder is idled and new sequence data are routed to disc from the Rainbow II host; and *idle*, in which all transfers are disabled. Other configurations allow for system initialisation and for diagnostic access. Secondly, for any particular configuration the FTL handles all the low level synchronisation, arbitration, timing, flow control and error detection required for managing the set of data transfers associated with that configuration. Recall from Section 4.3.2 that all such low level processing can be handled by the FTL directly without requiring Server Controller intervention. Thirdly, it implements the control and status interface to the Server Controller. This includes interpretation of Controller commands and configuration requests, and implementation of the event mechanism discussed in Section 4.3.2.

## 5.2 The FTL Processing Model

The model presented to the Server Controller by the FTL is that of a simple circuit switching node within a data network composed of the disc, the CFB/FCB, the Frame Decoder and the Server Controller itself. This is illustrated in Figure 5.1.

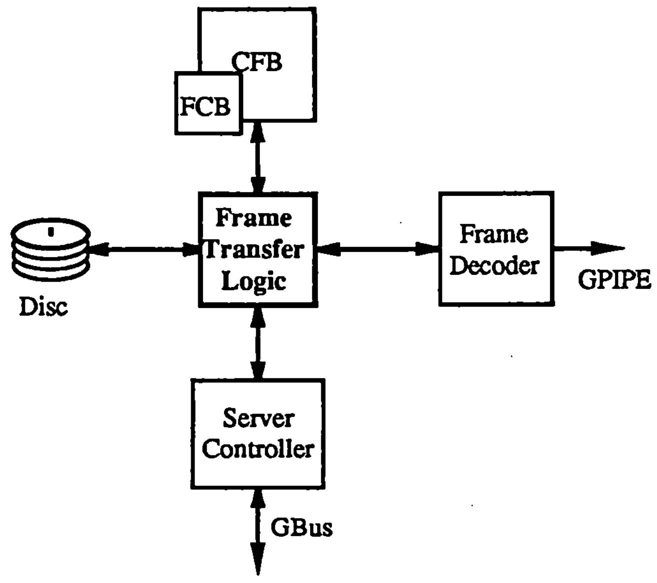


Figure 5.1: Logical Arrangement of the Animation Server as a Data Network

Each of the terminal nodes acts as a data source or sink (or both), depending upon the selected configuration.

The command interface presented by the FTL enables the Server Controller to establish logical connections between nodes. Thus, for example, disc data can be routed to the CFB (disc→FTL→CFB) and buffered frame data routed to the Frame Decoder (CFB→FTL→Frame Decoder). Typically, a number of such connections are managed simultaneously by the FTL. Further, any given set of logical connections, when viewed at a higher level, defines a particular configuration of the Server. As an example consider frame replay where three connections are required: (1) disc data are routed to the CFB/FCB; (2) buffered frame data in the CFB is routed to the Frame Decoder; and (3) frame header data in the FCB is routed to the Server Controller. All of the required configurations can be obtained by establishing the appropriate set the logical connections. Examples of Server configurations are given in Table 5.1.

Each logical connection is managed independently of any others, with the FTL scheduling and arbitrating between competing data transfers. The transfer of a data item proceeds only when a relevant set of conditions have been met. The details of how this control is achieved are discussed in more detail in Section 5.5. Each logical connection can also be controlled independently by the Server Controller and it is with this mechanism that the Controller achieves fine-grained control over the Server's operation. During frame replay, for example, the Controller may temporarily block disc→CFB transfers whilst it re-programs the operation of the disc; examples of such operations are examined in Chapter 7. CFB→Frame Decoder transfers are unaffected during this processing. The Frame Decoder continues operation as normal, unaware of the interruption in disc data transfer and there is no disruption to frame replay. Central to such processing is the CFB, whose purpose can now be defined more fully.

| Configuration | Logical Connection(s) | Comments |
|---|---|---|
| REPLAY | Disc → CFB/FCB<br>CFB → Decoder<br>FCB → Controller | Transfer frame data to Decoder via CFB, frame headers to Controller via FCB. |
| RECORD | Controller → CFB<br>CFB → Disc | Record frame data to disc via CFB. |
| ACCESS DISC | Controller → Disc | Initiate a Disc transfer, recover disc status, record frame data directly to disc, etc. |
| HOST REPLAY | Controller → CFB<br>CFB → Decoder | Replay frame data from Rainbow II host via GBus (for diagnostic purposes or inserting title frames into sequence, etc.). |
| START UP | Disc → CFB<br>CFB → Controller | Read Disc via CFB. Used for reading sequence headers at start of replay. Also for reading back recorded sequences during editing, for example. |

Table 5.1: Examples of Frame Transfer Logic Configurations

## 5.3 The Compressed-Frame Buffer

The CFB is a large block of DRAM, organised as a circular buffer[1], which is employed for the short-term storage of frames in their compressed form. The principal use of this buffer arises during frame replay where it acts to decouple the disc drive from the Frame Decoder. Both the demand for data from the Frame Decoder and the supply of data from the disc are uneven and the task of the CFB is to guarantee an uninterrupted supply of compressed frame data to the former under all circumstances and operating conditions. More precisely, the role of the CFB (during frame replay) can be summarised as follows:

1. It smooths out unevenness in disc data transfer. By the nature of disc operation, the supply of data from a drive's read head is erratic. Firstly, data transfer is interposed between transfer of formatting information (sector headers, checksum fields, etc.). Secondly, discontinuities are introduced by each head and cylinder switch. Ordinarily, much of this uneveness is reduced by buffering resident within the disc controller itself and by careful organisation of track layout. However, in the case of the Animation Server this controller buffering is insufficient for two reasons:

   (a) To guarantee optimal use of disc bandwidth it is arranged for the capacity of the link between the controller and Animation Server to exceed

[1]Mercer [Merc73] has demonstrated that for sustained, high-speed data transfers, such as occur here, a circular buffering scheme is a more natural and efficient choice than a double- or triple-buffered scheme.

that of the disc to controller link. The design of the FTL is optimised to support sustained data transfer at the full rate of the SCSI bus. In these circumstances the rate at which the disc controller's buffer is read continually exceeds the rate at which it is written, nullifying the effect of the buffering.

(b) Data transfer may be lost for significant periods (several frame times) whilst the disc controller attempts to recover transparently from read, seek and other errors which arise during normal disc operation. Intelligent error and defective media handling is a feature common to all SCSI device controllers. This is discussed further in Section 7.7.

2. It introduces flexibility by easing the requirement that each sequence be written to a single contiguous area of disc. A sequence can be dynamically assembled during replay from a number of sub-sequences located in different areas of the disc, the CFB absorbing the loss of disc transfer at each sequence discontinuity. More generally, the same mechanism allows the frames of a sequence to be replayed in an order other than that in which they were originally recorded. This provides a basis for certain "special-play" modes, video editing and support for handling unrecoverable disc errors, as discussed more fully in Chapter 7.

3. It facilitates the *pre-fetching* of frames ahead of the one being displayed. Any pre-fetched frame held in the CFB is available for immediate display. This therefore provides a facility for instantaneous local seeking without incurring any disc seek or rotational latencies. This mechanism can be used, for example, in the recovery from one form of frame decoding error, in which processing of the current frame is abandoned and the Frame Decoder is skipped to the start of the next frame (see Section 7.7)

   Prefetching is also used to improve the frame throughput of the Server. Compressed-frame transfer from disc is not related to frame-time boundaries; transfer of frame $n+1$ commences immediately on completion of the transfer of frame $n$, independent of the state of the Frame Decoder or frame replay. This allows the transfer of more poorly compressed frames to be compensated by the transfer of better-compressed ones. The CFB can be read at a greater rate than it is written and the Frame Decoder has the capacity to process source frames which have not achieved the required (average) degree of compression[2]. This means that a poorly compressed frame, transferred to the CFB from disc over several frame times, can be fed through the Frame Decoder in a single frame time. One example of this is with a temporally differenced sequence in which frames prior to an abrupt scene change, which normally compress well, can compensate for the frame immediately after, which often compresses badly.

[2]Actually, the Frame Decoder supports the sustained decoding of *uncompressed* frame data.

4. It simplifies the overall design of the Animation Server by allowing the disc and Frame Decoder to be controlled independently of one another. The adoption of an unmodified disc drive in conjunction with image compression (which results in variable sized frames) means that there is no direct or constant relationship between disc transfer and video refresh. Consequently maintaining close synchronisation between these two activities is impossible and proper replay performance can only be achieved with the provision of buffering.

## 5.4 Implementation of the FTL

An overview of the organisation of the FTL is given in Figure 5.2. Note that this figure encompasses the CFB and FCB (which were disjoint in Figure 5.1); in practice these elements are closely coupled to the FTL and their inclusion here, and in the following discussion, is both convenient and logical. Hardware in Figure 5.2 divides broadly into two categories: (1) interfacing logic; and (2) the control and routing logic associated with the management of logical connections. These categories are considered in turn.

The FTL has four interfaces, one for each of its "network links" in Figure 5.1[3]. These are:

1. *VMEbus Interface:* Server Controller access to the FTL and Frame Decoder is achieved via the VMEbus interface. The interface forms a *VME Slave subsystem* (VMEbus terminology) which appears to the Server Controller as a block of 16 registers mapped into the VMEbus' short I/O address space—see Figure 5.3. Registers 0–7 are the address, data and control registers related to the operation of the FTL itself. The functions of these registers are discussed in the following sections. The formats of the control registers are illustrated in Figure 5.4. Registers 8–15 relate to the control of the SCSI bus as outlined below. The interface also implements the event mechanism used for the asynchronous reporting of status and error conditions. VMEbus vectored interrupts are employed for this purpose and an event coded as the least significant 4 bits of an 8-bit interrupt vector.

2. *SCSI bus interface:* Access to the disc is achieved via the SCSI bus interface. This interface is based around an NCR 53C80 chip, configured to act as a SCSI Initiator[4], and operates in one of two modes. Its polled I/O mode is used for the transfer of SCSI command blocks, status information and some data blocks between the Server Controller and the disc (via VMEbus and I_bus). In this mode the SCSI protocol management and timing are

[3]Note that the layout of Figure 5.2 is organised such that there is a direct correspondence between the positioning of these interfaces and the orientation of the FTL's four links in Figure 5.1.

[4]The role of Initiator roughly corresponds to that of bus master in other bus systems. The terminology is adopted to reflect the fact that SCSI is a peer-to-peer protocol rather than a master-slave one.

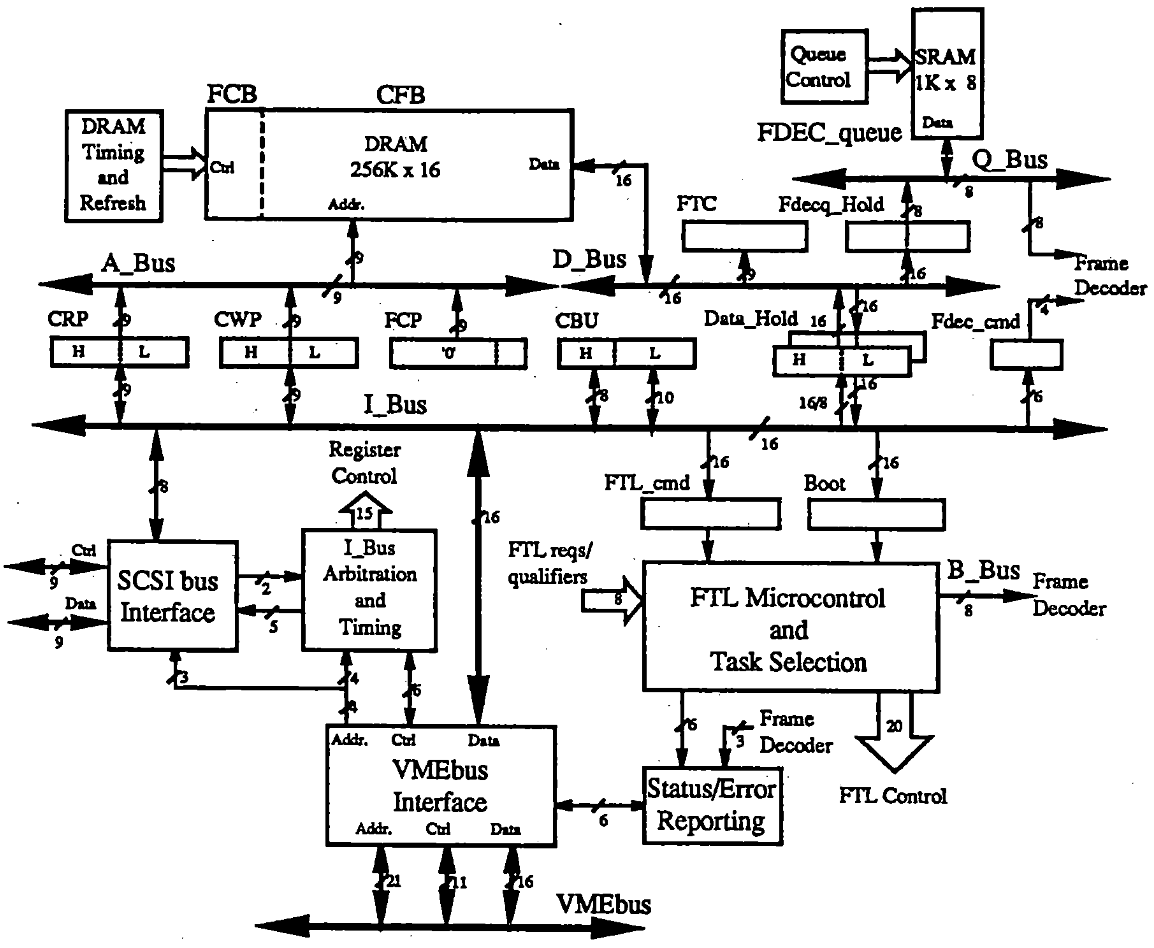


Figure 5.2: Overview of the organisation of the FTL

| | WRITE | READ | |
|---|---|---|---|
| BASE + 0 | Data_Hold _In | Data_Hold _Out | FTL Control |
| 1 | BOOT | ▨ | |
| 2 | FTL_cmd | CRP (low) | |
| 3 | FDEC_cmd | CRP (high) | |
| 4 | CRP/CWP (low) | CWP (low) | |
| 5 | CRP/CWP (high) | CWP (high) | |
| 6 | CBU (low) | CBU (low) | |
| 7 | CBU (high) | CBU (high) | |
| 8 | output data reg. | current SCSI data | SCSI bus Control |
| 9 | initiator cmd reg. | initiator cmd reg. | |
| 10 | mode reg. | mode reg. | |
| 11 | target cmd reg. | target cmd reg. | |
| 12 | select enable reg. | current SCSI status | |
| 13 | start DMA send | bus & status reg. | |
| 14 | DMA target receive | input data reg. | |
| 15 | DMA init. receive | reset parity/interrupt | |

Figure 5.3: The FTL register interface

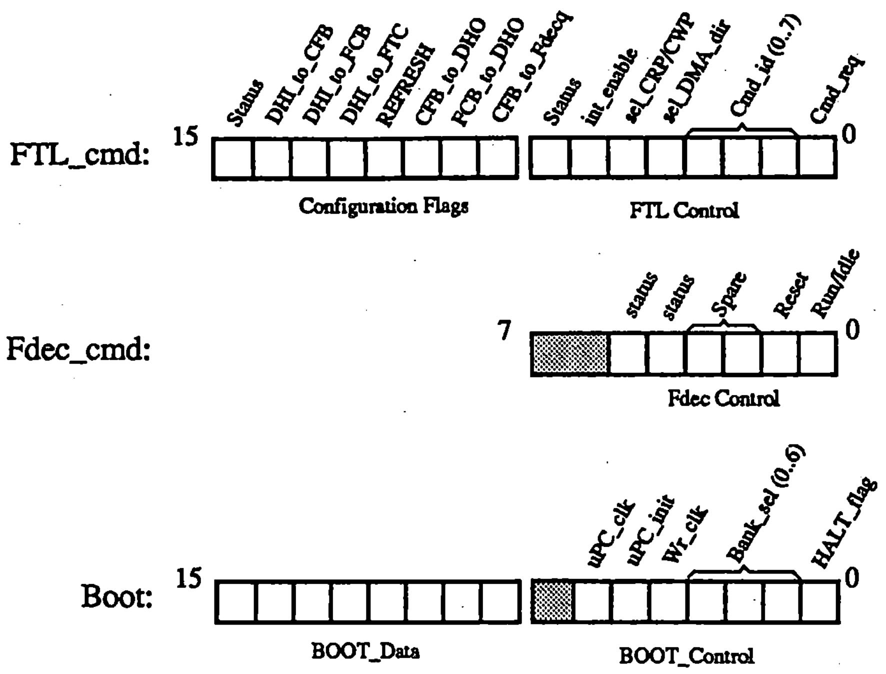


Figure 5.4: Formats of FTL control registers

performed directly by the Server Controller via registers 8–15 of the VMEbus interface; these registers are direct mappings of 53C80 registers, details of which are not discussed here (they can be found in [NCR]). For data transfers between the disc and CFB the interface is switched to its DMA mode. Here, data transfer requests received from the SCSI bus are translated to DMA requests which are directly processed by the FTL hardware to perform the required transfer.

3. *CFB/FCB Interface:* In practice the two (separate) buffers, CFB and FCB, are implemented in a common 1/2 MB block of dynamic RAM (DRAM), organised as 256K × 16-bit words. The FCB occupies the least significant 512 words (1K bytes), and the CFB the remainder. The CFB/FCB interface comprises one 18-bit usage register and three 18-bit addressing registers together with logic for DRAM timing and refresh. These elements are directly controlled by the FTL microcontroller (see Section 5.5). The current CFB reading and writing positions are maintained by the CRP and CWP registers, respectively. The current CFB usage is recorded by the CBU register; full and empty conditions reported by this register are used by the FTL microcontroller to control access to the CFB. The FCP is a dedicated address register used for both the writing and reading of the 16-byte frame header fields in the FCB. The three address registers (CRP, CWP and FCP) are accessed as two 9 bit fields via A_bus for the purposes of RAS/CAS address sequencing of the DRAM.

4. *Frame Decoder Interface:* The FTL is interfaced to the Frame Decoder via a 1024-byte queue (FDec_queue) and Q_bus. The use of a queue optimises

the flow of frame data through the Server by ensuring that both the disc and Frame Decoder can always operate at their maximum rates, with neither ever needing to be blocked because of access conflicts at the CFB. From the point of view of the Frame Decoder, FDec_queue guarantees immediate access to an uninterrupted supply of compressed segment data at all times, including peak demand when the data rate is considerably higher than the average rate. Peaks of demand occur naturally over the course of a segment, in sympathy with variations in compression efficiency, and also during operations such as prediction table loading. The queue ensures that the Frame Decoder need never be idled for lack of compressed source data. From the point of view of the CFB a queue ensures that data only has to supplied to the Frame Decoder at its *average* consumption rate (and not the much higher maximum rate). Further, as CFB read accesses are no longer time-critical, write access can have the higher priority, thus ensuring that disc transfers are never blocked by Frame Decoder transfers. The queue is implemented using a 1K×8 static RAM together with three 10-bit counters for read, write and usage counts.

The remaining elements of Figure 5.2 implement the control and data routing functions of the FTL, with the two busses I_bus and D_bus forming the principal internal data path. The control of these busses is discussed in the next section. It should be clear how all the transfers of Table 5.1 can be achieved using these paths. Simultaneous logical connections are implemented using time-multiplexed data transfers and by exploiting the parallelism introduced by the use of two separate busses.

I_bus and D_bus are interconnected via Data_Hold which acts as an internal intermediate node during transfers and provides for synchronisation between the two busses. Data_Hold is implemented as a pair of independent 16-bit registers: Data_Hold_In (DHI) allows for I_bus to D_bus transfers and Data_Hold_Out (DHO) for transfers in the reverse direction. In the case of I_bus to D_bus transfers for which the SCSI bus interface is the data source, DHI performs the required 8-to-16 bit conversion; the register is implemented as two independent 8-bit fields and an incoming byte is automatically routed to the lower or upper field as appropriate (that is, depending upon whether it is the first or second byte of the 16-bit transfer, respectively).

The Frame Transfer Counter (FTC) is a 16-bit counter which is loaded from D_bus with the Frame Transfer Count field of the compressed frame format (See Figure 4.15). The FTC is initialised at the beginning of each new compressed frame transfer and decremented by the microcontroller for each 16-bit disc to CFB transfer.

## 5.5 Controlling the FTL

Data transfers across I_bus and D_bus are controlled independently by the I_bus arbitration logic and the FTL microcontroller, respectively. The switching and

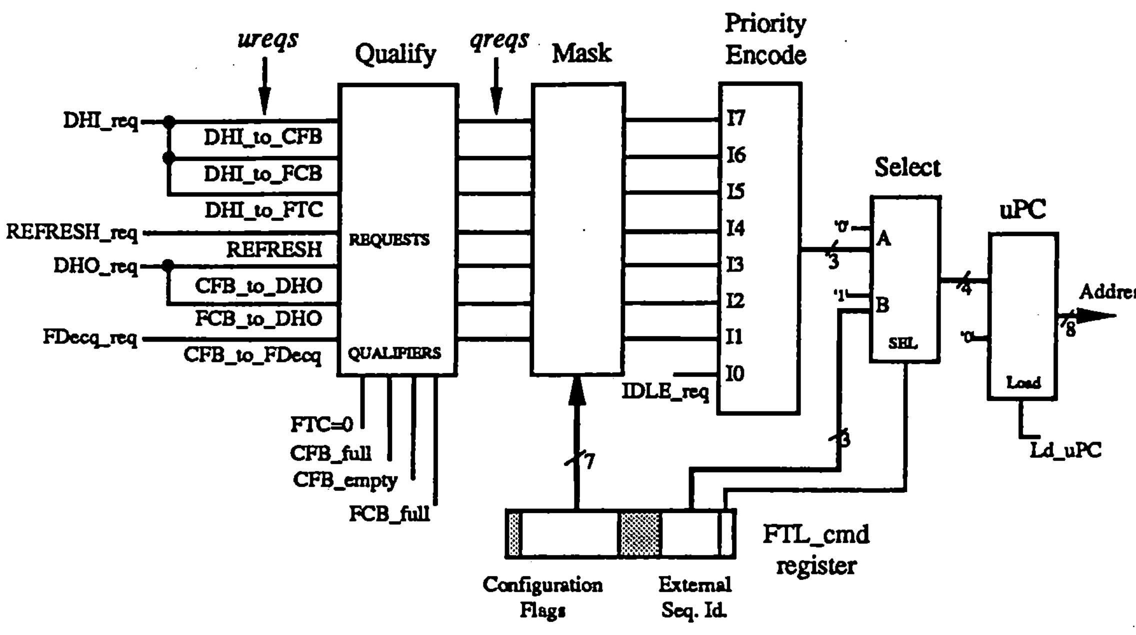


Figure 5.5: FTL microcode sequence selection pipeline.

routing function of the FTL is centered around D_bus, and the following discussion concentrates upon the control of this bus.

