Synthetic Movies

by

John A. Watlington

S.B., Electrical Engineering
Massachusetts Institute of Technology, 1987

Submitted to the Media Arts and Sciences Section
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

at the

Massachusetts Institute of Technology
September 1989

Author ______________________________

Media Arts and Sciences Section
August 4, 1989

Certified by ______________________________

Andrew Lippman
Lecturer, Associate Director, MIT Media Laboratory

Accepted by ______________________________

Stephen A. Benton
Chairman
Departmental Commitee on Graduate Students

Synthetic Movies

by

John A. Watlington

Submitted to the
Media Arts and Sciences Section on August 4th, 1989,
in partial fulfillment of the requirements
for the degree of Master of Science

Abstract

The introduction of digital video to the personal computer has generated interest in the integration of image sequences into the user interface. Traditional means of representing an image sequence lack the manipulability desired, requiring that an alternative approach be taken. Previous work in the field of computer graphics and videodisc applications suggest synthetic movies as the solution. A synthetic movie is a movie which is composed as it is viewed, from a description of the movie.

This thesis discusses synthetic movies and their application to the user interface of the personal workstation. An example application, Video Finger, is presented. Video Finger uses a synthetic movie to convey information about other people sharing the computer workspace.

Thesis Supervisor: Andrew Lippman
Lecturer, Associate Director, MIT Media Laboratory

The work reported herein was funded in part by a contract from CPW Technologies.

Acknowledgements

There is a small group of people without whose support, this thesis would probably never have been completed. I would like to thank:

Pascal, who fed me, officed me, and never let me take anything for granted.

Andy, who allowed me the freedom to explore beyond VQ.

Dee, who kept me company, even when the deadlines approached.

Walter and Mike, for keeping the facts straight, the spelling correct and being ready sources of information and support.

My family, for getting me here in the first place.

In addition, I must thank the inhabitants of the Garden, for providing a fun and challenging place to work.

This document was edited and typeset entirely on an Apple Macintosh, using the Microsoft Word text editor and the OzTeX text formatter.

Contents

List of Figures

Chapter 1

Introduction

The field of communication has always availed itself of the latest technology to enhance the message quality. This can be traced from the invention of the printing press, through the early use of electronics by the telegraph industry, to the development of film and later, video, for the transmission of visual information. In the last few years, the ready availability and low cost of digital electronics has sparked a wide interest in digital transmission of visual information. The flexibility of the digital medium, shown by its ability to integrate text, audio, and video, permits many new methods of obtaining and manipulating information.

The large amount of data required to represent an image sequence restricted research into digital video until recent years. A typical motion image sequence, such as standard broadcast TV, requires 8.2 MBytes/sec of data per second[1]. Conven-

[1]NTSC video uses three channels, transmitted with a bandwidth of 4.5 MHz (Luminance), 1.6 MHz (Chrominance I), and 0.6 MHz (Chrominance Q). The signal/noise levels desired require

tional image transmission and compression techniques utilize statistical properties of the image sequence and psychophysical properties of the human visual system to reduce the amount of data associated with the image sequence [Netravali89] [Schreiber86] [Pratt78]. Use of these techniques allows image sequence transmission and display using a state of the art personal computer [Watlington87][Watlington88a].

Conventional compression techniques, although making digital video possible, do not utilize the semantic content of the image sequence. This thesis proposes an alternative method of image sequence transmission: a synthetic movie, where the sequence is synthesized at the receiver from a description of the sequence's contents. An analogy may be drawn to digital text transmission, where two methods are commonly used: facsimile and ASCII. Facsimile encodes the text using a statistical model of the text image. ASCII provides a much more efficient coding by transmitting only the semantic content of the text. The text is reproduced at the receiver by using a local description of the characters in the text (the font) and the transmitted ASCII information. Synthetic movies attempt to separate the semantic content of a sequence from the description of the objects in the scene, allowing separate transmittal of the two components.

---

digitization at 8 bits/channel for the luminance channel and 5 bits/channel for the chrominance channels. Adjusting for the actual display area (70%) provides the bandwidth figure used.

1.1 What are Synthetic Movies ?

Synthetic movies are motion image sequences that are constructed as they are being viewed. They are composed at the receiver, usually under the interactive control of the user. The content of the components and the complexity of the reconstruction vary. Examples of synthetic movies range from videodisc training tools, to simple molecular modeling tools, interactive flight simulators and video games.

Synthetic movies may be categorized by the type of synthesis allowed. Intraframe synthesis permits the actual content of the images being displayed to be manipulated, whereas interframe synthesis is restricted to controlling the temporal evolution of the sequence.

Interframe synthesis has seen much attention over the past decade, due to the introduction of a relatively cheap random-access image frame store in the form of the videodisc. In interframe synthesis, the temporal content of a sequence is synthesized by selecting one frame of several possible frames of video for display. The manipulability of the interframe synthetic movie is rather limited, as every frame conceivably required must be present in the object description. In some videodisc applications this restriction is partially circumvented by the addition of a simple computer graphic overlay, such as text or still images.

Intraframe synthesis refers to synthesizing the