A D_bus transfer is executed by a short sequence of microinstructions which controls the timing, source/destination selection and the update of all registers (address, usage, etc.) relevant to that transfer. The selection of microcode sequences is achieved via a hardwired selection pipeline. Such an approach was adopted because it removes any microcode testing overheads between transfers and therefore allows immediate selection of the next microcode sequence. In particular, this allows continuous operation of the CFB/FCB DRAM by allowing arbitrarily mixed read and write accesses with no need for intervening idle cycles; such an arrangement optimises data flow through the FTL. The selection pipeline is also controllable from the Server Controller and provides both for overall configuration of the FTL and control of individual logical connections in the manner outlined in Section 5.2. Both these functions are essentially controlled by a single register, FTL_cmd, as discussed shortly. The control interface is thus very simple yet at the same time it gives complete control over the operation of the FTL.

The microcode sequence selection pipeline is illustrated in Figure 5.5. Microcode sequence selection proceeds in the following way. The pipeline is sourced by 7 *unqualified requests* (ureqs) which are mapped from four more basic signals in the way shown. The four basic requests are derived from conditions generated by Data_Hold, the Frame Decoder queue and a DRAM refresh timer. The occurrence of one of these four conditions asserts all the ureqs associated with that condition. There is a one-to-one correspondence between a ureq and a sequence of microinstructions which carries out the requested task. Thus, for example, a

DHI_to_CFB ureq occurs each time a 16 bit value is written to DHI and the corresponding microinstruction sequence transfers that value to the CFB at the current writing point (incrementing CWP, CBU and decrementing FTC in the process). Note that the implemented set of 7 ureqs is sufficient to carry out all the required control and routing functions of the FTL (for example, it supports all the operations listed in Table 5.1). The set of ureqs represents all possible D_bus transfers. At any one instant, however, not all these requests are necessarily valid and those which aren't must be blocked. The subset of valid ureqs is selected by two levels of masking operating in the following way:

1. *internal* (or "system") mask: The mask is controlled by a set of four *condition qualifiers* which reflect the operational status of the FTL. Two of the qualifiers, "CFB_full" and "CFB_empty", are generated by the CBU register and signal CFB full and empty conditions, respectively. "FCB_full" is generated by the FCP register and signals the completion of a frame header transfer to the FCB. "FTC=0" signals the expiry of the FTC counter, normally indicating the end of a compressed frame transfer. Each of the ureqs is qualified by zero, one or more condition qualifiers in either its true or inverted form. For example, a DHI_to_CFB request is qualified by NOT(CFB_full) and NOT(FTC=0). The internal mask supports two functions. Firstly, it implements low-level routing. The routing of data across D_bus at any particular time is determined by the set of active ureqs, which in turn is determined by the status of the condition qualifiers. As conditions change the set of active requests changes accordingly. This is the mechanism, for example, which strips the frame headers off newly arriving frames during sequence reply: the expiry of the FTC count causes future DHI_to_CFB requests to be blocked (see example above) and simultaneously DHI_to_FCB requests to be unblocked. Consequently, the next 16-bit word to be written to DHI (which is the first word of the following frame's header) is routed to the FCB instead of the CFB. Secondly, it provides for flow control. All requests involving transfers to the CFB, for example, are blocked when the CFB is full.

2. *external* (or "user") mask: The surviving qualified requests (*qreqs*) are further masked according to a set of configuration bits contained within the FTL_cmd register (see Figure 5.4). For each of the qreq signals there is a corresponding configuration mask bit in FTL_cmd. This mask thus gives the Server Controller individual control over each of the possible D_bus transfers and is the mechanism employed for blocking unwanted or conflicting transfers. Further, any particular pattern of flag values determines the overall configuration of the FTL. For example, "01111001" ("$79_H$") configures the FTL for replay by allowing DHI transfers to the CFB, FCB or FTC (as determined by the system mask) and CFB transfers to Frame Decoder queue. Similarly, "$4C_H$" configures the FTL for recording to disc from the Server Controller via the CFB whilst "$08_H$" idles the FTL (only DRAM refreshes allowed) (c.f. "$00_H$" which completely idles the FTL—all requests blocked).

Following masking, the resultant subset of requests defines all valid actions at that instant. These requests are priority encoded to yield the 3-bit identifier of the highest priority action. A further set of 8 microcode sequences can be executed from the Server Controller. These are used for such things as initialisation and for carrying out transfers not directly supported by the internally specified sequences. An external sequence is specified via a 3-bit sequence id contained within the FTL_cmd register. The result of the selection pipeline is then the 4-bit id of the highest priority runnable microcode sequence. When this code is loaded into the top four bits of the microprogram counter it causes a direct jump to the appropriate code sequence. Sequences execute linearly (i.e., there is no branching) and are typically very short (only one to three microinstructions). A sequence terminates itself by raising a microcode bit "Ld_$\mu$PC" which causes the microprogram counter to be loaded with the next (highest priority) sequence id. A permanently asserted and unmaskable idle request ensures that at least one sequence is always available to execute; the one-instruction idle sequence does nothing other than to immediately raise "Ld_$\mu$PC", thereby introducing a single FTL idle cycle.

Data can be transferred to or from D_bus via I_bus and Data_Hold. I_bus also provides VMEbus access to the set of interface registers (Figure 5.3) for the purposes of controlling the FTL and SCSI bus. Transfers across this bus are initiated from either the VMEbus interface or the SCSI bus interface and are managed by the I_bus arbitration and timing logic (Figure 5.2). In the case of conflict, SCSI bus transfers have the higher priority. For transfer requests involving the Data_Hold register as either source or destination the arbitration logic must poll the state of Data_Hold's status flags (DHI_req and DHO_req) in order to ensure the proper interlocking of data transfers between I_bus and D_bus.

## 5.6 Booting the Main Server Logic

At power-up the main Server logic enters a HALT state in which key elements are reset and the two microcontrollers (FTL and Frame Decoder) are stopped. In this state microcode can be loaded into the control stores of the two microcontrollers, 8 bits at a time, via the boot register (Figure 5.4). When the microcode has been loaded the logic is unhalted by resetting the halt flag in this register. The microcontrollers then enter an idle state; in the FTL microcontroller's case this is due to the idle_req being the only active request (the FTL_cmd register having been cleared at reset). The usual course is then first to execute an initialisation sequence via an external request in order to set the Server logic to a known state (flags and queues cleared, etc) and then to set address and usage registers to their initial values. Operation of the FTL may then proceed in the manner outlined earlier.

At any subsequent time the main Server logic can be halted by setting the halt flag in the boot register or by operation of a front panel reset switch. Halting the logic does not clear the microcode and the server can be restarted immediately if desired. Alternatively, from this state the microcode of either one or both of the microcontrollers can be re-loaded.

# Chapter 6

# The Frame Decoder

## 6.1 Organisation of the Frame Decoder

An overview of the Frame Decoder organisation is shown in Figure 6.1. The main decoding path consists of a pipeline formed by the *Runlength Distribution Logic*, the set of 8 *Plane Decoders* and the *Pixel Modifier Unit* (PMU). The purpose and operation of these elements is discussed in the following sub-sections. The pipeline decodes coded data contained within a sequence of segment bodies received from the FTL to produce the corresponding sequence of uncoded (raster) segments. Decoded segments are transmitted to Garland via a GPIPE link for further processing and display. This operation is controlled by the FDEC microcontroller which drives the pipeline according to a segment specification loaded into a set of registers from the segment header fields. The controller must also respond to commands issued from the Server Controller and to feedback from Garland (relating to the status of frame replay). Details of the microcontroller operation are discussed in Section 6.2.

### 6.1.1 The Runlength Distribution Logic

The Runlength Distribution Logic is responsible for the distribution of the coded runlength data amongst the 8 Plane Decoders. Distribution involves demultiplexing the first level of runlength interleave. Recall from Section 4.4.4 that the compressed format consists of a two level interleaving of coded stream information in which the outer level is a byte-wise multiplexing of the runlength data supplied from all 8 planes. The distribution process is complicated by the fact that in general the demand for runlength data varies across the Plane Decoders. For many frames there is a natural imbalance between the coding efficiency of the constituent planes, with more poorly coded planes demanding runlength data at a greater rate (and often much greater) than better coded ones. Also, one or more individual Plane Decoders may be disabled for the duration of a segment in which case no plane data are distributed to those planes at all. These two factors

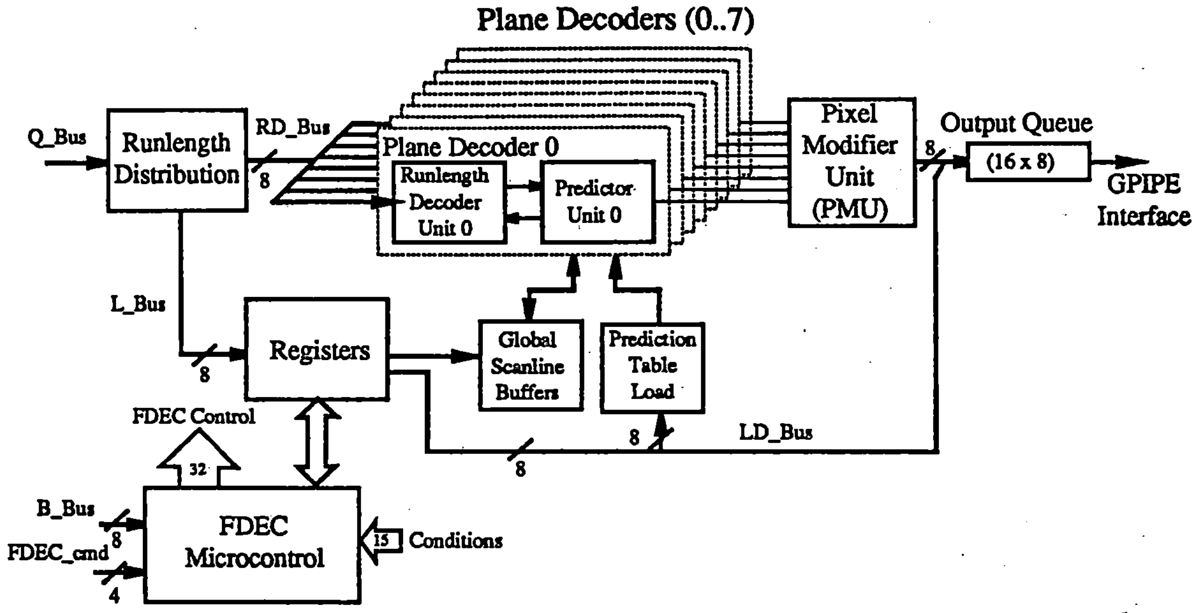


Figure 6.1: Organisation of the Frame Decoder

precluded the use of a *static* distribution scheme[1] and led to the adoption of the byte-wise interleaving scheme discussed above. The adopted scheme is simple yet meets all requirements for frame replay. In particular, it supports an arbitrary variation in distribution load across the 8 planes without introducing redundancy into the transmitted data (which would reduce replay efficiency). Moreover, it dynamically handles any change in this distribution load which may arise over the course of a segment.

The Runlength Decoder Unit (see Section 6.1.2) of each Plane Decoder signals its requirement for further input data via a request line. The distribution logic repeatedly cycles through its list of requesting Runlength Decoder Units in a round-robin fashion, distributing one input byte to each in turn. Each Decoder unit has a small input queue (16 bytes) which supplies a constant head of compressed data internally whilst the distribution logic is servicing requests from the other units. This simple distribution arrangement is facilitated by the deterministic nature of the Plane Decoder operation which allows the complexities of distribution to be shifted to the coding stage[2]. The predictable nature of the Frame Decoder means that the exact sequencing of Runlength Decoder Unit requests can be determined at the time of coding and the byte ordering of the composite coding scheme matched accordingly. Thus, the distribution logic can guarantee to supply the correct bytes to each Runlength Decoder Unit simply from the ordering of

[1]For example, one possible static allocation scheme is to assign bit 0 of each coded source byte to Plane Decoder 0, bit 1 to Plane Decoder 1, and so on.

[2]Recall the discussion of image coding requirements in Section 3.6 in which the placing of the greater complexity in the coder was proposed as a general design principal for real-time playback systems.

received requests (i.e., without the need for any additional addressing information to be associated with its input stream). This arrangement relies upon the correct synchronisation being maintained between the distribution logic and the set of Plane Decoders. Synchronisation is guaranteed so long as the distribution logic is provided with a constant supply of coded segment data (so that it is always able to supply a requested byte on demand); this function is achieved by the Frame Decoder queue in the FTL.

### 6.1.2 Plane Decoders

The central part of the decoding pipeline is formed by the array of 8 Plane Decoders. These operate in parallel and are collectively responsible for the implementation of the decoding algorithm outlined in Chapter 4. Internally, each Plane Decoder is formed from the coupling of a *Runlength Decoder Unit* to a *Predictor Unit*. In the following discussion the implementation of these two elements will be described separately, and then the operation of a Plane Decoder as a whole outlined.

#### Runlength Decoder Unit

The organisation of a Runlength Decoder Unit is illustrated in Figure 6.2. Each unit supports the decoding of 8 runlength streams simultaneously by maintaining a counter for each stream. Also associated with each stream is a single bit flag register (F) which signals whether or not the counter was originally loaded maximally (see Section 4.4.5). Upon receipt of a 3-bit stream index ($s$) from its associated Predictor Unit, the unit decodes and returns the next bit from the specified stream.

A decoding cycle executes within a single machine cycle. At the start of the cycle the *in_use* flag addressed by the stream index is examined to determine whether or not the run associated with that stream is current. If it is (in_use = 1) the corresponding internal count/F pair is selected. An 8-bit value is always read even though the stored runlength counters vary in size from 0 to 7 bits; for counter $s$ the top $7-s$ bits are ignored (see Figure 6.3). Otherwise in_use = 0 and a new run is selected from the input selection logic. Again, 8 bits are always read. Of these the least significant bit is always the F field of the new run, followed by 0 or $s$ count bits (depending upon whether or not the run is maximal), with the remaining bits undefined[3]. These formats are illustrated in Figure 6.3 (the coded stream format was discussed in Section 4.4.5). From the figure it can be seen that the internal and external formats are essentially the same and processing of the selected 8 bits is essentially source independent. (The only special case is that of a new maximal run which when first processed does not have a count associated with it.) Processing involves two distinct operations which are carried out in parallel:

[3]Actually these represent following coded runs which will be read on a subsequent occasion.

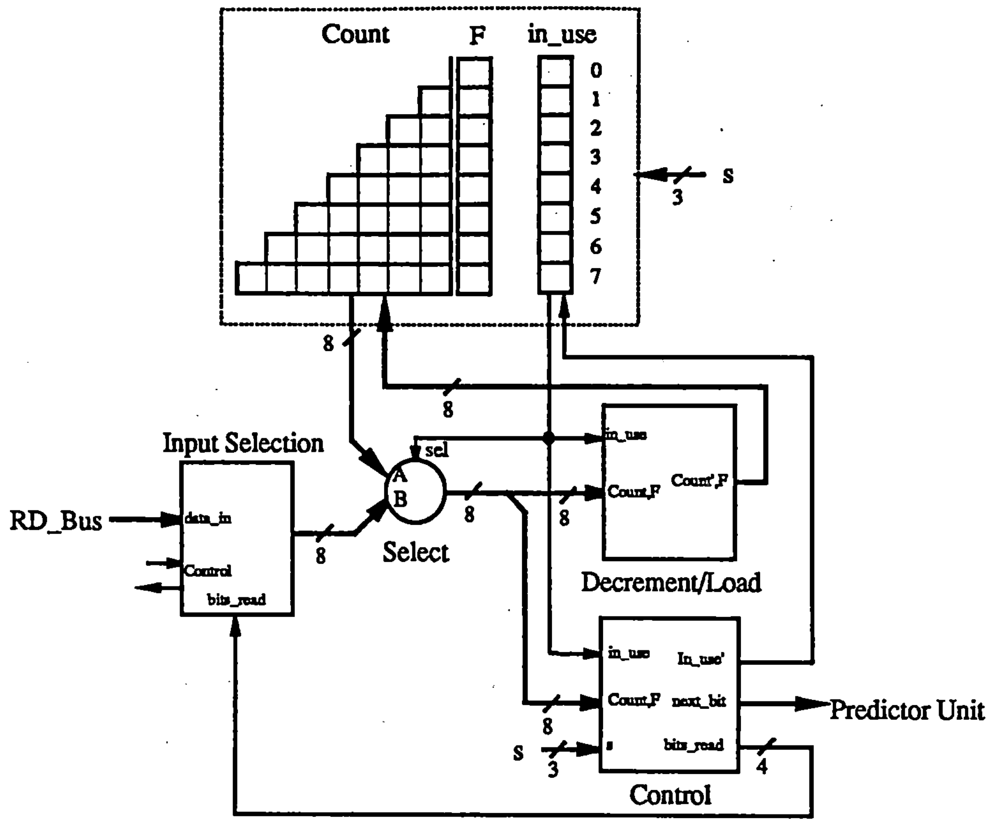


Figure 6.2: Conceptual organisation of a Runlength Decoder Unit

1. The next bit in the selected stream is decoded and the unit's status is updated. Three values are calculated in this process, namely: (1) the value of the next bit in the selected stream (0/1); (2) the new value of the in_use flag to indicate whether or not the run now being processed has terminated (0/1); and (3) the number of input bits consumed (0 to $s+1$).

2. The count field is decremented and written (or written back) to the selected counter. In the case of a new maximal run a count field has first to be generated and assigned the value $2^s$. Note that an updated count and (unmodified) F are always saved, even if the combination is not meaningful; the in_use bit records when a valid combination has been written.

The input to a Runlength Decoder Unit is a bit-wide data stream representing stream-interleaved runlengths. Whenever the unit selects new data from its input logic the 8 bits supplied represent the head of this data stream. From this data between 1 and 8 bit bits are actually consumed and at the end of the cycle these bits are replaced by shifting the bitstream up that amount. Thus, conceptually the input logic appears as an 8-bit serial-in, parallel-out (SIPO) shift register. The actual organisation of the input logic is shown in Figure 6.4. A pair of 8-bit registers holds the next 16 input bits from which an 8-bit window is selected by an alignment unit. The origin of this window is defined by an index register, and at the end of each cycle it is displaced down by the number of bits read. When the

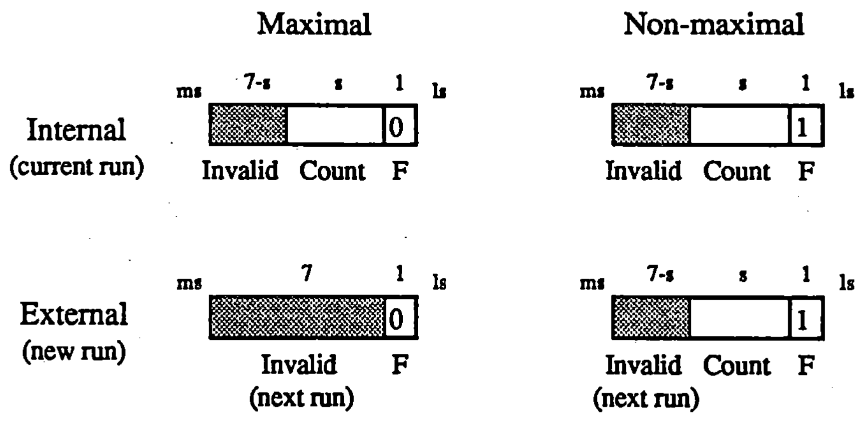


Figure 6.3: Internal and external Runlength Decoder Unit formats

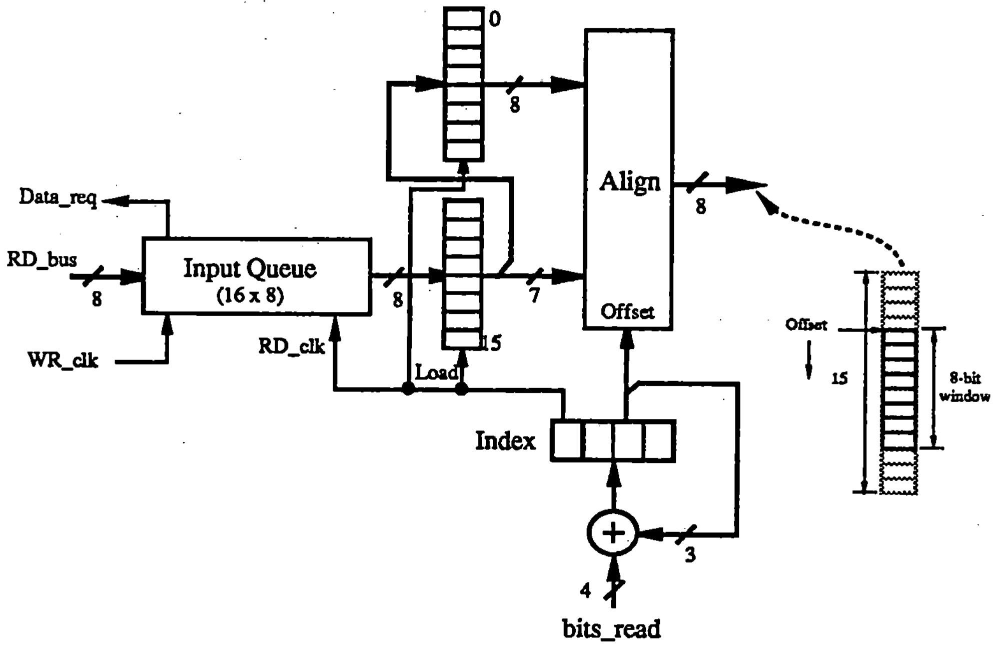


Figure 6.4: Organisation of a Decoder Unit's input logic

window origin passes the half way point the least significant byte is discarded and replaced by the most significant byte, and the most significant byte is replaced from the queue. Note that after such realignment the origin of the window is still correctly defined because the (3-bit) offset is updated modulo 8. The advantage of this arrangement is that an arbitrary shift of between 0 and 8 bits is possible in a single operation whilst only ever requiring byte reads from the decoder unit's input (where a byte is the unit of runlength distribution). The function of the input queue, in relation to runlength distribution, has already been mentioned.

The hardware detail of a Runlength Decoder Unit is shown in Figure 6.5. Note that the in_use register is implemented separately to the counter/F array; this allows the former to be cleared in a single operation (e.g., at the start of a segment), avoiding the need to access each register individually. Each time a

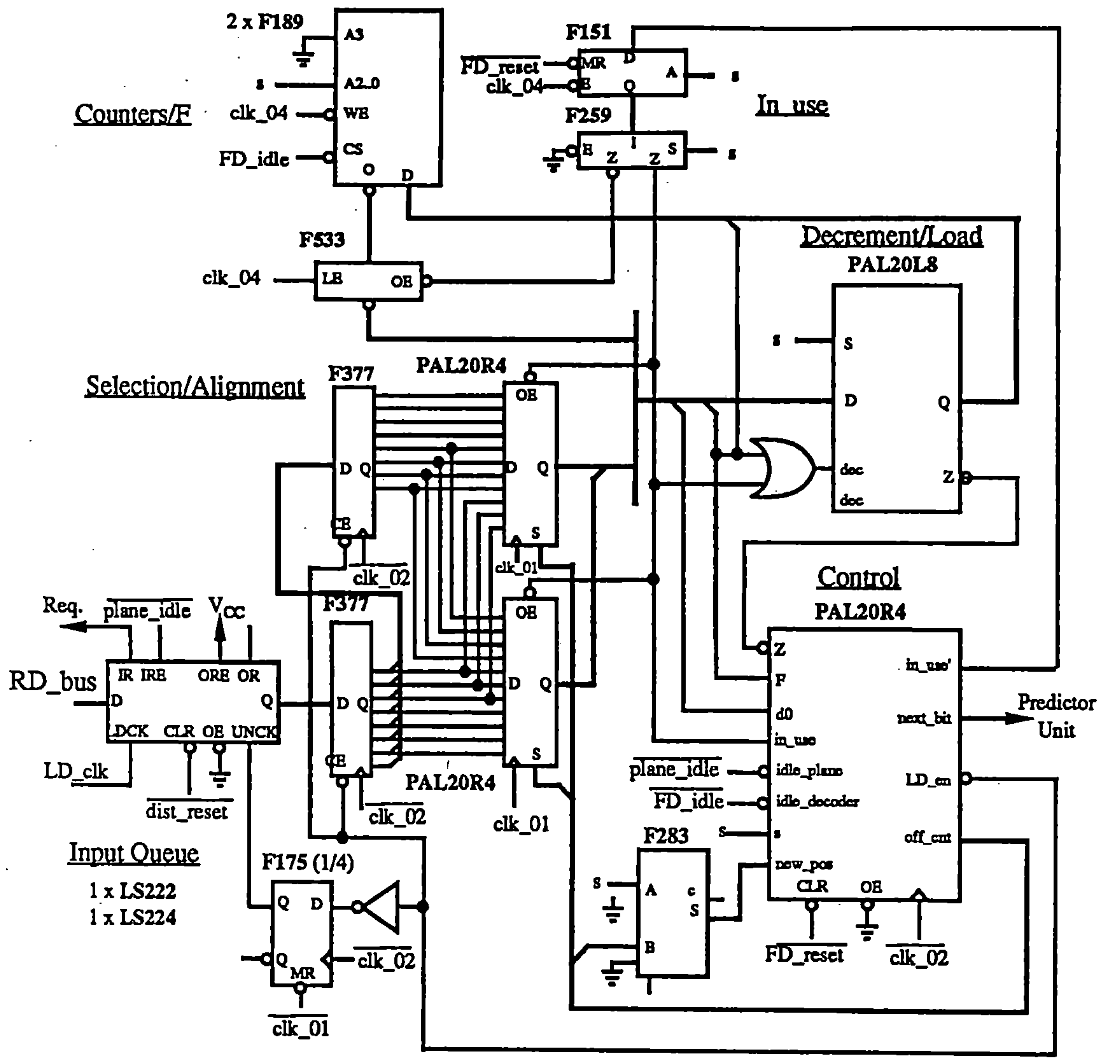


Figure 6.5: Implementation detail of a Runlength Decoder Unit (1 of 8)

stream is referenced for the first time a new runlength is read directly from the input logic and the counter/F loaded at the end of that cycle.

### Predictor Unit

The operation of a Predictor Unit is more straightforward than that of a Runlength Decoder Unit, essentially involving a table look-up on 8-bits drawn from a prediction template to obtain a prediction and corresponding stream index (see Section 4.4.5). The 8 bits are freely selected from a possible 24 taken from the current and two previous scanlines, as illustrated in Figure 6.6. This selection is made on a per-unit basis, allowing different selections to be made for each plane if so desired.

The conceptual organisation of a Predictor Unit is illustrated in Figure 6.7. The buffer is a shift register that is shifted left one position during each active cycle in order to receive a a newly decoded pixel bit. This buffer thus represents a

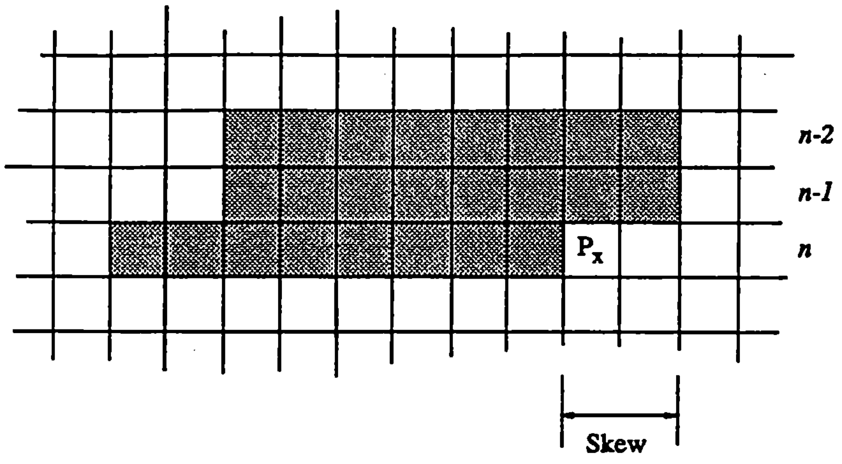


Figure 6.6: Locations of bits available for use in a prediction template

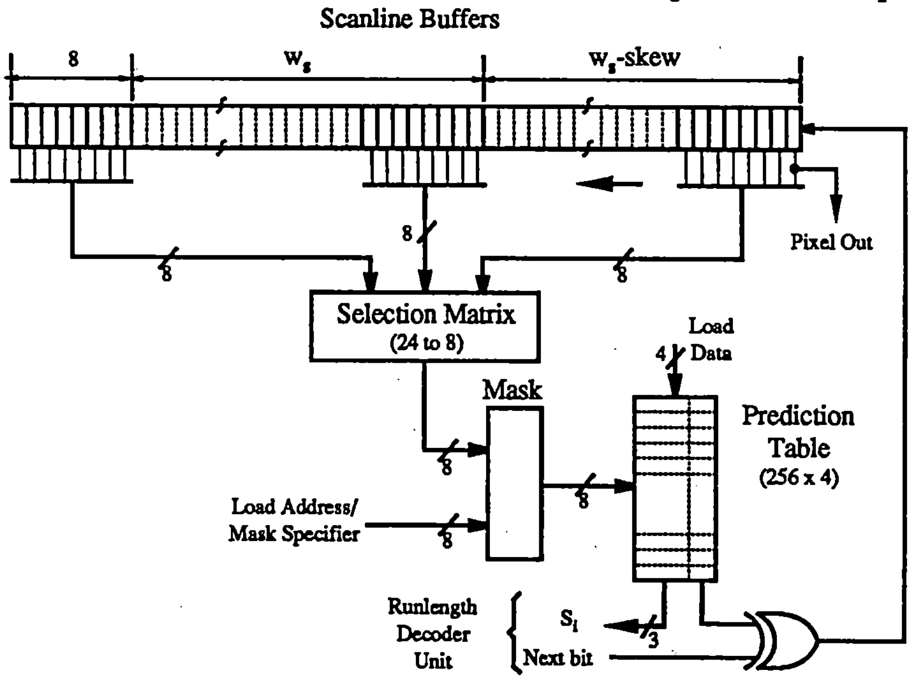


Figure 6.7: Conceptual organisation of a Predictor Unit

rolling window on the previous $n$ bits, where $n$ is dependent upon the width of the segment. The total length of the buffer is given by $(W_s-\text{skew})+W_s+8$, where $W_s$ is the segment width. Skew refers to the arrangement in Figure 6.6 whereby *previous* scanlines potentially employ template bits to the right of the current point, $P_x$. Such an arrangement improves coding efficiency by allowing the early detection of upcoming image features. Implementation of skewing requires the length of the first part of the buffer (that corresponding to the first scanline) to be reduced by an amount equal to the desired skew.

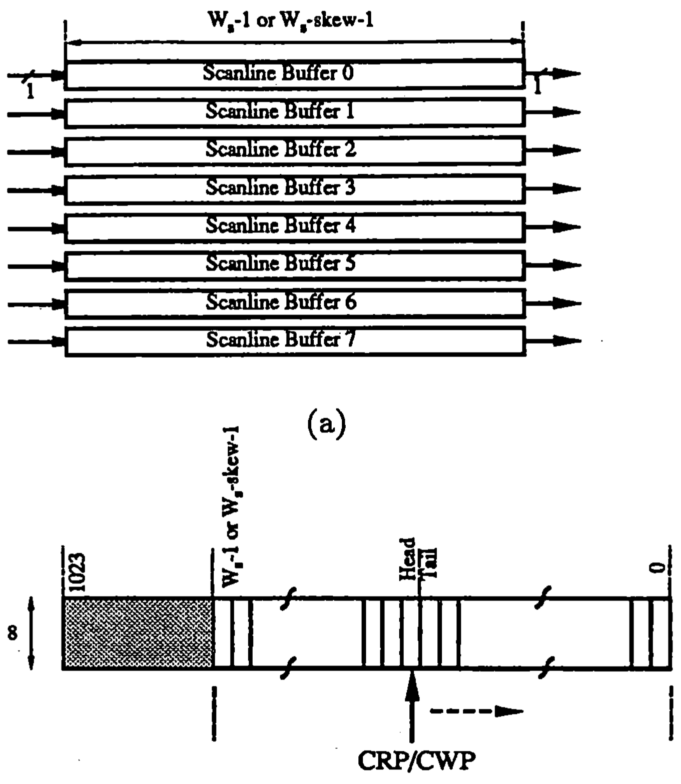
The width of a segment varies between 4 and 1024 pixels and the length of the shift register must vary accordingly. Moreover, the 24 template bits must always be selected from the appropriate points in the buffer (as determined by $W_s$ and skew) in order to maintain the correct template pattern (Figure 6.6) for all possible values of $W_s$.

In reality the implementation of the buffer as shown in Figure 6.7 is impracticable. Firstly, the implementation of such a long serial-in, parallel-out (SIPO) shift register is impossible[4]. Secondly, such an arrangement requires the ability to select three groups of 8 bits from arbitrary points within the shift register. For these reasons an alternative arrangement is employed within the Frame Decoder. The scanline buffers for all 8 Predictor Units are maintained globally, external to the units themselves. Each Predictor Unit maintains three 8-bit SIPO shift registers representing the contents of the three 8-bit "windows" that together form the 24-bit prediction template. In effect Predictor Units locally cache scanline bits from the global buffers. The global buffering is implemented with two separate buffers, each holding one scanline's worth of pixel data; only two buffers are required for the arrangement of Figure 6.6 as the 8 bits of the topmost scanline need only be maintained locally. A global buffer is implemented in a 1K×8 static RAM, organised as a circular buffer. Each global buffer simulates 8 parallel variable length serial-in, serial-out shift registers, one per plane, as shown in Figure 6.8 (a). A buffer is addressed by a single 10-bit pointer cycling in the range 0–(buffer size−1). The head and tail of the buffer are adjacent (Figure 6.8 (b)).

The relationship between global and local scanline buffers is illustrated in Figure 6.9. Note that the two global buffers require separate address pointers as their lengths differ (by the skew). During operation pixel information is exchanged both between the global buffers themselves and between the global and local buffers, in the following way. During the first half of the global buffers' cycle the two address pointers are incremented and the newly addressed values read. Incrementing a pointer corresponds to shifting the register and the value read represents that shifted out. The 8 bits from the PSL Buffer are latched into the 8 local "$n-2$" buffers whilst the 8 bits from the CSL Buffer are latched into both the 8 local "$n-1$" buffers and the PSL latch. In parallel with this the 8 Predictor Units are decoding the 8 constituent bits of the next pixel which are then simultaneously latched into both the CSL latch and the local "n" buffers. During the second half of the global buffers' cycle the values in the two latches are written to the newly freed locations in their respective memories to form the new head of the buffer. This corresponds to shifting the new data into the shift register.

The hardware detail of a Predictor Unit is illustrated in Figure 6.10. $\text{PSL}_i$ and $\text{CSL}_i$ are the two scanline update bits from the global buffers. In order that the global buffers do not have to be cleared at the start of each segment, an operation which would take up to 1024 Decoder cycles, two mask signals "SM0" and "SM1" are generated. Following the start of a segment, these mask out pixel data received from the global buffers for the duration of one and two scanlines, respectively. These signals thus create the effect of an extra two blank scanlines at the top of each segment and ensures that encoding and decoding are performed

[4]The maximum size of this buffer is 1024+1024+8 = 2056 bits.



(b)

Figure 6.8: Organisation of a global scanline buffer

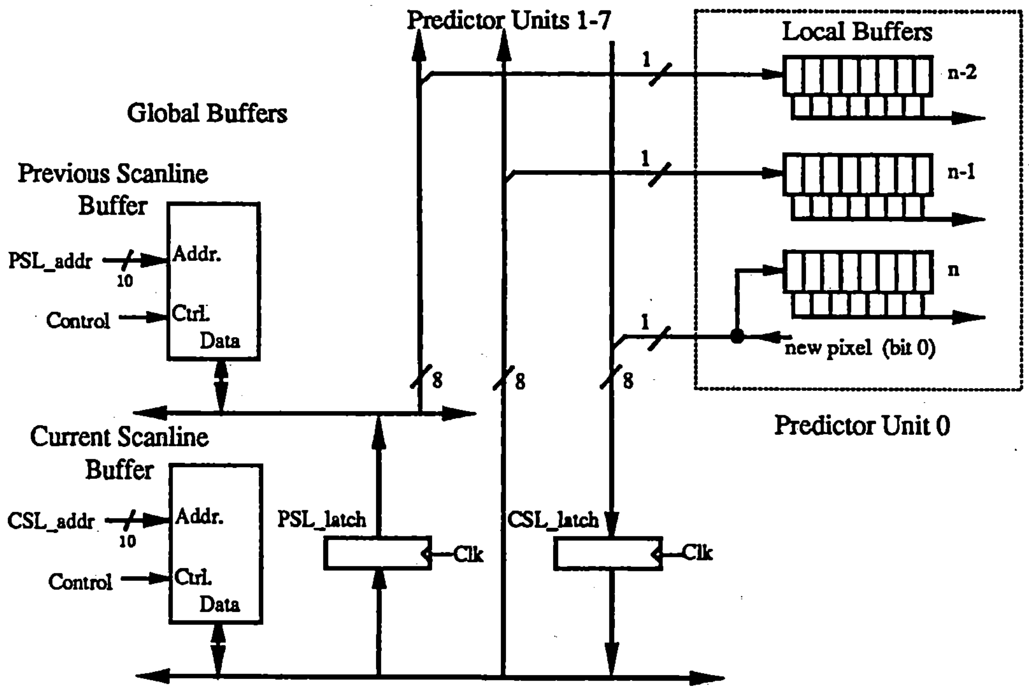


Figure 6.9: Relationship between global and local scanline buffers

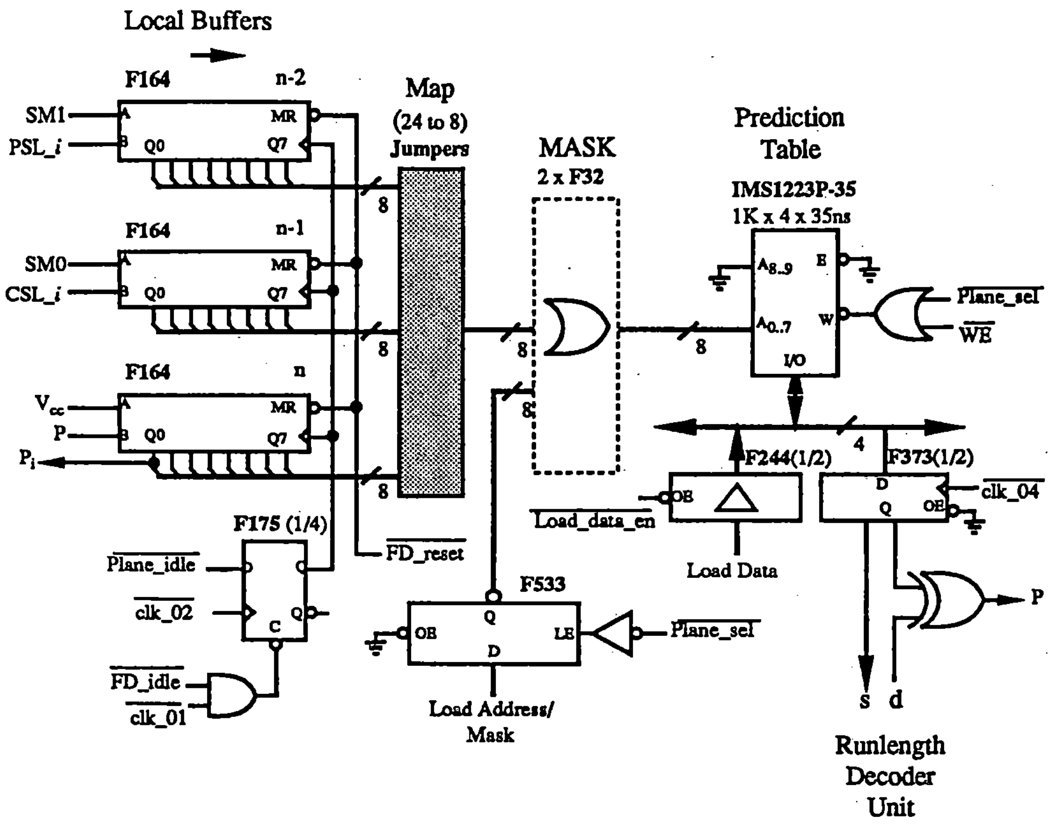


Figure 6.10: Implementation detail of a Predictor Unit (1 of 8)

in a consistent environment.

Prediction table loading is controlled by the Frame Decoder microcontroller and at the start of a segment each of the Predictor Units for which a prediction table has been supplied is loaded in turn. When a unit is selected for loading $2^n$, $2 \leq n \leq 8$, table entries are sequentially written to its prediction table, according to the table size encoded within the segment format. At the end of the load cycle the final load address reached $(2^n - 1)$ is latched and defines a mask that blocks those template bits which address the unloaded region of the table. For example, if 32 table entries are loaded then only the 5 least significant bits are used in the prediction process and the remaining 3 template bits are masked. Thus with this mechanism active prediction template bits are determined and selected automatically from the size of prediction table loaded. The mapping between the 24 possible and 8 actual template bits is defined by jumper settings within the selection matrix and no constraints are imposed upon this mapping. However, the contribution made to the prediction process by an individual bit is related to its position within the prediction template and it is clearly advantageous to select the 8 most effective bits. Further, optimal prediction performance can be obtained across all table sizes if these 8 bits are ordered by importance, with the bits making the greatest contribution assigned the least significant positions, Then, as the size of the loaded table is decreased bits these bits are the last to be lost from the

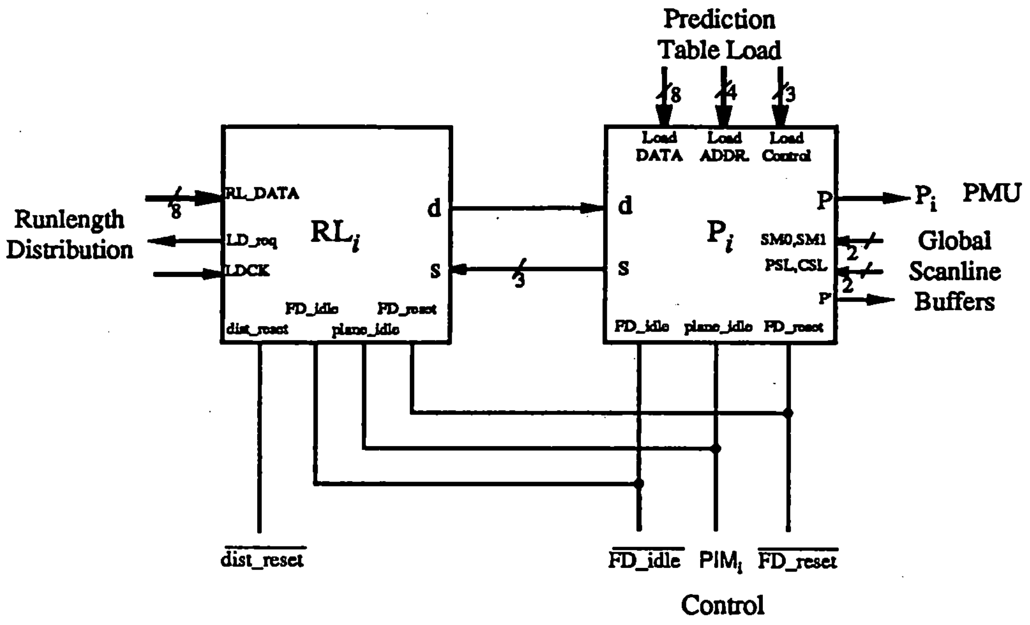


Figure 6.11: External interface to a Plane Decoder

prediction template.

## Runlength Decoder/Predictor Unit Interaction

The interconnection between a Predictor Unit and its associated runlength decoder unit is illustrated in Figure 6.11. Recall that the two units together form a Plane Decoder, responsible for the decoding of a single binary plane. The only part of the external interface which has not so far been described is the set of four control signals; these are examined in Section 6.2.

The Predictor Unit and Runlength Decoder Unit are designed to operate in parallel. However, fully overlapped operation is not possible because of an unavoidable sequential cycle in the decoding process. The Predictor Unit having made a prediction cannot proceed until its Runlength Decoder has returned the next decoded bit from the selected stream. Similarly, the Decoder cannot commence a decoding cycle until it has received a stream index from the Predictor Unit. To optimise the execution of this critical loop the Predictor and Runlength Decoder Units operate 1/4 cycle out of phase with respect to each other. This introduces a processing overlap and a degree of pipelining into the decoding operation: whilst the Predictor Unit is starting cycle $t + 1$ by performing a table look-up to obtain the next prediction, the Runlength Decoder Unit is simultaneously completing its operation of the previous cycle by updating the stream counter referenced in cycle $t$ and re-aligning runlength data within the input logic. Similarly, the operation of the 8 Predictor Units with respect to the global scanline buffers is also overlapped, again with a 1/4 cycle phase shift.

In a possible alternative arrangement of the Frame Decoder, the Runlength

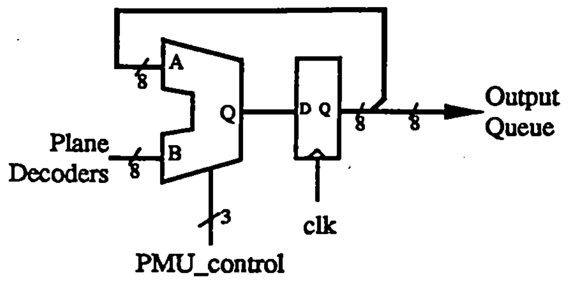


Figure 6.12: Organisation of the Pixel Modifier Unit

Decoder Units would be replaced by a set of 8 *stream decoders*, with each decoder responsible for decoding one of the runlength streams 0–7. A Predictor Unit would use the stream index obtained from its prediction table to determine the appropriate stream decoder to access. The advantages of this arrangement are that only one runlength counter is required per decoder (as opposed to 8 in a Runlength Decoder Unit) and that a decoder can pre-emptively decode its input stream without requiring Predictor Unit requests (as the stream index is implicitly known). The principal disadvantage is that a decoder has to service a variable number of Predictor Units (and quite possibly all 8) within a single machine cycle. A corollary of this is that in any one cycle up to 7 stream decoders may be idle, resulting in an uneven distribution of decoding effort. Further, the implementation of a stream decoder is more complicated than that of the alternative runlength decoder.

### 6.1.3 The Pixel Modifier Unit (PMU)

Values decoded by the Plane Decoders are post-processed by the Pixel Modifier Unit (PMU). The PMU is employed in the implementation of the pixel interpolation and spatial difference frame reconstruction modes (see Section 4.5). The PMU is a simple element whose 8-bit output is a specified function of the previous PMU output and the 8-bit output of the Plane Decoders. This arrangement is illustrated in Figure 6.12. The support of an arbitrary function between two 8 bit values would require a 64K×8 lookup table, but such a table is not employed in this implementation for two reasons. Firstly, such generality is not required for the reconstruction modes to be implemented. Secondly, the implementation of some modes requires the ability to select the PMU function on a per-pixel basis. In these circumstances it would be impossible to reload the table quickly enough and the desired behaviour could therefore only be achieved with multiple 64K tables. In the actual PMU implementation a set of 8 pre-defined functions is supported (Table 6.1) and the required function specified via a 3-bit microcode field.

The 8-bit values dispatched by the Animation Server are normally mapped into 24-bit pixels for display by a video look-up table within Garland. The values processed by the Frame Decoder therefore represent table indices. The video look-up table allows an arbitrary mapping between incoming indices and the 24-

| PMU_control | Operation | |
|---|---|---|
| 000 | 0 | [constant] |
| 001 | A | |
| 010 | B | |
| 011 | A+B | [8] |
| 100 | A+B | [3:3:2] |
| 101 | (A+B)/2 | [8] |
| 110 | (A+B)/2 | [3:3:2] |
| 111 | $FF_{16}$ | [constant] |

Table 6.1: List of implemented PMU functions

bit output values. For some of the frame reconstruction modes this arrangement is problematic since manipulating table indices may not give the desired result. In particular, new indices obtained through interpolation cannot be guaranteed to yield sensible values when used to access the video look-up table. In certain cases, such as for a greyscale function, the contents of the table can be ordered and interpolation then operates as intended. However, for a more general solution other strategies must be sought. Therefore, in addition to handling single field 8-bit values the PMU also supports a (fixed) 3:3:2 field structure, principally intended for mapping RGB fields to 8-bit Frame Decoder values. Although this fixed field structure imposes a fixed organisation on the video look-up table, it does mean that colour values can be interpolated correctly by interpolating the three fields separately (such that any carry from one field does not affect the next field). In the hardware implementation of the PMU the 8 functions are encoded within three 1K×4 PROMS, with one PROM assigned to each of the channels R, G and B.

## 6.2 Control of the Frame Decoder

Overall control of Frame Decoder operation is the task of the microprogrammed controller, FDEC_$\mu$control. The design and organisation of this controller is more conventional than that adopted for the FTL. A 256 word control store is employed, with sequencing based upon condition testing and jump addresses specified as an 8-bit field contained within the (32-bit wide) microinstruction word format.

The controller operates on two levels. Firstly, it is responsible for sequencing of the Frame Decoder operation itself. Recall that the main decoding path within the Decoder (Figure 6.1) is organised as a pipeline of processing elements, each of which has autonomous control over its internal timing and operation (as discussed in the previous section). It is the *relative* sequencing of these elements, under the direction of the microcontroller, which provides the basis for the implementation of individual frame reconstruction modes. Secondly, it allows the operation of the Decoder as a whole to be externally controlled, both from the Server Controller and from Garland. Externally the Frame Decoder appears as an atomic entity

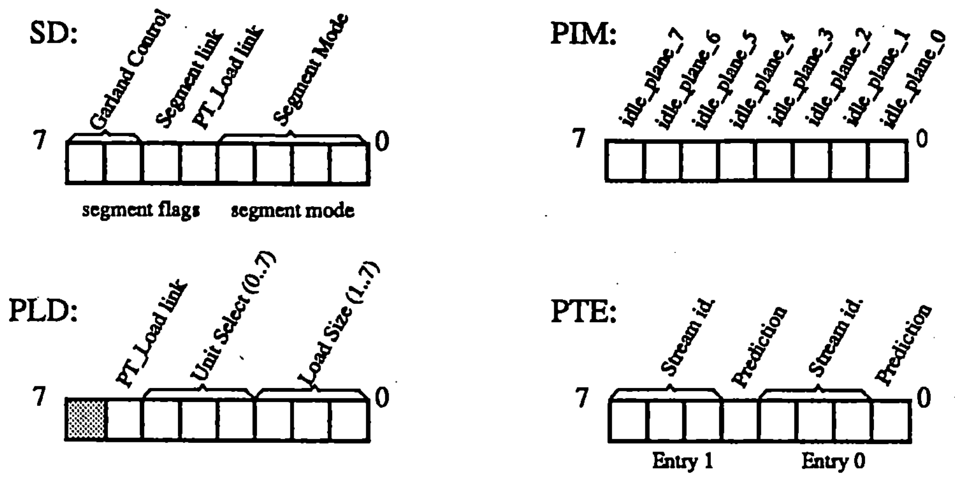


Figure 6.13: Formats of the key Frame Decoder control fields

which can itself be sequenced relative to the remainder of the encompassing *replay pipeline* (that is, the disc, the FTL, and Garland).

External control of the Frame Decoder from the Server Controller is specified by the FDEC_cmd register (Figure 5.4) which contains a set of microcode-testable flags. The operation of the Decoder's internal sequencing is controlled by a set of registers which are loaded from the information supplied in-line by the segment header (Figure 4.15, (page 80)). Registers are provided for all the header fields except X and Y—the segment origin information has no relevance within the Frame Decoder itself. The formats of the Segment Descriptor (SD), Plane Idle Mask (PIM), Predictor Load Descriptor (PLD) and Prediction Table Entry (PTE) fields are shown in Figure 6.13.

The remainder of this section outlines how the microcontroller controls the operation of the Frame Decoder during the processing of segment data.

At power-up the Decoder logic enters a halt state, from which the microcontroller may be booted using the mechanism described in Section 5.6. When the halt flag of the FTL_cmd register is reset the microcontroller enters an idle loop within which its only action is to monitor the run/idle flag of the Fdec_cmd register. When this flag is set by the Server Controller, the microcontroller commences processing on the first segment of the first frame in the Fdec_queue.

First, the segment header is read and the values transferred to their respective destination registers. In addition, the SD, X, Y, W and H fields are also placed in the Frame Decoder's output queue from where they are transferred to Garland. The Segment Descriptor contains the mode definition for the segment. The segments within a frame are chained together by link bits, each segment's link bit signalling whether or not a further segment follows. This allows an arbitrary number of segments to be included within the frame format. Similarly, within each segment any prediction tables specified are also chained together by link bits. After the header has been read any prediction tables included with the segment are then loaded in the order supplied. Each prediction table is preceded by its load descriptor which identifies the Predictor Unit in question and the table size. Tables are loaded in the way discussed in Section 6.1.2. Loading of a $2^n$ entry table only requires $2^{n-1}$ machine cycles as two 4-bit entries can be loaded per cycle. Finally, the compressed size of the segment is read and latched in the CSS register.

During this segment initialisation stage the decoding pipeline must be idled by the microcontroller and no pixels are generated. This reduces the total pixel handling capacity of the Frame Decoder. For this reason the layout of the segment format is carefully chosen to minimise the segment processing overheads. In general, parallelism is exploited whenever possible. For example, the segment control information shared by Garland is placed at the start of the header so that initialisation of Garland occurs in parallel with initialisation of the Frame Decoder (and in particular the loading of any prediction tables). At a lower level, the testing of link bits proceeds in parallel with the loading of registers, using look-ahead to avoid idling the Decoder unnecessarily.

Following the loading of the final register (CSS), processing of the segment body is initiated by the raising of a microcode bit "`new_mode`". This causes an appropriate microcode routine to be entered via a control store resident jump table which is indexed by the mode field in the segment descriptor. A microcode routine is supplied for each of the modes supported by the Frame Decoder. As previously mentioned, the implementation of a particular frame reconstruction mode is governed by the relative sequencing of the Frame Decoder pipeline and the microcontrol signals used to control this pipeline are illustrated in Figure 6.14. Asserting `FD_idle` at the start of a machine cycle forces all eight Plane Decoders to idle for that cycle. An idled Plane Decoder maintains its state from the previous cycle, and continues to output the last value it decoded (this property is required in the implementation of some of the modes). Idling a Plane Decoder does not disable the input queue of its Runlength Decoder Unit, which continues to accept data from the runlength distribution logic. Runlength distribution is idled separately by another signal `LD_inhibit`. Separate control signals are employed to allow idling of the Plane Decoders whilst the distribution of runlength data from the FTL continues unabated. Individual Plane Decoders can be idled for the duration of a segment by setting the appropriate bit in the Plane Idle Mask (PIM). In this case disabled units do not request any input data from the distribution logic. `FD_reset` is asserted at the start of each segment to reset the Plane Decoders to a known state (counters and scanline buffers cleared and input data re-aligned). The control signals shown in Figure 6.14 together allow implementation of all the modes discussed in Section 4.5 (Page 73)[5].

Two counters, initialised from the W and H fields of the segment header, are maintained by the microcontroller and allow for end-of-segment detection. In addition, the CSS counter is decremented for each byte of runlength data transferred to the Plane Decoders. If the end of decoding does not coincide with reaching the end of the segment body then a *segment framing error* is reported to the Server

[5]With the exception of temporal differencing, which is implemented in Garland (see Chapter 7)

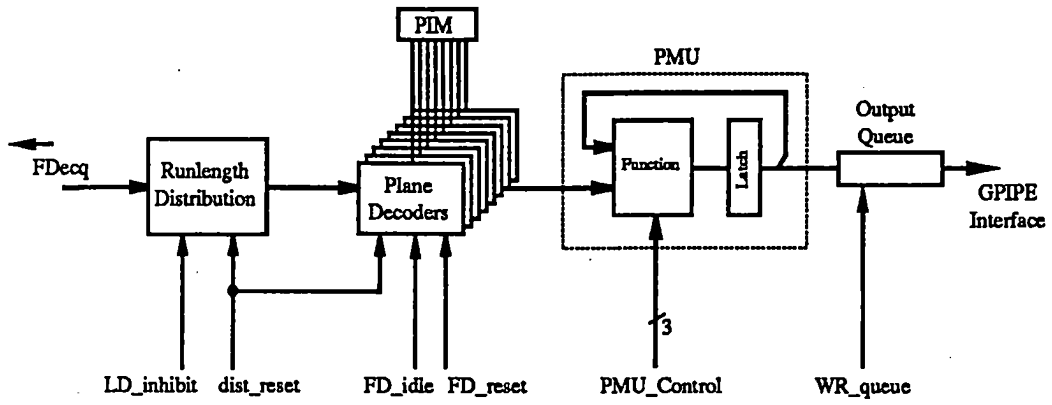


Figure 6.14: The decoding pipeline and its microcode control signals

Controller[6]. Framing errors are caused by either an original frame misalignment (at the start of decoding) or by corruption to one or more bits within the segment data. In either case the lack of redundancy within the coded form means that the decoding process is highly intolerant of errors contained within its source data. A single-bit error can have a devastating effect upon the sequence being processed. Framing errors are handled by the Server Controller as part of its error recovery strategy (see Chapter 7).

On completing the decoding of a segment the microcontroller commences immediately upon the processing of the following segment. Note that the next segment may be the first one of the *following* frame. The Frame Decoder is unconcerned with frame boundaries and the frame structure of Figure 4.15 is transparent at this level; a complete animated sequence is essentially treated as a continuous sequence of segments[7]. Synchronisation of the Decoder's operation with frame display and update in Garland is achieved via the output queue and GPIPE link, in the following way. The output queue is designed to connect to a "standard" GPIPE interface [King88] with a minimum of interfacing logic. Values queued by the Frame Decoder are available for immediate transfer whenever a GPIPE request is received. The queue maintains a "half full" flag which is monitored by this microcontroller. Whenever this flag is asserted the microcontroller forceably idles the Frame Decoder until the queue usage again falls back below this point[8]. If the output queue becomes empty then any GPIPE transfer request blocks until a further decoded value becomes available. Thus, the output queue and GPIPE

[6]The term "framing error" here is taken directly from the field of serial data transfer, to refer to a misalignment within a serial bitstream. This should not to be confused with the term *video frame*, a completely separate concept.

[7]However, the microcontroller does detect the final segment of each frame in order to be able to report a "frame_decode_complete" event to the Server Controller.

[8]With this "unscheduled" (non-deterministic) idling it is impossible to calculate in advance the byte ordering in the composite coded runlength stream required to maintain the correct synchronisation between the distribution logic and the 8 Runlength Decoder Units, as discussed in Section 6.1.1. Consequently, in this case the whole Frame Decoder must be idled (that is, including the runlength distribution logic).

link facilitate flow control between the Frame Decoder and Garland. Further, this is the mechanism by which frame synchronisation is achieved. The operation of the Frame Decoder is entirely demand driven—values are decoded whenever there is space in the output queue to accommodate them—and the Decoder has no concept of frame times or of frame-time boundaries. When a frame switch occurs in Garland an Image Plane update task running there is awoken and starts demanding data from its GPIPE input. This continues until all the pixel data for that frame has been received and written to the frame buffer. The task is then suspended until the next frame switch, when the process repeats. Thus, data demand is synchronised to frame update. The only consideration for the Frame Decoder is that it must guarantee to supply all the data for the frame within the time available (but this requirement can be met at the time of coding).

The Frame Decoder can also be idled from the Server Controller by clearing the run/idle bit in the `FDEC_cmd` register. This flag is examined by the microcontroller after it has completed the processing of the last segment in a frame and when cleared causes the idle loop to be re-entered.

## 6.3 Physical Organisation of the Server Logic

The physical organisation of the Animation Server logic is shown in Figure 6.15. The main Server logic is partitioned between the FTL/CFB (Chapter 5) and the Frame Decoder logic (Chapter 6) over two triple-height Eurocards (14.44 × 11.02 inches). Such a split represents a natural division of functionality. These boards are mounted in the same rack as Garland. The VMEbus interface logic is contained on a separate double-height Eurocard (9.19×6.30 inches) mounted in a second rack. This rack, which has a VMEbus backplane, contains the 68020 (Server Controller) card, the 68020's 5MB main memory card and the CFR interface card. The disc and its controller are also mounted in this rack, purely for reasons of mechanical convenience (that is, there is no electrical interconnection).

The physical organisation of the FTL, Frame Decoder and VMEbus interface Eurocards is shown in colour Plates 1–5 at the end of this dissertation (Pages 157–161).

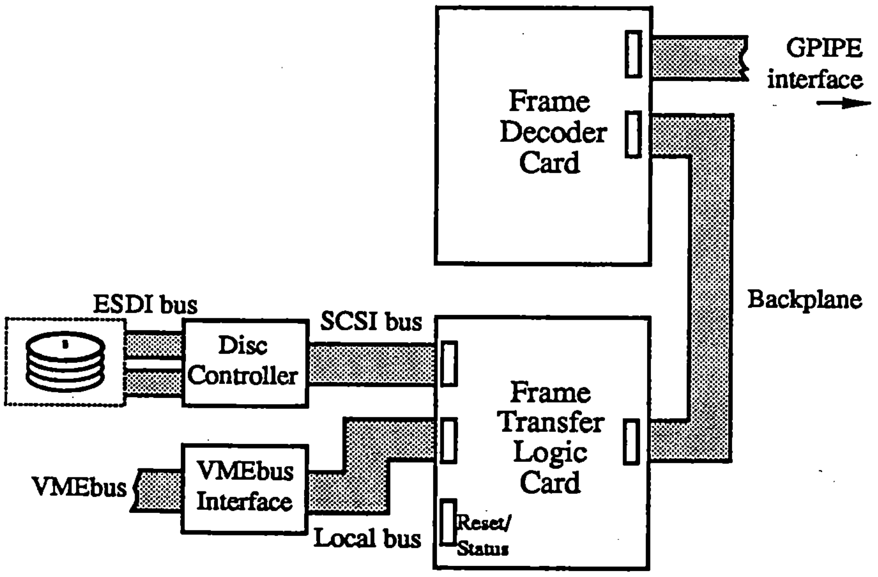


Figure 6.15: Physical organisation of the Animation Server

# Chapter 7

# Control of the Server

## 7.1 Introduction

The previous two chapters considered the implementation and lower level sequencing of the main Server logic. In this chapter the operation of this hardware at the frame level and the synchronisation of the replay pipeline as a whole is examined. Recall from Section 4.3.2 (page 63) that the timing and control of the replay pipeline is the task of the Server Controller (Figure 7.1). The replay path is configured through the FTL in the manner discussed in Chapter 5. Operating at the frame level the Controller can take a longer term view of sequence replay (it is not concerned with the details of the processing of individual frames which are handled by the lower level replay path). This allows it to carry out forward sequencing planning and to make intelligent scheduling decisions which optimise the replay performance of the Server and eliminate (or at least minimise) disruption to the sequence in the event of errors. This operation is facilitated by the generality and power of the M68020-based system upon which the Server Controller is implemented.

## 7.2 Software Organisation

In the experimental version of the system the 68020-based processor is run under the Tripos operating system [Knig82]. The Server control software running on this system comprises a handler task together with three device drivers which manage the controller's external interfaces, as illustrated in Figure 7.2. Two of the drivers manage access to the main Server logic (via the VMEbus) whilst the third provides access to GBus, allowing communication with both Garland and the Rainbow II host. Communication between the handler task and its device drivers is achieved in the standard way for Tripos via the operating system's packet mechanism ([Knig82]).

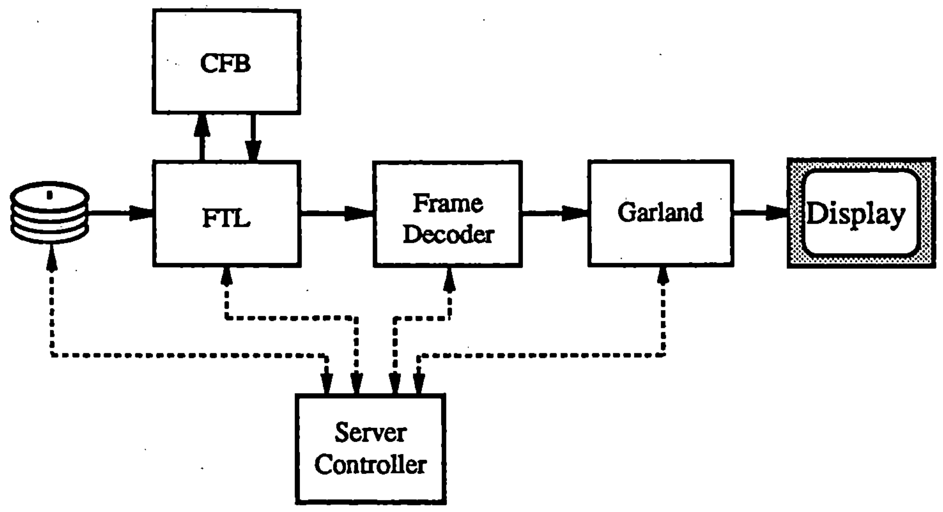


Figure 7.1: The replay pipeline

### 7.2.1 Handler Task

The handler task is responsible for the translation of high level primitives received from the Rainbow II host into the corresponding sequence of low level actions required for the control and synchronisation of the disc, main Server logic and Garland. Any delays or errors occurring within the replay pipeline are handled automatically by this task in such a way that the overall execution of a host-requested operation continues unaffected; the aim of the handler task is to ensure that the complexities of handling sequence replay are transparent from higher levels.

The handler task controls the replay operation using information derived from three sources. Firstly, there is feedback from the main Server Logic in the form of events (Section 4.3.2) which notify the task of completing operations and/or error conditions. A list of normal events is shown in Table 7.1 and a list of reportable errors in Table 7.2. Secondly, information about individual frames is obtained from the Frame Control Buffer after receipt of each "frame_transfer_complete" event. Thirdly, global control information (e.g., the default replay rate) is obtained from the Controller specific portion of the sequence header (Figure 4.15). From this information the handler builds up a model of the current Server status which is used to monitor the progress of replay and as the basis of all sequencing and error recovery decisions (see later). Amongst the details included in the model are: the identifiers of the frame being transferred between the disc and CFB, the frame being decoded and the frame being displayed; the replay time for the current frame; the number of frames remaining until the end of the current (sub-)sequence; and a table describing the current contents of the CFB—the number of frames buffered, their sizes and locations and total buffer usage (to within one compressed frame).

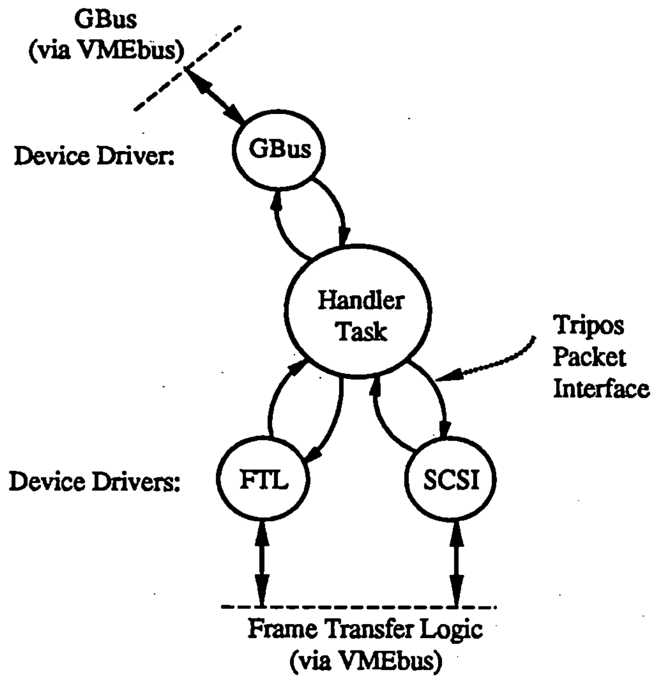


Figure 7.2: Organisation of Server Controller software

| Event | Meaning |
|---|---|
| frame_transfer_complete | Another compressed frame has been transferred to CFB |
| frame_decode_complete | Decoding of another frame has been completed |
| frame_display_complete | End of a video refresh period (as reported by Garland) |
| host_command | New command has been received from Rainbow II host |
| disc | Disc has reported command complete or error |
| FTL_command_complete | Externally initiated FTL microcode sequence has completed |

Table 7.1: Normal events reported to the handler task

| Event | Meaning |
|---|---|
| segment_framing_error | End of compressed segment data does not agree with end of frame decoding |
| DH_overflow | Controller attempted to write Data_Hold_In when full |
| DH_underflow | Controller attempted to read Data_Hold_Out when empty |
| FDecq_empty | Frame Decoder attempted to read empty Frame Decoder queue |
| FTL_error | Internal FTL microcontroller error |
| FDec_error | Internal FDec microcontroller error |

Table 7.2: Errors reported by the main Server logic

## 7.3 The Role of Garland During Replay

The role of Garland during frame replay was outlined in Chapter 4. To summarise, frame replay requires the following operations: (1) the assembly of segments into a single composite frame as they are received from the Server; (2) the implementation of temporal differencing; (3) the buffering and display of completed frames; and (4) the synchronisation of the replay pipeline with video update. These interrelated requirements map directly and naturally onto the functionality provided by Garland; indeed, it is for exactly this reason that these operations were delegated to this system at an early stage in the design.

For dynamic graphics it is normal for the display process to be *double buffered*, meaning that the image is refreshed to screen from one buffer whilst the next frame is being written to a second buffer. At the end of each frame period, the buffers exchange roles. However, such an arrangement presents a challenge for Garland when any form of temporal differencing is required (i.e., the temporal differencing mode itself or the use of techniques such as conditional frame replenishment); the buffer being updated by the Animation Server must contain the image of frame $t-1$ (the last frame it sent) for differencing to work correctly, and not the frame before that (frame $t-2$) as happens in a standard double buffering scheme.

Double-buffered temporal differencing *can* be implemented within Garland, but only with the use all three Image Planes (see Section 4.2.2)[1]. One arrangement which achieves the desired operation is illustrated in Figure 7.3. Image Planes IP1 and IP2 form the double buffers seen by the Animation Server[2], whilst Image Plane IP3 acts as the display buffer from which the screen is refreshed. IP3 always contains an up to date copy of the image. Both IP1 and IP2 alternately execute a buffer update task and a video refresh task, whilst IP3 executes only a single (video refresh) task. During even frame periods the next frame is written by the Server into IP1 whilst the screen is refreshed via IP2/3. During odd frame periods the buffers reverse roles and IP2 is updated from the Server whilst the screen is refreshed via IP1/3. The operation of all tasks is synchronised to the video vertical sync signal.

The update task operates by reading new frame information from the Server in segment order and writing it directly to the appropriate region of the Image Plane. As the Image Plane is always blank at the start of the cycle (see below), the effect of this is to leave an image of the required frame updates in the plane.

The operation of the IP1/IP2 refresh task is synchronised to the operation of the refresh task on IP3, which in turn is synchronised to the operation of the video processor. During video refresh the two Image Planes are read in scanline order, in synchrony with the sweep of the electron beam in the display. Updates in IP1/IP2

[1]Note that whilst three Image Planes are required for a full implementation of the Server as described in this dissertation, working versions of the system can be achieved with two or even a single Image Plane(s).

[2]Note that even though with this arrangement the Animation Server is required to drive *two* GPIPE links, a single GPIPE interface can be used for both since access to these pipes is mutually exclusive. A single one-bit flag is all that is required to identify the active pipe.

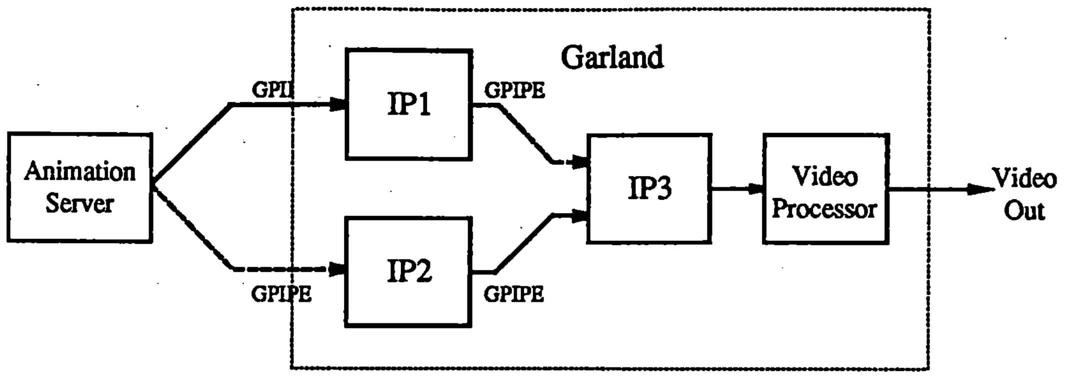


Figure 7.3: Configuration of Garland for animation replay

are merged with the image in IP3 to give the new frame which is simultaneously displayed and written back to IP3. The refresh task on IP1/IP2 uses read-modify-write (RMW) memory cycles to read the update image; these allow it to recover the frame information and simultaneously write back null values, thus leaving a blank buffer ready for the next update cycle. The refresh task on IP3 always writes both to the image plane and to the screen, using either straight write cycles or RMW cycles (depending upon whether incoming values represent full pixels or temporal differences, respectively). In the case of differences, RMW cycles allow merging of an incoming difference with the corresponding previous pixel value via a differencing function contained within the Image Plane's blending memory (Figure 4.3). The boundaries between differenced and non-differenced regions within the update image can be calculated by the image update task from the transmitted segment origin and size description received from the Server before each segment raster. This information is then transmitted to IP3 prior to the start of video refresh (during the frame blanking period) in order that the refresh task can correctly select the appropriate merging function for each pixel. One possible representation of this information is in the form of a *band structure* along the lines of that used in the Rainbow Display [Styn85]; in this case the processing required at region boundaries is much simpler than for Rainbow, and simply involves a switch between two sequences of microcode within the Plane's Task Execution Unit.

## 7.4 Simple Sequence Replay

The previous two chapters described the control interface presented to the Server Controller by the main Server logic. The use of this interface for the control and synchronisation of the replay pipeline is now illustrated by considering the simplest case—that of forward replay at a pre-determined fixed rate. The control required is outlined in the following code fragment (which essentially defines the "PLAY" primitive of Table 4.1):

```
    /* Initialisation */
```

```
frames_buffered := 0;
frames_remaining := END_FRAME - START_FRAME + 1;
frame_being_transferred := START_FRAME;
frame_being_decoded := START_FRAME;
initialise CFB registers (CRP,CWP,CBU);

/* Start disc transfer */
send SCSI 'READ' command block;
set DMA mode in SCSI interface (Register 13);
/* Enable disc to CFB, FCB or FTC transfers */
set DHI_to_CFB, DHI_to_FCB and DHI_to_FTC flags in FTL_cmd register;

/* Wait until CFB full */
do
{
    wait (frame_transfer_complete);
    read frame header from FCB;
    frames_buffered := frames_buffered+1;
    frame_being_transferred := frame_being_transferred+1;
    update CFB_table with new frame size, position, etc.;
} until (new frame size > CFB space available);

/* Enable CFB to Frame Decoder transfers */
set CFB_to_Fdecq flag in FTL_cmd register;

/* Release Frame Decoder */
set run/idle flag in Fdec_cmd register;

start Garland display process; /* Replay starts */

/* Main loop */
while (frames_remaining>0) do
{
    wait (event);
    case (event) of
    {
        frame_transfer_complete :
                read frame header from FCB;
                frames_buffered := frames_buffered+1;
                frame_being_transferred := frame_being_transferred+1;
                update CFB_table with new frame size, position, etc.;

        frame_decode_complete :
                frames_buffered := frames_buffered-1;
                frame_being_decoded := frame_being_decoded+1;
                delete frame details from CFB_table;

        frame_display_complete :
            frames_remaining := frames_remaining-1;
        }
    }
```

Once the replay pipeline has been primed, the control of sequence playback is handled by the main loop. This consists of a single multi-way conditional branch instruction which processes events whenever they are received from the replay pipeline or the Rainbow II host. Events arrive asynchronously and in a non-deterministic order. Only the code related to three events is shown in this example, but note that these three are sufficient to control the simple replay under discussion here. Further, note that in this example the handler task assumes an entirely passive role, only *monitoring* the progress of sequence replay—none of this code has need to access the Server logic and the lower level replay path is allowed to be self-sequencing and self-regulating.

The above code fragment represents only the framework used for Server control, intended principally to illustrate the form and level which this control takes. In practice the implementation is complicated by the need to consider a range of practical factors. Code must be supplied for each of the events which can be reported to the handler task (Tables 7.1 and 7.2), and for the sake of simplicity the above outline ignores: (1) feedback from the disc (operations completing and/or error conditions); (2) errors reported by the main Server logic; and (3) commands arriving from the Rainbow II host, or contained in-line within the frame format, which may override the programmed replay behaviour. All these factors need to be considered for the more complex sequencing and/or interactive operation of the Server. In general, the handler task must take an active role in sequence replay, intervening periodically to issue commands to the replay pipeline in order to achieve the sequencing specified by the instructions being received from the Rainbow II host.

## 7.5 Practical Server Operation

In the remainder of this chapter the discussion is extended to cover, in broader terms, strategies for the more general control of the Animation Server. For the purposes of explanation, the operation of the Server can be conveniently divided into three categories: (1) "special-play" modes; (2) error recovery; and (3) video editing. In practice there is a significant degree of overlap between these activities and the same general principles can be applied in all three cases. In particular, a common requirement is to be able to rearrange the order in which frames are replayed. The mechanism used for achieving this will be described first and then each of the three cases considered in turn.

### 7.5.1 Re-sequencing

The term *re-sequencing* is adopted in this dissertation to refer to the replaying of a frame sequence in an order other than that in which it was originally recorded. A re-sequencing operation can be classified as being either *scheduled* or *unscheduled*, depending upon whether or not it was known about prior to the start of sequence replay. Scheduled re-sequencing includes programmed special-play operations and video editing, unscheduled re-sequencing includes interactive special-play and error recovery.

All re-sequencing relies upon the ability to disengage the disc temporarily whilst its data heads are moved without affecting the overall progress of frame replay in any way. The individual control provided over the connections within the replay pipeline allows data transfer between the disc and CFB to be blocked whilst the remainder of the pipeline continues without interruption. For scheduled re-sequencing a disc transfer is divided into its component sub-sequences and the disc re-programmed after each sub-transfer completes. For unscheduled re-sequencing the current disc operation can be aborted and the disc re-programmed with the new transfer. In either case, re-sequencing introduces seek and/or rotational latencies during which time no disc data can be transferred. This loss is absorbed by the CFB (recall from Section 5.3 that this is one of this buffer's central functions) and the supply of compressed frame data to the Frame Decoder is unaffected. Consequently, the Frame Decoder and Garland are never aware of any re-sequencing which has taken place.

Responsibility for ensuring continuity of replay falls to the handler task. This task does forward planning, using its model of the replay pipeline's status, to ensure that any re-sequencing operation will not result in disruption to the replayed sequence. Some time before the disc operation is due (e.g., 10 frame times) the handler task considers the cost of the operation in relation to the number of frames buffered in the CFB. Cost can be expressed in terms of the number of frame times for which disc data will be lost and can be calculated from knowledge of the seek distance involved and the known performance of the drive (see Chapter 8). Ordinarily the cost of this data loss will be met by the frames buffered and no further action is required. It is normally arranged for the data rate into the CFB to slightly exceed the data rate out so that the CFB has a natural tendency to fill up and counteract the loss caused by any re-sequencing operations. In practice re-sequencing is a relatively rare event and the use of the CFB allows its cost to be spread over several tens or hundreds of frame times. Consequently, the required disparity between the two CFB data rates is normally small (e.g., perhaps 0.5% or less—see Chapter 8). If for any reason sufficient frames are not held within the CFB a number of strategies can be adopted. One possibility, for example, is to repeat the display of frames in the run up to re-sequencing in order to make up the CFB shortfall; each repeated frame allows (on average) one extra compressed frame to be transferred to the CFB.

A second difficulty associated with re-sequencing relates to the fact that frames are not necessarily independent entities: a frame may be dependent upon its natural predecessor[3]. Thus, the frame replay order is important. Frames interdependence operates on two levels:

[3]That is, the predecessor determined at the time of coding

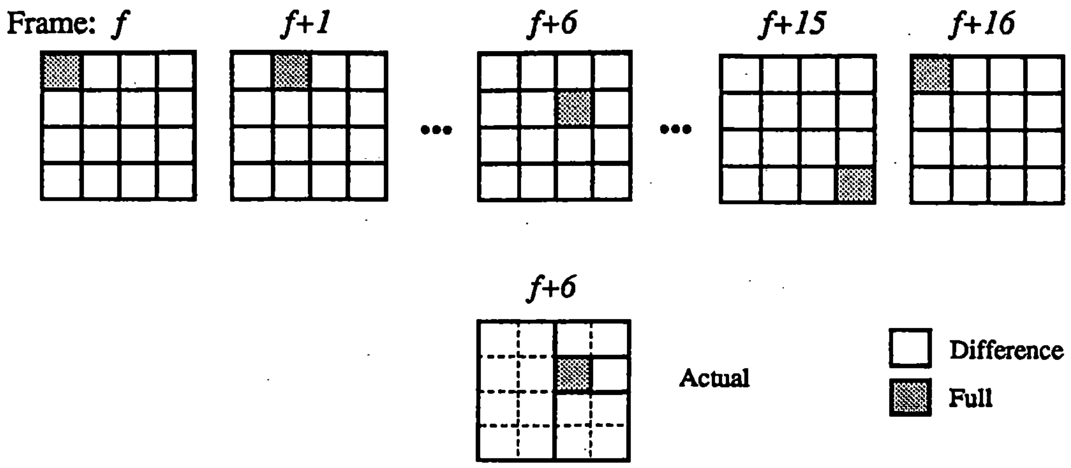


Figure 7.4: Use of a roving full segment for continuous sequence correction.

1. The adoption of frame differencing as the primary frame reconstruction mode means that during replay each frame is derived from its predecessor. Consequently, the first frame after a re-sequencing operation, and all subsequent frames, will be incorrectly reconstructed.

   One solution to this problem is to insert correction frames at regular intervals throughout the sequence. A correction frame is one which has no dependence upon any other frame (that is, one which makes no use of temporal differencing). Then, following a re-sequencing operation the incorrect update of frames continues only up until the next correction frame is reached. Taking this idea a stage further, a more elegant approach is one based upon continuous correction of the sequence using a "roving" full pixel segment[4]. With this arrangement part of each frame is transmitted in full. In Figure 7.4 a tiling of 16 segments is used to illustrate the idea. (Note that in practice adjacent segments can be merged and the implementation of such a scheme only requires between 3 and 5 actual segments, depending upon the position of the full segment—see the example for frame "$f + 6$" in the figure.) Following any disruption, the sequence automatically corrects itself within a fixed number of frame times (e.g. 16 in this case—approximately 2/3 s).

2. The use of *pre-fetching* within the runlength distribution logic introduces a dependency between adjacent pairs of segments within a frame. Pre-fetching is used in order to optimise the performance of the Server. On completion of the transfer of a segment's runlength data the input queues within the Runlength Decoder Units (see Figure 6.4) commence transfer of the following segment by continuing to request coded data. This is a form of pipelining in which the use of spare distribution cycles at the end of a segment period allow the decoding of the following segment to commence immediately

[4]Full pixels segments are used in this example but again any mode other than temporal differencing can be used.

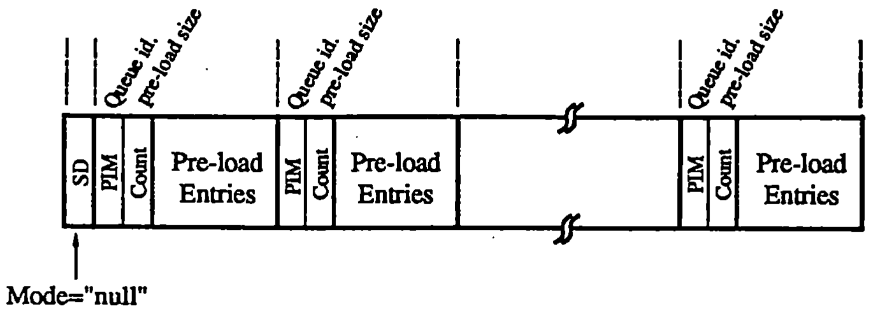


Figure 7.5: Format of null-mode segment

(that is, the plane decoders do not have to wait whilst their input queues build up to a working level). However, this arrangement requires that part of each segment's coded data be actually coded within its predecessor's segment body. This presents no problem for the segments contained *within* a frame as re-sequencing operates at the frame level (and a frame represents an indivisible entity at this level). Unfortunately, however, it also introduces an inter-frame dependency, with the first segment of the next frame being dependent upon the last segment of its natural predecessor. If the decoding of a frame is not preceded by the decoding of its natural predecessor then the runlength decoder queues will not have been left in the state required and misinterpretation of the coded data will result (a segment framing error will be reported). This only represents a problem at re-sequencing points. The solution is the introduction of a new type of segment which contains just the prefetched data for a frame and can be prepended to that frame whenever it is not preceded by its natural predecessor. The format of this "null mode" segment is illustrated in Figure 7.5[5]. The prepending of a null-mode segment is performed transparently within the CFB by the handler task as part of the re-sequencing processing and the Frame Decoder is never aware of when this has been performed. The null-mode segments for a sequence can either be stored in a separate area of disc or, more conveniently, contained within the sequence header and loaded into the Server Controller prior to replay. Segments can be calculated for each frame or limited to a number of key frames (to which re-sequencing is restricted).

It should be noted that the above discussion relates principally to unscheduled re-sequencing; different strategies can be employed when all the re-sequencing points are known in advance. In particular, action can be taken at sequence assembly time to ensure that frame update occurs properly across all re-sequencing points. That is, before replay commences the coder can ensure that the sequence is *logically* contiguous even if it isn't *physically* contiguous.

[5]The name "null mode" derives from the fact that this is effectively a new type of frame reconstruction mode in which no pixels are decoded but the runlength distribution logic is allowed to operate freely.

## 7.6 Special-Play Operation

The term "special-play" is adopted here to refer collectively to the variations on the straightforward frame replay (as outlined in Section 7.4) which are supported by the Animation Server architecture. Special-play operations can be conveniently divided into two groups. Still-frame, single-stepped and "slow-motion" (i.e., reduced replay rate) operation is easily and directly achieved via control of the frame buffer update tasks running in Garland. Preventing the buffer switch at the end of a video refresh cycle causes the current frame to be redisplayed and update of the other (free) frame buffer with the next frame to be halted. Frame replay may then remain halted for one or more frame times. With frame replay blocked at Garland the remainder of the replay pipeline automatically adjusts accordingly, with frame decoding, CFB transfers and ultimately disc transfer all halting. Recall that the replay pipeline was designed to be demand driven and the flow control mechanisms which achieve this operation have already been discussed in Chapters 5 and 6.

The second group of special-play operations employ re-sequencing to modify the order in which the sequence is replayed. Examples of such operations include:

- Replaying a sequence starting from an arbitrary point. It is undesirable for the adoption of temporal differencing to always force frame replay to start from the head of the sequence. This is particularly true of long sequences, where the delay before the required start point is reached may be of the order of minutes. The use a randomly accessible frame storage device together with the re-sequencing mechanisms discussed above allow replay to start directly from anywhere in the sequence, and for the correct frame content to be established automatically within a fixed number of frames times (e.g. 16 in the previous example) from this point.
- Frame skipping. This allows, for example, fast forwarding to a point further on in the sequence. Again, the displayed sequence will recover automatically within a short time after the re-sequencing.
- Cycling back through sub-sequences to repeat sections of animation.

In all the above examples special-play operation can either be specified interactively or via programmed control.

## 7.7 Error Recovery Strategies

Various types of error can potentially arise during frame replay, of which only disc errors (as reported by a disc event—Table 7.1) are expected under normal circumstances. Even so, provision must be made for handling each of the possible error conditions (Table 7.2). This is particularly true given the lack of redundancy within the compressed frame format since any misinterpretation of this representation results in a catastrophic failure of the decoding process. For the purposes of error detection/correction the frame replay path (Figure 7.1) is divided into three regions as follows:

**Disc to CFB** This covers all types of disc fault (read, seek errors, etc) and SCSI bus transfer errors. Errors are detected by the disc controller, SCSI interface or by a frame header checksum. Disc errors can be classified as being either recoverable or unrecoverable, with all errors reported to the handler task by definition of the latter type. In common with all SCSI devices, the disc controller employed in the Animation Server is intelligent and capable of handling the majority of disc faults transparently. Following a read error the controller will automatically force a pre-defined number of re-reads before attempting to recover the bad block using the sector's error correcting code. If all this fails an unrecoverable read error is reported. Known bad blocks can be relocated to reserved blocks (assigned during formatting) either on the same track or in a relocation track. Relocation is handled as part of SCSI's logical to physical block mapping process and relocated blocks are fetched automatically during the course of a read command[6]. Similarly, seek errors are handled by an automatic disc recalibration and attempted re-seek. Parity detected transmission errors during data transfer are reported by the Server's SCSI bus interface and handled by the device driver using the SCSI message system. All recoverable errors thus result in a loss of disc data for a period of time. Whilst this period may be significant (i.e., several frame times), the loss is absorbed by the CFB, without requiring intervention from the Server Controller, and sequence replay is not affected in any way.

Following an unrecoverable error the handler task must intervene to resume disc data transfer from some point after the bad block. A typical approach would be to seek to the start of the next compressed frame whilst reseting the CFB's writing point back to the end of the previous (i.e., last successfully) transferred frame. The effect of this operation is to splice out the entire frame within which the error is contained. When this splice point eventually reaches the Frame Decoder the resultant sequence disruption is handled automatically by the re-sequencing techniques discussed in Section 7.5.1, and the sequence rapidly recovers from the discontinuity.

**CFB to Frame Decoder** Errors arising between the CFB and Frame Decoder are detected by the receipt of a segment framing error from the Frame Decoder. Such errors arise either because of an original mis-alignment at the start of segment decoding or because of corruption to one or more bits within the coded segment data stream. A typical recovery strategy would be to flush the Server's internal queues and to then advance the CFB's reading position to the start of the next (pre-fetched) frame. Again, the Server will rapidly recover from the resulting discontinuity.

**Frame Decoder to Screen** Errors occurring after data has left the Frame Decoder, such as pixel transmission errors for example, are not directly detected. However, the use of a roving full segment ensures that the sequence is continually correcting itself and the effects of any errors are thus short-lived. No intervention

[6]Note, therefore, that frame data will never knowingly be recorded in bad data blocks

is required from the handler task. Also, note that the decoded data stream is inherently more error tolerant than the coded stream in that errors only affect individual pixels.

## 7.8 Video Editing

One of the stated goals of the Animation Server is to provide support for the interactive development of animated sequences (Section 2.5). The support provided for interactive control over the frame replay process has already been discussed in this chapter. A further requirement is to provide assistance for the editing of sequences already resident on the disc; that is, to give the user the ability to rearrange frame orderings, add and delete sub-sequences, and so on. This represents a challenge for two (interrelated) reasons: (1) for efficient replay frames must be laid down contiguously; and (2) the sizes of recorded frames vary due to the adoption of image compression. A simplistic approach which meets both these requirements is to re-write the whole sequence for each edit made. This can be done either *in situ* or by copying the sequence to a new area of disc. In the latter case the disc might be divided up into fixed size time slots of, say, 30s duration and a sequence copied into the next free slot. The Server hardware provides assistance for overwriting sequences *in situ* by allowing the CFB to be configured as a working store during disc re-organisation. Data can be transferred to or from this store at high speed through the use of hardware controlled DMA transfers (see Chapter 5). The CFB has the capacity to store approximately 21 full disc tracks. Even so, such an approach is fundamentally inefficient. For example modifying a single frame at the start of a 3 minute sequence would require at least 6 minutes to effect (3 minutes to read the sequence and a further 3 minutes to re-write it), even ignoring other processing overheads. Clearly such an arrangement would demand that a sequence of edits be batched up and executed in one pass; such a constraint is undesirable, however, as it restricts the freedom of the animator during the editing process.

An alternative and more elegant solution to the video editing problem exploits the ability of the Server to assemble frame sequences dynamically during replay. This allows frames to be skipped or new sub-sequences to be inserted into an existing sequence. With this approach editing represents just another example of frame re-sequencing for which the techniques already discussed can be employed. The use of the CFB gives the ability to skip over splice points in an edited sequence. Figure 7.6 illustrates the disc organisation for the two basic cases of deletion and insertion, and for modification which is a combination of these two. All editing operations can be defined in terms of these primitives. With temporally differenced sequences the last frame before the splice point in the original sequence must be re-calculated in order that its new successor (i.e., the first frame after the splice) is properly derived. Note, however, that this is the *only* frame for which this must be done and the remainder of the original sequence needs no further processing; this is perhaps a non-intuitive point. At a topmost level the user is unaware of this organisation and a sequence can be edited rapidly and easily and then

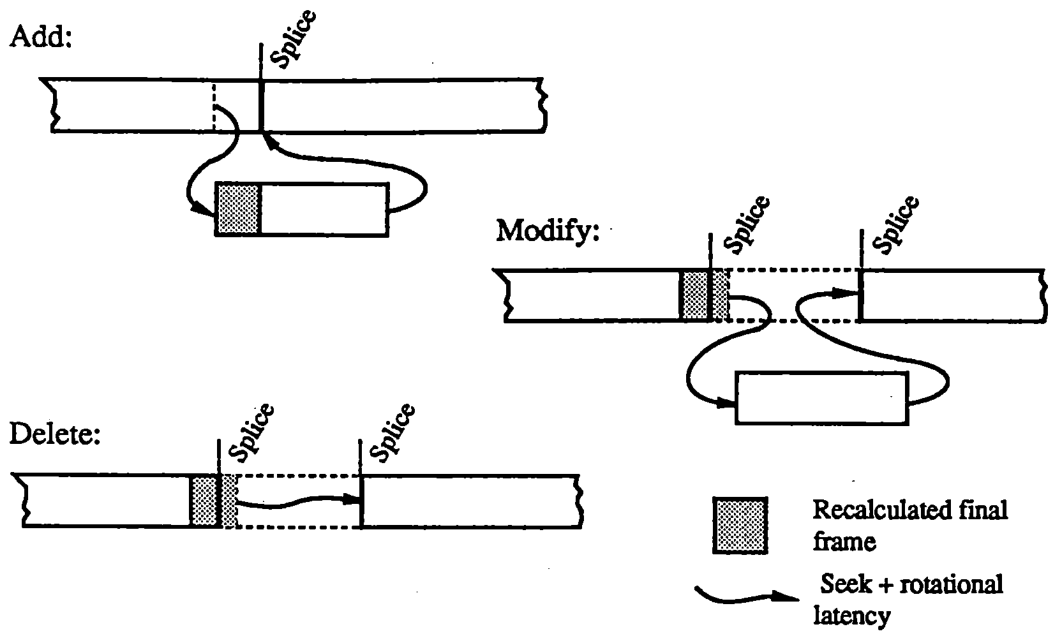


Figure 7.6: Disc organisation for the three basic editing operations

replayed immediately to observe the results. The management of this process is handled by high-level editing software which hides the underlying details and complexities from the user. This software aims to optimise both edit time and replay performance by taking into account the known performance of the Server. Strategies include the use of clever disc placement algorithms, the merging of adjacent edits into a single edit, and the periodical re-writing of the sequence to avoid fragmentation.

# Chapter 8

# Performance Issues

## 8.1 Introduction

In this chapter the performance of the Animation Server is considered in more detail. The chapter divides into two parts. In the first part a simple, non-rigorous, analysis of the Server's replay performance is presented, taking into account factors such as the effects of disc organisation, the performance of the Frame Decoder and runlength distribution logic, and the effects of errors and re-sequencing operations. This analysis allows the requirements for frame sequence replay to be derived. In particular, this includes determination of a lower bound for the frame compression ratio—that is, the minimum level of frame compression which must be achieved to ensure proper replay operation. In the second part the actual compression performance achieved by the coding scheme employed (Chapter 4) is considered for a range of test stills and frame sequences.

## 8.2 Replay Performance

In the following analysis attention is concentrated upon that portion of the frame replay path contained within the Animation Server (that is, the path formed by the disc, FTL/CFB and Frame Decoder—see Figures 4.4 and 4.5).

### 8.2.1 Disc Performance

For frame replay the performance of the disc drive is of paramount importance. In particular three parameters—its transfer rate, capacity and track-to-track seek time—have a major bearing upon the overall performance of frame replay. The Maxtor drive employed within the Animation Server was selected on the grounds that its performance in these key three areas (in particular) significantly exceeded that of all other 5¼ drives available at that time. A summary of the relevant disc parameters, drawn from [Max86], is given in Table 8.1.

For frame replay the principal consideration is that of disc data transfer rate. The XT-8380E drive has a quoted transfer rate of 15Mb/s. However, the effective

| Parameter | | Value | Units |
|---|---|---|---|
| Capacity | per drive | 408.00 | MBytes |
| (unformatted) | per surface | 51.00 | MBytes |
| | per track | 31,250 (min) | Mbytes |
| Transfer rate | | 15.00 | Mb/s |
| Access | average | < 16.00 | ms |
| Time[a] | track-to-track | 2.00 | ms |
| | maximum | 35.00 | ms |
| Rotational speed | | 3600 | rpm |
| Average latency | | 8.33 | ms |
| Cylinders | | 1,632 | — |
| Tracks | | 13,056 | — |
| Data heads | | 8 | — |
| Error | soft read | 10 per $10^{11}$ bits read | — |
| Rates | hard read[b] | 10 per $10^{13}$ bits read | — |
| | seek | 10 per $10^7$ seeks | — |

[a]maximum, includes settling
[b]Not recoverable within 16 retries

Table 8.1: Summary of relevant Maxtor XT-8380E disc parameters

*data* rate is lower than this because a number of practical factors must be be taken into account. In particular:

- *Formatting Overheads:* The quoted transfer rate refers to the 15MHz READ clock of the drive's native ESDI interface and represents the *raw* transfer rate of the drive. That is, it refers to the rate at which recorded bits are directly recovered by the selected data head. However, this bitstream also contains formatting information in addition to the required data bits, which is used by the disc controller for the purposes of timing and synchronisation, disc addressing, and error detection and correction. As only the data blocks are transferred to the SCSI bus by the controller, the effective transfer rate, the disc's *data* rate, is lower than this raw transfer rate. Calculation of this rate requires determination of the disc's usable data area. The raw physical sector size of the disc is determined by [WDD86]:

$$\text{secsize} = \text{ISG} + 2\times\ \text{PLO} + \text{CHKS} + \text{SYNC} + \text{PAD} + \text{OVHD} + \text{BLKSIZ}$$

where ISG = intersector gap (26 bytes), PLO = Phase-lock oscillator sync field (24 bytes), CHKS = checksums (8 bytes), SYNC = synchronisation and ID bytes (7 bytes), PAD = pad fields (8 bytes), OVHD = miscellaneous overheads (8 bytes) and BLKSIZ = data block size. For a data block size of 1024 bytes this gives a sector size of:

$$\text{secsize} = 105 + 1024 = 1129 \text{ bytes}$$

Thus, formatting represents an overhead of $(105/1129) \times 100 = 9.3\%$. This gives a formatted track capacity of:

$$\begin{aligned} \text{sectors per track} &= \frac{\text{bytes per track}}{\text{secsize}} \\ &= \frac{31250}{1129} \\ &= 27.68\,\text{sectors} \end{aligned}$$

The drive can therefore support 27 full sectors per track. One track can be read per disc revolution (16.67ms), and so the effective data rate is given by:

$$\begin{aligned} \text{disc data rate} &= \frac{1024 \times 27}{16.67 \times 10^{-3}} \\ &= 1.66\,\text{MB/s} \\ &= 13.27\,\text{Mb/s} \end{aligned}$$

- *Head and Cylinder Switching:* Compressed frames are generally laid contiguously on disc to avoid waste of transfer bandwidth due to sequence discontinuities. Discontinuities caused by head and cylinder switching, however, cannot be avoided. A head switch is required at the end of each disc revolution in order to advance to the next track within the cylinder. Similarly, a single track seek is required after every 8 revolutions in order to advance to the next cylinder. This potentially introduces a loss of data for one revolution (16.67ms) on *each* revolution, an unacceptably high overhead. However, this situation is avoided by utilizing the disc controller's support for track and cylinder *skewing*. Skewing is accomplished via the mapping between logical and physical sectors[1] which allows the *logical* start of a track to be displaced by some number of physical sectors from the end of the previous track. The size of skew is arranged such that after a head and/or cylinder switch has occurred and any data head settling has taken place the *next* sector to pass the data head represents the logical start of the new track (i.e., it is the next consecutive logical block following the last logical block in the previous track).

  With a sector size of 1Kbyte the minimum skew of 1 sector is sufficient to accommodate a head switch [WDD86]. A cylinder switch takes 2ms for the

[1]Physical sectors represent actual physical disc sectors. The idea of logical sectors is encapsulated within the SCSI protocol as a device independent way of addressing block structured devices. Logical sectors are always sequentially numbered in the range 0–(NUMBLOCKS-1), with the actual the mapping between logical and physical sectors determined at the time of device formatting (and handled transparently by the SCSI controller).

required track-to-track seek, which represents $2/16.67 \approx 1/8$ of one revolution or $1/8 \times 27 = 3.24$ blocks. Thus, a skew of 4 sectors is sufficient to accommodate a cylinder switch. Then, the number of blocks which can be transferred in 8 disc revolutions (1 cylinder) is given by:

$$\begin{aligned}\text{Number of blocks} &= (27 - \text{trackskew}) \times 8 - \text{cylinder skew}\\ &= (27 - 1) \times 8 - 4\\ &= 204\,\text{blocks}\\ &= 208.9\,\text{KB}\end{aligned}$$

This takes place in $8 \times 16.67 = 133.33$ms, giving an effective data rate of:

$$\begin{aligned}\text{Effective disc data rate} &= \frac{208.9 \times 10^3}{133.33 \times 10^{-3}}\\ &= 1.57\,\text{MB/s}\\ &= 12.53\,\text{Mb/s} \qquad (8.1)\end{aligned}$$

- *SCSI bus synchronisation:* The disc controller supports asynchronous data transfer across its SCSI interface at a rate of up to 1.5MB/s (12Mb/s). Note that this is actually slightly less than the effective disc data rate (Equation 8.1). Further, for each bus transfer there is an additional cost introduced at the FTL's SCSI interface (Figure 5.2) as asynchronously received transfer requests must be synchronised to the Server's internal clock. This introduces a SCSI REQ-to-ACK delay of, on average, half of one machine cycle. The *actual* disc data rate can therefore be calculated as follows:

$$\begin{aligned}\text{Actual SCSI cycle time} &= \text{Minimum cycle time} + \text{synchronisation delay}\\ &= \frac{1}{1.5 \times 10^6} + \frac{152.59 \times 10^{-9}}{2}\\ &= 742.92 \times 10^{-9}\,\text{s}\end{aligned}$$

Thus, the actual available disc transfer rate is given by:

$$\begin{aligned}\text{Actual SCSI rate} &= \frac{1}{742.92 \times 10^{-9}}\\ &= 1.35\,\text{MB/s}\\ &= 10.77\,\text{Mb/s}\end{aligned}$$

(Compare this with the raw transfer rate of 15Mb/s.) In one frame time (1/25s) this allows the transfer of:

$$\begin{aligned}\text{transfer per frame time} &= \frac{1.35 \times 10^6}{25} \\ &= 53.84\,\text{KB} \qquad (8.2)\end{aligned}$$

Equation 8.2 gives the amount of data which can be transferred to the FTL from the disc within a single frame time. This, therefore, defines the *average* size of compressed frame required for the correct operation of the Server at 25fps.

## 8.2.2 Frame Decoder Performance

Analysis of the Frame Decoder's performance is straightforward. The decoder cycle time (152.59ns) was selected to allow the decoding of 262144 pixels within 1/25s; that is, $1/(262144 \times 25) = 152.59 \times 10^{-9}$s. One pixel can be decoded per cycle. This corresponds to a (notional) frame size of $512 \times 512 \times 8$. However, the total number of pixels decodable within a frame time is reduced by segmentation overheads, where the general cost of a segment switch is given by:

$$\text{segment overhead} = 8 + \left(1 + \frac{\text{ptable\_size}}{2}\right) \times \text{ntables}$$

For all segments there is a fixed overhead of 8 cycles imposed by the loading of the segment header registers (SD,X,Y,W,H,PIM) and the compressed segment size (CSS) (see Figure 4.15). The loading of each prediction table (and its descriptor) imposes an additional load of 1+ptable_size/2 cycles (Section 6.1.2). Between 0 and 8 prediction tables can be loaded for each segment. Thus, the total segmentation cost per frame can vary from frame to frame, depending upon the number of segments and the number of prediction tables to be loaded. For the purposes of illustration, it will be assumed that each frame employs 16 segments and that a single full prediction function (i.e., 256 entries) is employed for the duration of the whole frame (a not unlikely situation). The prediction table entries defining this function are loaded once, in the first segment of the frame. The total segmentation cost for each frame is then given by:

$$\begin{aligned}\text{segmentation cost/frame} &= 8 + (1 + 256/2) \times 8 + (8 + 0) \times 15 \\ &= 1160\,\text{cycles}\end{aligned}$$

During segment initialisation the Plane Decoders must be idled and consequently no pixels can be decoded. Thus, frame segmentation overheads introduce a loss of 1160 pixels from the theoretical maximum decodable size (approximately

0.4% of the total frame area). Segmentation also reduces the amount of coded segment data which can be transferred within a compressed frame by 1160 bytes (plus a further 16 bytes are lost to the frame header). The ratio of uncompressed to compressed frame size (Equation 8.2) gives the required compression ratio:

$$\begin{aligned}\text{Compression ratio} &= \frac{\text{Decoded frame size (bytes)}}{\text{coded frame size (bytes)}} \\ &= \frac{262144 - 1160}{53.84 \times 10^3 - 1160 - 16} \\ &= \frac{261104}{52624} \\ &= 4.96\end{aligned}$$

The *average* compression ratio required for correct server operation is therefore approximately 4.96:1. With a total of 13,056 tracks, each with 27 1K byte sectors, the drive has a total formatted capacity of $13056 \times 1024 \times 27 = 360.97 \times 10^6$ bytes. This gives a total frame capacity of:

$$\begin{aligned}\text{Total frame capacity of drive} &= \frac{\text{formatted capacity}}{\text{average compressed frame size}} \\ &= \frac{360.97 \times 10^6}{53.84 \times 10^3} \\ &= 6704.53\ \text{frames}\end{aligned}$$

and a total replay time of $6704/25 = 268$s or 4½ minutes.

Note that the above analysis derives the lower limit of the Server's replay performance (that is, for sequence replay at full frame rate with the minimum permissible level of frame compression). At 12½ fps, for example, the total replay time increases to 9 minutes.

### 8.2.3 Re-sequencing Performance

Re-sequencing, and its applications (error recovery, special-play, etc), will be discussed only briefly, and in general terms. The average cost of a re-sequencing operation is that of an average seek followed by a rotational latency:

$$\begin{aligned}\text{re-sequence cost} &= \text{average seek time} + \text{rotational latency} \\ &= 16 \times 10^{-3} + 8.33 \times 10^{-3} \\ &= 24.33\ \text{ms}\end{aligned}$$

which corresponds to approximately 0.6 of a frame time (at 25fps). With the use of the minimum degree of compression (4.96:1) the 0.5MB Compressed-frame Buffer has an average (worst case) frame capacity of:

$$\begin{aligned}\text{CFB capacity} &= \frac{\text{CFB capacity (bytes)}}{\text{average compressed frame size}} \\ &= \frac{524288}{53.89 \times 10^3} \\ &= 9.73 \text{ compressed frames}\end{aligned}$$

Prior to the start of sequence replay the CFB is preloaded to capacity (see Chapter 7) and so disc transfers initially lead frame decoding by approximately 10 frame times. This lag means that during the course of replay the CFB has the capacity to absorb $9.73/0.6 \approx 16$ re-sequencing operations without any disruption to sequence replay. For many sequences this is sufficient; re-sequencing will often be used as a fairly coarse-grained mechanism (such as when skipping over the occasional splice point in an edited sequence, for example). When it isn't, the cost of a re-sequencing operation can be covered by ensuring that the CFB's input rate slightly exceeds its output rate. In practice, a very small difference between the two rates is sufficient to meet most ordinary re-sequencing requirements. For example, a difference of just 0.5% (achieved with a 0.5% improvement over the minimum coded image size, perhaps) means that one additional compressed frame is transferred every 200 frame times (i.e., 8 seconds) and that the cost of a typical re-sequencing operation is covered within approximately 100 frame times (4 seconds). Note that whilst the cost of re-sequencing is spread over perhaps several hundred frame times, this represent only a relatively short time at the user level (a few seconds).

Similar comments apply to error handling. For example, a disc mis-read incurs a one rotational latency—16.67ms or 0.4 frame times—thus giving the CFB the capacity to absorb $9.73/0.4 \approx 24$ mis-reads during the course of sequence replay. In practice, the expected error rate is much less than this: the quoted (soft) read error rate is 10 in $10^{11}$ bits read (1 in $1.25 \times 10^9$ bytes) which corresponds approximately to 1 in 23195 compressed frames, or one error for every 15 minutes of replay.

## 8.3 Preliminary Coding Results

This chapter concludes with the presentation of a range of preliminary compression results obtained for the adopted coding strategy. The generality of the coding scheme means that it is impossible to present results for, or even to test, all combinations of the frame reconstruction modes supported by the segmentation mechanism. Further, the unavailability of the prototype Server hardware for experimentation (see Chapter 9) meant that the *subjective* effect of various compression strategies could not be evaluated; such an evaluation is felt to be vital since considerable improvements over the basic coding performance can be obtained with little subjective loss of quality. It is also difficult, until a degree of practical experience has been accumulated with an operational system, to determine the sort of "typical" sequences which users wish to replay; again, additional compression can often be obtained by optimising a coding strategy to the particular characteristics of an image sequence. (These last two points were discussed in Chapter 3.) Even so, it is felt that the early results presented below do serve to give some indication of the figures which might be expected in practice.

### 8.3.1 The Test Images

A range of different test images, generated using a range of techniques and obtained from a number of different sources[2]), were used during experimentation. Both stills and frame sequences were employed, as follows:

#### Stills

- "Mandrill" ($512 \times 480 \times 8$): A 24-bit digitised colour photograph of the face of a mandrill (a kind of large baboon). This is a classical image within the field of computer graphics. Three versions of the image were used, differentiated by the means used to map the original 24 bits/pixel to the required 8 bits/pixel: (1) A median cut algorithm ("mcut") (see Chapter 3); (2) a direct mapping from 24 to 8 bits ("to8"); and (3) conversion to an 8-bit greyscale ("tobw"). These stills represent a coding challenge both because of the original fine image detail and because of the introduction of *dither* during the mapping process[3].
- "Balloons" ($388 \times 512 \times 8$): A 24-bit digitised colour still of a girl surrounded by balloons. A single mapping technique was used ("mcut"), with similar comments to above applying equally here.
- "Hills" ($512 \times 512 \times 4$): An image of a range of hills, created on a paint system running on the Rainbow Display. The image represents a challenge because of the use of an "airbrush" effect which introduces considerable "noise" into the image.
- "IronBridge" ($512 \times 512 \times 5$): As above, except of a scene of an iron bridge spanning a valley.
- "Phong" ($512 \times 512 \times 8$): An image of two large spheres rendered using a smooth shading technique (Phong shading).

[2]Hence the variation in frame size amongst the set. Note, however, that since it is the compression *ratio* which is of interest, the actual original frame size is of little relevance.

[3]Dither is a technique used to improve the subjective quality of images by reducing the effects of *banding* which occur when too few colours are available for the display of the image. It works, essentially, by introducing psuedo-random noise into the image. The technique was originally invented by Roberts [Robe62].

**Sequences**

Four sequences were used, all 25 frames in length (i.e., 1 second of animation):

- "Phoenix" ($200\times200\times8\times25$): A 2-d animation created by Jeremy Ball, of the Cambridge University Computer Laboratory, which shows an egg evolving into a phoenix over the course of the sequence.
- "Fire" ($200 \times 200 \times 8 \times 25$): A 3-d animation, involving the same phoenix as above, except here its evolution is obscured by the evolution of a ball of fire on top of it. The effect of fire is created using a particle system.
- "Spheres1" ($256 \times 256 \times 8 \times 25$): A 3-d animation created by Eng Lim Goh, also of the Computer Laboratory, and involving a fly past of a ray-traced scene made up of one large and 138 smaller spheres. All spheres are highly reflective which introduces considerable visual complexity into the scene.
- "Spheres2" ($256 \times 256 \times 8 \times 25$): A 3-d animation using the same spheres model as above, except here the eye-point remains stationary as the light source is swept across the scene. The variation in the lighting conditions introduce dramatic changes both in the inter-sphere reflections and in the shadowing in the scene.

The phoenix and fire sequences were developed as part of some research into animation language design using object oriented programming techniques. The spheres sequences were developed during research into the optimisation of ray tracing techniques using parallel arrays of transputers. All four are felt to be representative of "typical" sequences; indeed, both developers had expressed an interest in the use of the Animation Server as a practical means of animating their sequences.

## 8.3.2 Results

Tables 8.2 to 8.5 illustrate the performance of the coding scheme under a range of operational conditions:

Table 8.2 illustrates the compression factors obtained for the set of stills. These results were obtained using an 8 bit prediction pattern (with a 2-bit skew) and 8 runlength streams, as used in the hardware implementation of the scheme.

Table 8.3 gives the results obtained for the full pixel mode applied to the four sequences. Again, an 8 bit prediction template and 8 runlength streams were employed.

Table 8.4 is similar to Table 8.3 except it gives the results obtained for coding temporal differences rather than full pixels.

Finally, Table 8.5 illustrates how the basic compression ratios achieved might be improved by the use of appropriate frame reconstruction modes. The example given show the improvements gained by the use of the interpolation mode, as applied to a range of stills (including one still from each of the four sequences).

| Still | Compression Ratio |
|---|---|
| Mandrill-mcut | 1.35:1 |
| Mandrill-to8 | 1.20:1 |
| Mandrill-tobw | 1.14:1 |
| Balloons | 1.56:1 |
| Hills | 3.98:1 |
| Ironbridge | 5.07:1 |
| Phong | 8.57:1 |

Table 8.2: Compression ratios obtained for test stills

| Sequence | Compression Ratio | | |
|---|---|---|---|
| | Max. | Min | Average (25 frames) |
| Phoenix | 6.69:1 | 6.44:1 | 6.61:1 |
| Fire | 3.38:1 | 3.29:1 | 3.35:1 |
| Spheres1 | 2.64:1 | 2.33:1 | 2.47:1 |
| Spheres2 | 2.70:1 | 2.61:1 | 2.66:1 |

Table 8.3: Results for full pixel mode on the four sequences

| Sequence | Compression Ratio | | |
|---|---|---|---|
| | Max. | Min | Average (24 frames) |
| Phoenix | 71.43:1 | 33.82:1 | 51.83:1 |
| Fire | 4.20:1 | 3.96:1 | 4.03:1 |
| Spheres1 | 2.29:1 | 2.04:1 | 2.15:1 |
| Spheres2 | 4.14:1 | 3.73:1 | 3.93:1 |

Table 8.4: Results for temporal differencing mode on the four sequences

| Still | Compression Ratio | |
|---|---|---|
| | Full | Interpolation |
| Phoenix | 6.64:1 | 13.78:1 |
| Fire | 3.37:1 | 7.02:1 |
| Mandrill | 1.35:1 | 2.66:1 |
| Balloons | 1.56:1 | 2.48:1 |
| Spheres1 | 2.39:1 | 4.40:1 |
| Spheres2 | 2.66:1 | 4.98:1 |
| Hills | 3.98:1 | 7.70:1 |
| Ironbridge | 5.07:1 | 8.96:1 |

Table 8.5: Coding improvements gained by the use of interpolation

# Chapter 9

# Summary and Conclusions

This dissertation has discussed the design and implementation of a stored-frame animation system. Computer animation was classified as either real-time or real-time playback, and stored-frame animation was identified as a subset of the latter. The requirements for such animation were discussed. A number of previous stored-frame animation systems were examined and the advantages and disadvantages of the various individual approaches identified. In the light of these observations the design of the new system was proposed, based upon the use of an unmodified 5¼ inch Winchester disc in conjunction with a hardware implementation of an image decompression scheme.

## 9.1 Work Completed

A full hardware design of the Server architecture, as described within this dissertation, was produced and a prototype system constructed (see Plates 1–5 at the end of this dissertation). For reasons of economy, the initial version of the Frame Decoder board contains only four Plane Decoders, instead of the full eight. This move was prompted by the unexpected complexity of a Plane Decoder (and in particular, a Runlength Decoder Unit), which placed severe demands upon the available board area. It was wished to avoid the complexities of a two board implementation of the Frame Decoder, particularly as a four plane system is entirely sufficient to demonstrate both the operation of a Plane Decoder individually and the multi-plane operation of the Frame Decoder as a whole (with respect to runlength distribution, etc). The problems with Decoder complexity were due largely to the use of MSI/LSI FAST TTL technology together with wirewrap construction techniques[1]. The extensive use of Programmable Logic Devices (PLDs) throughout the design helped to reduce the total chip count, but not to the level desired. Note, however, that the parallel nature of the Frame Decoder design means that any future implementation would benefit greatly from the use of semi- or full-custom logic; a Runlength Decoder Unit and a Predictor Unit (or possibly an entire Plane Decoder) can be implemented on a single chip. This would

[1]This represented the only viable option at the time the prototype was being developed.

dramatically reduce the total chip count for the Frame Decoder implementation.

Testing of the prototype was started on the FTL board, with the interface and lower levels of the routing logic on this board (see Chapter 5) being successfully tested. During this period various test and low level control software (e.g., a SCSI protocol manager) was also developed. However, the system as a whole was not fully commissioned. This was for a number of reasons. Firstly, the overall design process took much longer than originally anticipated, leaving less time than planned for system construction and testing. This in part was due to a certain amount of re-evaluation of ideas as new concepts evolved during the course of design (including, for example, the whole idea of a segmentation mechanism). Also, for a variety of reasons, delays were encountered during the early part of testing (for instance, a failure of the disc drive in combination with the receipt of several incorrect disc controller boards from the supplier led to delays which totalled over five months). Secondly, during this period Tony King, who had been developing Garland, left the Department. At this stage Garland was not fully operational and it became clear that completion of the prototype by others was infeasible, and that certainly a full implementation of Rainbow II would never be realised. To remedy this situation it was initially proposed to build a simple video card for the purposes of getting a limited form of the Animation Server operational. However, this was not a favoured solution as many of the Server's more interesting and unusual features are centered around the use of frame segmentation, and this mechanism was deliberately designed to integrate closely with the operation of Garland. Because of this, it was felt that the investment of a large amount of effort in commissioning a restricted version of the hardware that would not allow these features to be explored any further would not have added significantly to the research element of the project.

In spite of these shortcomings it is felt that sufficient progress has been made overall to allow valid conclusions to be drawn about the successes and failures of the research undertaken.

## 9.2 Re-examination of Research Objectives

In this section the design and implementation of the Animation Server is re-examined in the context of the original research objectives. It is felt that this research has made two principal contributions towards the design of such systems, namely:

1. It investigated the use of *frame segmentation* as a means of dramatically increasing the power and flexibility of an animation system's replay performance. In the Animation Server the segmentation mechanism provides a unified interface to a range of alternative image compression and more general frame reconstruction modes, allowing these to be freely mixed within a single frame.

2. It introduced the idea of a *replay pipeline* (the path from disc to screen) and examined ways in which flexible control and configuration of this pipeline

might be achieved. In particular, in the Animation Server individual sections of this pipeline can be controlled independently of one another. Moreover, rather than attempting to minimise the latency of this pipeline, additional delay is purposely created by the introduction of the large Compressed-Frame Buffer. This approach is unique to the Animation Server and differs radically from the conventional wisdom, as illustrated in all other "directly coupled" replay systems—film, videotape, optical/videodisc, and so on. The principal benefit of such an arrangement is that it supports the idea of *re-sequencing*, in which a frame sequence can be dynamically assembled during replay from a number of constituent sub-sequences. Uses for this mechanism were discussed in Chapter 7.

These points will now be discussed in relation to the original research objectives, as enumerated in Section 2.5.

1. *To investigate the extent to which performance limitations associated with many previous animation systems can be overcome by the development of a custom real-time playback architecture:* A new replay architecture was developed which is capable of the sustained transfer and processing of frame data at a rate greater than that which can be supplied by the disc. Thus, it is the disc which is the limiting factor within this system (c.f. software or bus transfer overheads in many previous systems). Further, the available disc bandwidth is optimised by the use of frame compression and by architectural features such as the Frame Control Buffer (which allows continuous transfer of compressed frame data into the CFB without having to wait for frame headers to be collected by the Server Controller).

2. *To employ an unmodified, non-video-rate storage device on conjunction with a hardware based implementation of a frame compression algorithm:* The contribution made by this research to the field of image compression is twofold:

   (a) It investigated and implemented a new combined predictive/runlength frame coding scheme. This scheme is simple, yet powerful, and maps well to its use in a real-time playback system.

   (b) It developed the idea of frame segmentation as a means of providing a generalised coding framework. With such a mechanism a number of different compression strategies can be freely mixed within a single frame. This allows the implementation of a range of existing and "classic" techniques (examples cited in this dissertation include conditional frame replenishment, exchanging spatial and temporal resolution, and approximating frames after a scene cut), as well as providing the potential for the development of new techniques. Note that in practice much of the research into the development of "new" image coding techniques actually involves the combining of existing techniques in new ways.

3. *To exploit the functionality and graphical processing power of Garland:* Garland naturally supports many of the ideas underlying the design of the Animation Server—the updating of frames in segment order and the use of temporal differencing, for example—and the functionality it provides was fully used in order to simplify the design of the Server. In turn, the idea of an animation server fits in naturally with the Garland processing model and with the Rainbow II concept in general. The Server was designed to appear to Garland as a video-rate frame source which supplies expanded (i.e., uncompressed) frame data on demand.

4. *To develop a workstation-based system that provides an integrated environment for the interactive development of animated sequences:* The close coupling of the Server to Garland results in a potentially powerful workstation system. Text, graphics and animation can all be mixed on the same screen, and rearranged at will via some form of windowing interface. The segmentation mechanism introduces considerable flexibility into sequence replay by allowing the system to be configured exactly as required for a particular application. The animator can trade frame size, frame replay rate and degree of frame approximation against one another, and all of these against the degree of frame compression required. Flexible control over frame replay is facilitated by the re-sequencing mechanism. This supports special-play operation, both programmed and interactive, and transparent automatic sequence error recovery. In addition, it provides a basis for the interactive editing of recorded sequences.

Thus, it is felt that all the research objectives have been met, and, to some degree, exceeded. In particular, the Animation Server architecture devised gives much more control over the frame replay process than at first thought possible for a disc based system. This is due to the adoption of a frame segmentation mechanism in combination with flexible control over the replay pipeline. The generality of the Server's architecture leads to the possibility of its use in applications other than that originally intended. For example,

- *Stills Store:* The Server can be used as a stills store by exploiting its ability to replay frames individually and in a random order. At a frame size of $512 \times 512 \times 8$, the Server has a capacity of approximately 6,700 stills. Any one of these could be fetched and displayed (at the hardware level) within an average of approximately 1/25s. Similarly, at a frame size of $1024 \times 1024 \times 24$[2] any still can be fetched and fully displayed within an average of approximately 1/2s. Such a store has a variety of uses. For example, the BBC have developed a system called Slide File for the storage of stills used within the television industry (photographs, caption frames, cue cards, etc) [West86].

[2]Created by using a set of 12 $512 \times 512 \times 8$ segments.

- *"Map Server":* A variation on the stills store is to utilise the segmentation mechanism to allow scrolling of the display over a single (static) image which is very much larger than the displayable screen area[3]. This image would be defined as a tiling of smaller segments and a "demand paging" system employed; whenever an edge is approached the Server fetches the relevant set of segments, representing the next strip of image, and writes them into the appropriate (currently undisplayed) part of the frame buffer. Smooth scrolling can be implemented by Garland, under the control of a mouse (for example).

## 9.3 Future Work

It is to be hoped that work on any research project is never fully completed, with new ideas and future directions constantly emerging. During the course of this research three main areas have emerged as offering potential for future fruitful investigation:

1. Further investigation of the idea of frame re-sequencing. A mechanism which provides the ability to assemble a sequence dynamically, "on the fly", during replay is unique to the architecture which has been described (or, more precisely, to the adoption of frame compression in conjunction with the use of substantial compressed-frame buffering). In particular, it is interesting to investigate control strategies for such a mechanism which take account of subjective as well as objective factors, with a view to improving the performance or utility of the mechanism. That is, with a human observer as the ultimate receiver in the replay pipeline it is possible to consider strategies which potentially modify the sequence during re-sequencing operations (by repeating or deleting frames, for example) without introducing any *subjective* degradation to replay quality.

2. Incorporation of an Animation Server into a distributed system. Here the Server would be accessed as a central resource with (compressed) frame data transmitted over a local area network to one of several display stations. Such an arrangement was originally proposed for the present system[4]. However, typical network bandwidths available at that time were of the order of 10Mb/s (raw), considerably less than the compressed-frame source bandwidth (i.e., that of the disc—15Mb/s raw). This would have introduced a severe bottleneck in the replay pipeline. Such a problem is much reduced with newer networks. For example, the Cambridge Fast Ring (CFR) has a design bandwidth of 100Mb/s and a practical (raw) bandwidth of between 40 and 50Mb/s [Hopp88]. There has been increased interest over recent years in the transmission of video data across networks, particularly as part

[3]A good example of such an image is a map (or set of plans), hence the choice of name.

[4]Hence the use of the name "server" in the system's title.

of integrated service systems (i.e., voice, video and data). Note that the use of image compression is unavoidable with present network bandwidths, a situation which looks set to continue for some time to come.

3. The investigation of parallel transfer storage systems. A principal aim of the Animation Server was to give the user as much control as possible over the use of the available disc bandwidth. This freedom is encapsulated within the frame segmentation mechanism which allows the system to be configured according to the needs of the particular application. However, the degree of lossless compression available is fundamentally limited (see Chapter 3) and the use of approximation is not always appropriate. Consequently, there will always be situations for which no configuration of the Server (with respect to frame size, replay rate, etc) achieves the desired replay performance. In such situations the only effective solution is to employ video-rate frame storage. For such storage it is becoming ever more feasible, both economically and technically, to consider the use of multiple disc drives operating in parallel (as discussed in Section 2.3.5). For example, an array of 16 40MB drives could be used, giving a total storage capacity of 640MB[5]). These drives can be much simpler (and therefore more robust) than a single larger drive, and use more proven technology. Then, for individual drive transfer rates of 10Mb/s (say), the total transfer rate is in the region of 160Mb/s. This figure is similar to that successfully used in the BBC Television Animation Store for replaying uncompressed video. However, it is proposed that such a disc array would be used in *conjunction* with image compression. In a hybrid system the use of compression would greatly improve the frame capacity and total replay time of the system, but it is not relied upon for achieving replay. Such an approach has not previously been investigated.

## 9.4 Conclusions

This dissertation set out to investigate computer-based alternatives to film and video tape for the replay of animated sequences. All known previous real-time playback systems were examined and described. This analysis led to the rejection of a software-only based approach, on the grounds that such an approach has inherent performance limitations, and to the investigation of hardware support for frame replay. Two key hardware approaches were examined in some detail, namely the BBC Television Animation Store and the General Electric/Intel Digital Video Interactive (DVI) system.

The Television Animation Store uses a modified disc to achieve a very high frame transfer rate (i.e., video rate) without the need for image compression; DVI employs a low transfer rate optical disc in conjunction with a very high degree of image compression. Thus, given a continuous spectrum of possible replay

[5]And at a total cost which is comparable with that of a single 640MB drive such as the Maxtor XT-8760E.

systems, these two approaches represent the two extremes. The advantages and disadvantages of the approaches are very much as expected. The BBC system achieves high replay quality, supports interactive operation and video editing (for which it was designed), but has a strictly limited total replay time (approximately 30s). DVI, on the other hand, achieves extended replay times (minutes to hours) on (potentially) inexpensive equipment, but supports replay only, is non-interactive, and suffers from limited image quality.

The Animation Server discussed in this dissertation is an attempt to design a system which lies in the unexplored middle ground between these two extremes. The aim was to exploit the advantages of both approaches, whilst minimising the disadvantages. The Server employs an unmodified, medium performance, disc drive in conjunction with a low to moderate degree of image compression. The resulting system supports the interactive and flexible replay of frame sequences and gives a reasonable total playback time (measured in minutes). Its principal limitation is that it doesn't support the generalised replay of arbitrarily complex frames.

A key application identified for the Animation Server was that of the interactive development of animated sequences within a workstation environment. It is felt that the new system meets the requirements of this application more closely than all previous real-time playback systems.

The last few years have witnessed a continual improvement in the performance of both graphics workstations and of magnetic/optical disc technology. Consequently, workstation-based real-time playback is becoming ever more practicable and increasingly attractive as an alternative to real-time animation. The advantages of real-time playback over real-time animation were discussed in Section 1.5.2. Note that even when operating at half speed, 12½ fps[6], the *present* design of the Animation Server exceeds the target performance of most current real-time systems (e.g., the IRIS workstation, which has a target replay performance of 10 fps [Akel88]). Thus, the use of real-time playback in workstations can be expected to increase and the future of non-real-time animation looks assured for some time to come.

[6]Requiring only 2.5:1 compression.

# References

[Ackl80] B. Ackland and N. Weste. Real Time Animation Playback on a Frame Store Display System. *Proceedings of ACM SIGGRAPH '80, Computer Graphics*, **14**(3):182–188, July 1980.

[Akel88] K. Akeley and T. Jermoluk. High-Performance Polygon Rendering. *Proceedings of ACM SIGGRAPH '88, Computer Graphics*, **22**(4):239–246, August 1988.

[Aned88] C. Anedda and L. Felician. P-Compressed Quadtrees for Image Storing. *The Computer Journal*, **31**(4):353–357, August 1988.

[Asal86] M. Asal, G. Short, T. Preston, R. Simpson, D. Roskell and K. Guttag. The Texas Instruments 34010 Graphics System Processor. *IEEE Computer Graphics and Applications*, **6**(10):24–39, October 1986.

[Baec69] R. M. Baecker. *Interactive Computer-Mediated Animation.* Ph.D. thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, June 1969. Available as technical report MAC-TR-61.

[Baec79] R. Baecker. Digital Video Display Systems and Dynamic Graphics. *Proceedings of ACM SIGGRAPH '79, Computer Graphics*, **13**(2):48–56, August 1979.

[Barn88] M. F. Barnsley and A. D. Sloan. A Better Way to Compress Images. *Byte*, **13**(1):215–223, January 1988.

[Bart78] R. A. Bartolini, A. E. Bell, R. E. Flory, M. Lurie and F. W. Spong. Optical Disk Systems Emerge. *IEEE Spectrum*, **15**(9):20–28, August 1978.

[Bell86] F. A. Bellis and M. A. Parker. *An Experimental Helical Scan DVTR.* Technical Report BBC RD 1986/15, BBC Research Department, Engineering Division, December 1986.

[Bult79] K. Bulthuis, M. G. Carasso, J. P. J. Heemskerk, P. J. Kivits, W. J. Kleuters and P. Zalm. Ten Billion Bits on a Disk. *IEEE Spectrum*, **16**(9):26–33, August 1979.

[Catm78] E. Catmull. The Problems of Computer-Assisted Animation. *Proceedings of ACM SIGGRAPH '78, Computer Graphics*, **12**(3):348–353, July 1978.

[Clar82] J. H. Clarke. The Geometry Engine: A VLSI Geometry System for Graphics. *Proceedings of ACM SIGGRAPH '82, Computer Graphics*, **16**(3):127–133, July 1982.

[Clar87] C. K. P. Clarke. *Future Television Systems: Comparison of sequential and interlaced scanning.* Technical Report BBC RD 1987/18, BBC Research Department, Engineering Division, December 1987.

[Cohe85] Y. Cohen, M. S. Landy and M. M. Pavel. Hierarchical Coding of Binary Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-7**(3):284–298, May 1985.

[Cole87] A. J. Cole. Compaction Techniques for Raster Scan Graphics using Space-filling Curves. *The Computer Journal*, **30**(1):87–92, February 1987.

[Conn72] D. J. Connor, R. C. Brainard and J. O. Limb. Intraframe Coding for Picture Transmission. *Proceedings of the IEEE*, **60**(7):779–791, July 1972.

[Corm87] G. V. Cormack and R. N. S. Horspool. Data Compression Using Dynamic Markov Modelling. *The Computer Journal*, **30**(6):541–550, December 1987.

[Cowl84] M. F. Cowlishaw. Bit Requirements for Monochrome and Colour Pictures. In *Eurodisplay '84 Proceedings*, pages 107–108, September 1984.

[Csur79] C. Csuri, R. Hackathorn, R. Parent, W. Carlson and M. Howard. Towards an Interactive High Visual Complexity Animation System. *Proceedings of ACM SIGGRAPH '79, Computer Graphics*, **13**(2):289–299, August 1979.

[Denb86] M. J. Denber and P. M. Turner. A Differential Compiler for Computer Animation. *Proceedings of ACM SIGGRAPH '86, Computer Graphics*, **20**(4):21–27, August 1986.

[Dure84] A. J. Durey. *Television Animation Store: Recording Pictures on a Parallel Transfer Magnetic Disc.* Technical Report BBC RD 1984/17, BBC Research Department, Engineering Division, December 1984.

[Eccl83] D. Eccles and J. Robinson. Special Purpose Hardware For 3D Computer Animation. In C. Outwater, editor, *Optics in Entertainment, Proceedings of the SPIE,* **391**, pages 54–58, SPIE—The International Society for Optical Engineers, 1983.

[Egan84] J. T. Egan, J. Hart and R. D. MacElroy. Analyzing and Comparing the Performance of Two Real-Time Playback Systems. *Computers and Graphics*, 8(1):67–79, 1984.

[Fole82] J. D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[Font88] G. Fontenier and P. Gros. Architectures of Graphics Processors for Interactive 2D Graphics. *Computer Graphics Forum*, **7**(2):79–89, June 1988.

[Fox88] B. Fox. Realism at the Movies. *New Scientist*, **120**(1644/1645):52–56, December 1988.

[Fuji84] L. Fujitani. Laser Optical Disk: The Coming Revolution in On-Line Storage. *Communications of the ACM*, **27**(6):546–554, June 1984.

[Garg82] I. Gargantini. An Effective Way to Represent Quadtrees. *Communications of the ACM*, **25**(12):905–910, December 1982.

[Gill87] C. H. Gillard. Error-Correction Strategy for the New Generation of 4:2:2 Component DVTRs. *Journal of the SMPTE*, **96**(12):1173–1179, December 1987.

[Glau85*a*] T. H. Glauert. *Presentation Issues for Interactive Computer Graphics*. Ph.D. thesis, University of Cambridge Computer Laboratory, Cambridge, England, October 1985.

[Glau85*b*] T. H. Glauert and N. E. Wiseman. Real-Time Image Combination. In *MICAD '85: Proceedings of the 4th European Conference on CAD/CAM and Computer Graphics*, pages 68–73, 1985.

[Gome85] J. E. Gómez. TWIXT: A 3D Animation System. *Computers and Graphics*, 9(3):291–298, 1985. Re-printed in revised form from Eurographics '84 Proceedings.

[Gonz77] R. C. Gonzalez and P. A. Wintz. *Digital Image Processing*. Addison-Wesley, Reading, Mass., 1977.

[Gupt81] S. Gupta and R. F. Sproull. A VLSI Architecture for Updating Raster-Scan Displays. *Proceedings of ACM SIGGRAPH '81, Computer Graphics*, **15**(3):71–78, August 1981.

[Hack77] R. J. Hackathorn. ANIMA II: A 3-D Colour Animation System. *Proceedings of ACM SIGGRAPH '77, Computer Graphics*, **11**(2):54–64, July 1977.

[Hamm80] R. W. Hamming. *Coding and Information Theory*. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1980.

[Hask72] B. G. Haskell, F. W. Mounts and J. C. Candy. Interframe Coding of Videotelephone Pictures. *Proceedings of the IEEE*, **60**(7):792–800, July 1972.

[Hask81] B. G. Haskell and R. Steele. Audio and Video Bit-Rate Reduction. *Proceedings of the IEEE*, **69**(2):252–262, February 1981.

[Hear86] D. Hearn and M. P. Baker. *Computer Graphics*. Prentice Hall International, Englewood Cliffs, New Jersey, 1986.

[Heck82] P. Heckbert. Colour Image Quantization for Frame Buffer Display. *Proceedings of ACM SIGGRAPH '82, Computer Graphics*, **16**(3):297–307, July 1982.

[Hopp88] A. Hopper and R. M. Needham. The Cambridge Fast Ring Networking System. *IEEE Transactions on Computers*, **37**(10):1214–1223, October 1988.

[Huff52] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of the IRE*, **40**:1098–1101, September 1952.

[Isai85] J. Isailović. *Videodisc and Optical Memory Systems*. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1985.

[Jain81] A. K. Jain. Image Data Compression: A Review. *Proceedings of the IEEE*, **69**(3):349–389, March 1981.

[Kim86] M. Y. Kim. Synchronized Disk Interleaving. *IEEE Transactions on Computers*, **C-35**(11):978–988, November 1986.

[King85*a*] T. R. King. *Garland GPIPE Preliminary Specification*. Rainbow Group Research Note, University of Cambridge Computer Laboratory, Cambridge, England, April 1985.

[King85*b*] T. R. King. *GBus Specification 0.1 (Preliminary)*. Rainbow Group Research Note, University of Cambridge Computer Laboratory, Cambridge, England, August 1985.

[King88] T. R. King. *Parallel Image Manipulation Machine Architecture*. Ph.D. thesis, University of Cambridge Computer Laboratory, Cambridge, England, February 1988.

[Kirb88] D. G. Kirby. *Television Animation Store: Control system operation and implementation*. Technical Report BBC RD 1988/4, BBC Research Department, Engineering Division, May 1988.

[Knee88] M. J. Knee and N. D. Wells. *Bandwidth Compression for HDTV Broadcasting: Investigation of some adaptive subsampling strategies*. Technical Report BBC RD 1988/9, BBC Research Department, Engineering Division, July 1988.

[Knig82] B. J. Knight. *Portable System Software on Personal Computers on a Network*. Ph.D. thesis, University of Cambridge Computer Laboratory, Cambridge, England, April 1982. Also available as Cambridge University Computer Laboratory, Technical Report No. 26.

[Kret85] F. Kretz and D. Nasse. Digital Television: Transmission and Coding. *Proceedings of the IEEE*, **73**(4):575–591, April 1985.

[Kunt85] M. Kunt, A. Ikonomopoulos and M. Kocher. Second-Generation Image-Coding Techniques. *Proceedings of the IEEE*, **73**(4):549–574, April 1985.

[Lang84] G. G. Langdon. An Introduction to Arithmetic Coding. *IBM Journal of Research and Development*, **28**(2):135–149, March 1984.

[Levo77] M. Levoy. A Color animation system based on the Multiplane Technique. *Proceedings of ACM SIGGRAPH '77, Computer Graphics*, **11**(2):65–71, July 1977.

[Lipp80] A. Lippman. Movie-Maps: An Application of the Optical Videodisc to Computer Graphics. *Proceedings of ACM SIGGRAPH '80, Computer Graphics*, **14**(3):32–42, July 1980.

[Magn85*a*] N. Magnenat-Thalmann and D. Thalmann. *Computer Animation—Theory and Practice*. Springer Verlag, Tokyo, 1985.

[Magn85*b*] N. Magnenat-Thalmann and D. Thalmann. An Indexed Bibliography on Computer Animation. *IEEE Computer Graphics and Applications*, **5**(7):76–86, July 1985.

[Magn87] N. Magnenat-Thalmann and D. Thalmann. *Image Synthesis—Theory and Practice*. Springer Verlag, Tokyo, 1987.

[Max86] *Maxtor XT-8000E OEM Manual and Product Specification*. Maxtor Corporation, 150 River Oaks Parkway, San Jose, California., December 1986.

[Merc73] C. A. Mercer. Buffering for Sustained, High Speed Transfers. *Software—Practice and Experience*, **3**(4):351–354, October–December 1973.

[Musm85] H. G. Musmann, P. Pirsch and H. Grallert. Advances in Picture Coding. *Proceedings of the IEEE*, **73**(4):523–548, April 1985.

[NCR] *NCR 5380-53C80 SCSI Interface Chip Design Manual*. NCR Microelectronics Division, Colorado Springs.

[Netr80] A. N. Netravali and J. O. Limb. Picture Coding: A Review. *Proceedings of the IEEE*, **68**(3):366–406, March 1980.

[Newm81] W. M. Newman and R. F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, Tokyo, Second Edition, 1981.

[Ng88] S. Ng. *Some Design Issues of Disk Arrays.* Technical Report RJ 6590 (63550), IBM Almaden Research Center, San Jose, California., December 1988.

[Nomu87] T. Nomura, K. Yokoyama, S. Nakagawa and K. Kimoto. High-Quality Magneto-Optic Disk Video Recording. *Journal of the SMPTE,* **96**(11):1062–1067, November 1987.

[Oliv83] M. A. Oliver and N. E. Wiseman. Operations on Quadtree Encoded Images. *The Computer Journal,* **26**(1):83–91, January 1983.

[Page83] I. Page. DisArray: A 16x16 RasterOp Processor. In P. J. W. ten Hagen, editor, *Eurographics '83 Proceedings,* pages 367–381, Elsevier Science Publishers B.V. (North Holland), August 1983.

[Pote77] M. J. Potel. Real-Time Playback in Animation Systems. *Proceedings of ACM SIGGRAPH '77, Computer Graphics,* 11(2):72–77, July 1977.

[Prat78] W. K. Pratt. *Digital Image Processing.* Wiley, New York, 1978.

[Prat79] W. K. Pratt, editor. *Image Transmission Techniques.* Academic Press, London, 1979.

[Pull87] A. M. Pullen. *Motion Development for Computer Animation.* Ph.D. thesis, University of Cambridge Computer Laboratory, Cambridge, England, August 1987.

[Rile87*a*] J. L. Riley. *A Review of the Semiconductor Storage of TV Signals: Part 1—Historical Introduction and Design Philosophy.* Technical Report BBC RD 1987/5, BBC Research Department, Engineering Division, July 1987.

[Rile87*b*] J. L. Riley. *A Review of the Semiconductor Storage of TV Signals: Part 2—Applications 1975–1986.* Technical Report BBC RD 1987/6, BBC Research Department, Engineering Division, August 1987.

[Robe62] L. G. Roberts. Picture Coding Using Pseudo-Random Noise. *IRE Transactions on Information Theory,* **IT-8**(2):145–154, February 1962.

[Robi81] J. F. Robinson. *Videotape Recording.* Focal Press, London, Third Edition, 1981.

[Roge85] D. F. Rogers. *Procedural Elements for Computer Graphics.* McGraw-Hill, Singapore, 1985.

[Rose82] A. Rosenfeld and A. C. Kak. *Digital Picture Processing.* Academic Press, London, Second Edition, 1982. Two Volumes.

[Same84] H. Samet. The Quadtree and Related Hierarchical Data Stuctures. *ACM Computing Surveys*, **16**(2):188–260, June 1984.

[Same85] H. Samet. Data Structures for Quadtree Approximation and Compression. *Communications of the ACM*, **28**(9):973–993, September 1985.

[Schr67] W. F. Schreiber. Picture Coding. *Proceedings of the IEEE*, **55**(3):320–330, March 1967.

[SCS84] *SCSI: Small Computer Systems Interface.* American National Standards Committee, X3T9.2/82-2—Revision 14, May 1984.

[Shan48] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, **27**:379–423 and 623–656, 1948.

[Shir86] G. Shires. A New VLSI Graphics Coprocessor—The Intel 82786. *IEEE Computer Graphics and Applications*, **6**(10):49–55, October 1986.

[Shou79] R. G. Shoup. Colour Table Animation. *Proceedings of ACM SIGGRAPH '79, Computer Graphics*, **13**(2):8–13, August 1979.

[Spro83] R. F. Sproull and I. E. Sutherland. The 8 by 8 Display. *ACM Transactions on Graphics*, **2**(1):32–56, January 1983.

[Stew86] I. P. Stewart. Quadtrees: Storage and Scan Conversion. *The Computer Journal*, **29**(1):60–75, February 1986.

[Stra83] W. Strasser. Videodisc. *Computers and Graphics*, **7**(3–4):351–353, 1983.

[Styn85] B. A. Styne, T. R. King and N. E. Wiseman. Pad Structures for the Rainbow Workstation. *The Computer Journal*, **28**(1):68–72, February 1985.

[WDD86] *WD-D200S Disc Link Manual.* Western Digital Corporation, 2627 Pomona Blvd., Pomona, California., 1986.

[West86] M. Weston. *Development of Slide File™—A Digital Store for TV Stills.* Technical Report BBC RD 1986/10, BBC Research Department, Engineering Division, October 1986.

[Whee86] D. Wheeler. Notes on Runlength Coding. October 1986. Personal Communication.

[Whol61] J. S. Wholey. The Coding of Pictorial Data. *IRE Transactions on Information Theory*, **IT-7**(2):99–104, April 1961.

[Wilk84] A. J. Wilkes, D. W. Singer, J. J. Gibbons, T. R. King, P. Robinson and N. E. Wiseman. The Rainbow Workstation. *The Computer Journal*, **27**(2):112–120, April 1984.

[Wilk87] J. H. Wilkinson. A Review of the Signal Format Specification for the 4:2:2 Component Digital VTR. *Journal of the SMPTE*, **96**(12):1166–1172, December 1987.

[Will88] R. D. Williams. The Goblin Quadtree. *The Computer Journal*, **31**(4):358–363, August 1988.

[Wint72] P. A. Wintz. Transform Picture Coding. *Proceedings of the IEEE*, **60**(7):809–820, August 1972.

[Witt87] I. H. Witten, R. M. Neal and J. G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, **30**(6):520–540, June 1987.

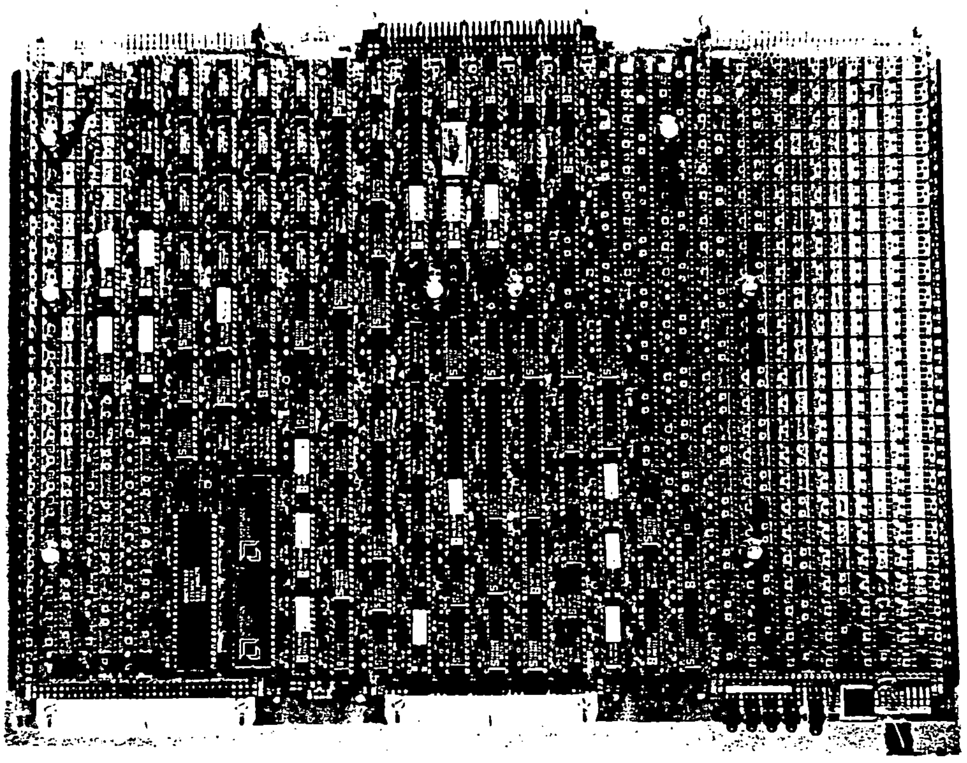
[Wolf82] M. J. Wolfe and D. G. Kirby. A Video Rostrum Camera and an Advanced Stills Store for Television Graphics. In *IEE Conference Publication No. 220*, 1982. International Broadcasting Convention IBC '82.

[Wood84] J. R. Woodwark. Compressed Quad Trees. *The Computer Journal*, **27**(3):225–229, June 1984.

[Wood86] R. W. Wood. Magnetic Recording Systems. *Proceedings of the IEEE*, **74**(11):1557–1568, November 1986. In special issue on Magnetic Information Storage Technology, pp 1494–1590 (7 papers).

[Wyne81] A. D. Wyner. Fundamental Limits in Information Theory. *Proceedings of the IEEE*, **69**(2):239–251, February 1981.

# Colour Plates



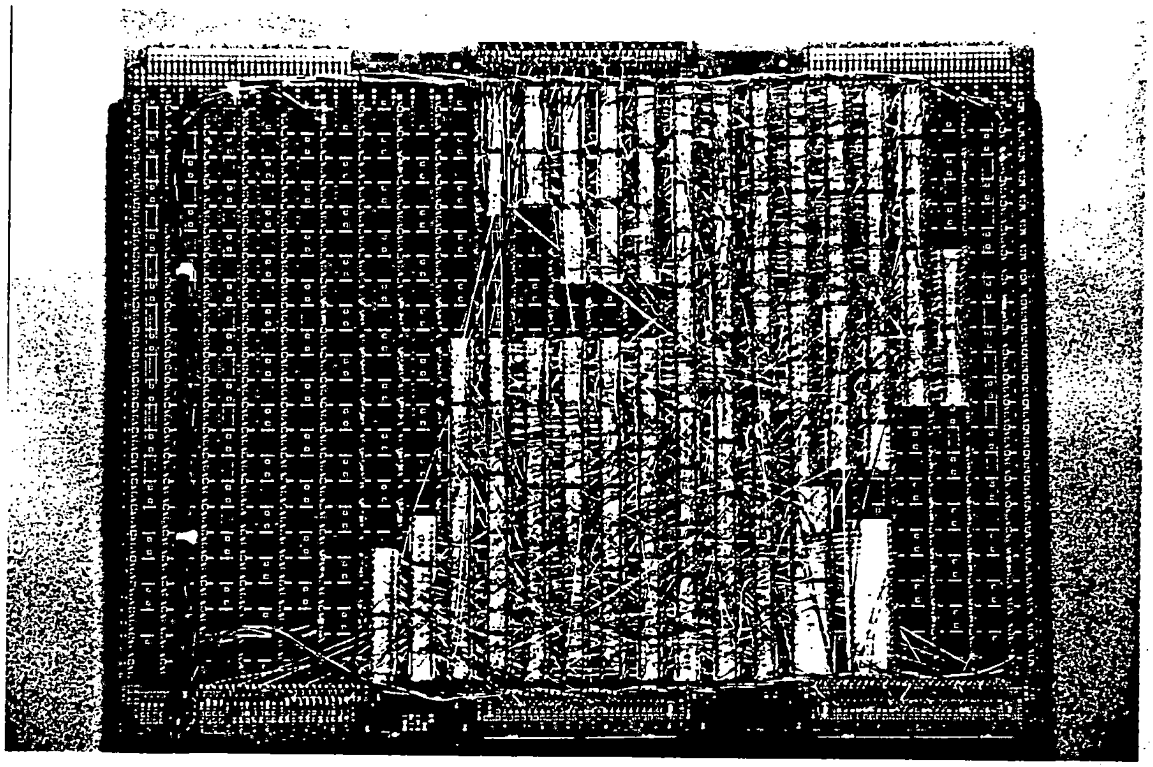
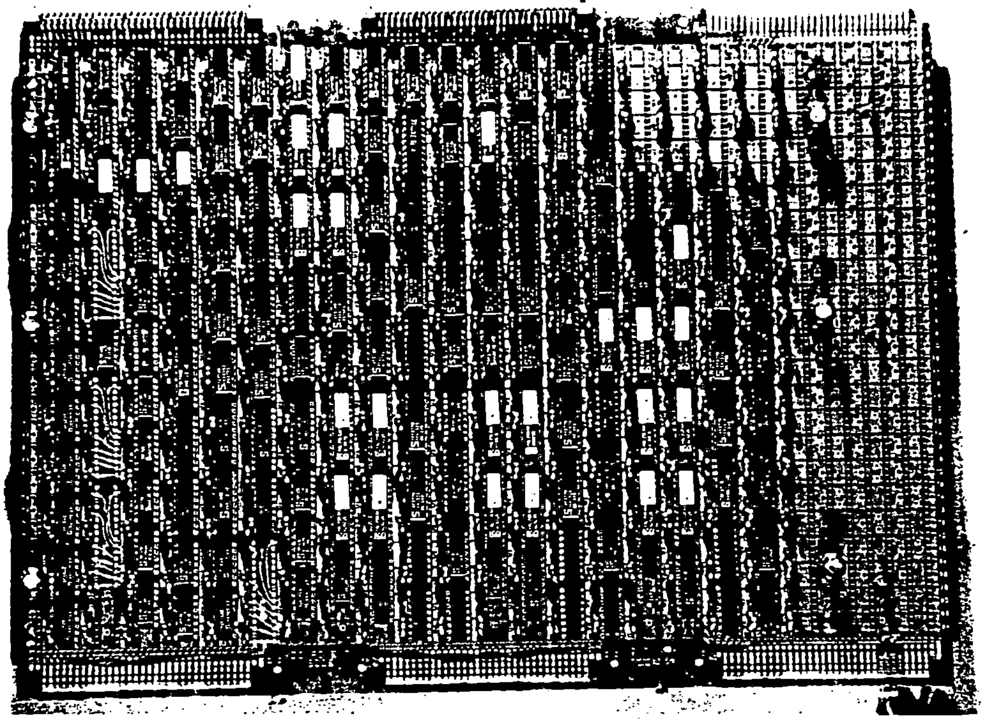


Plate 1: The Frame Transfer Logic card



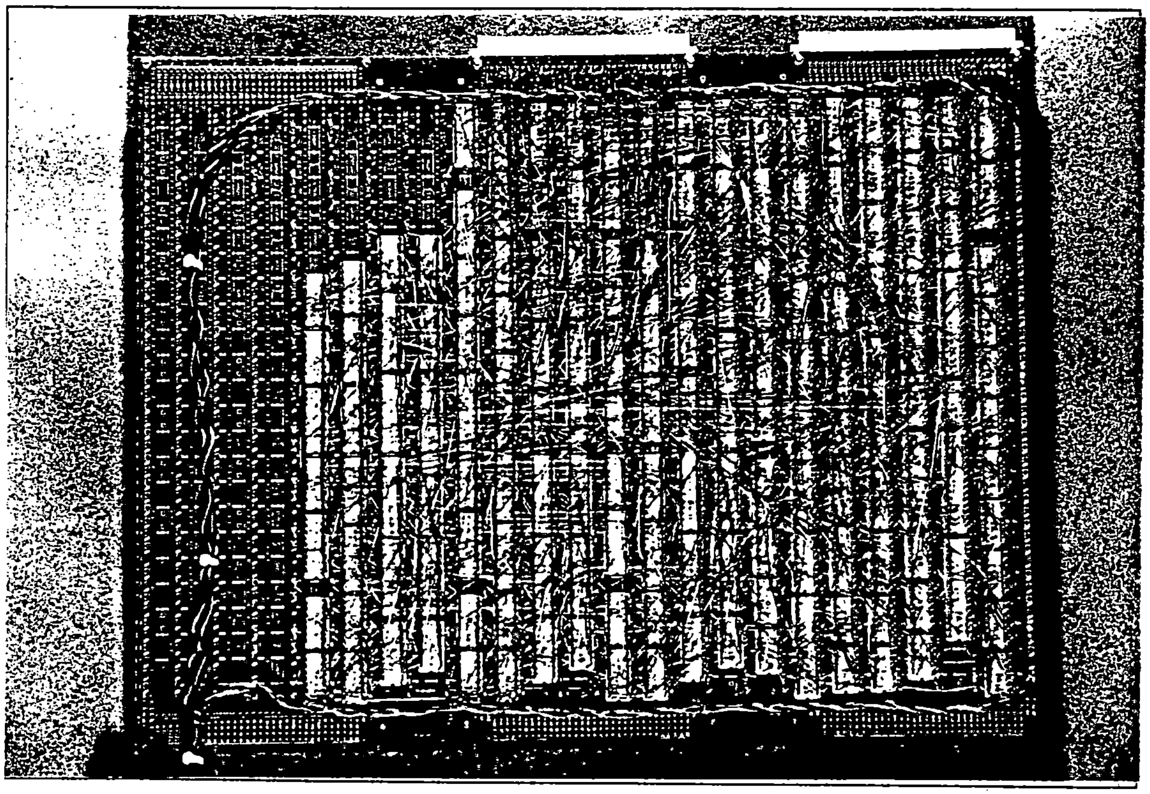


Plate 2: The Frame Decoder card

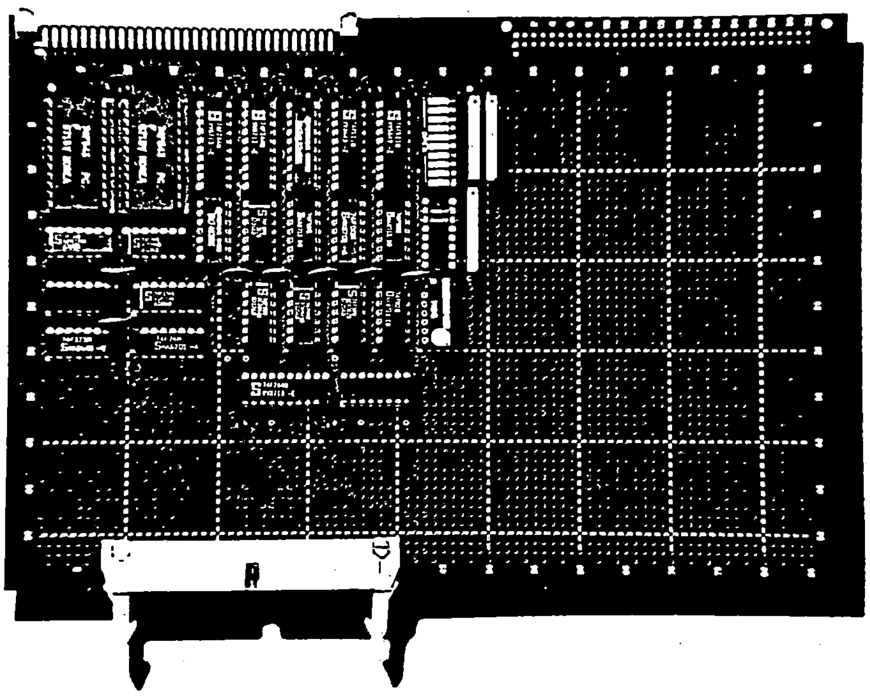


Plate 3: The VMEbus Interface card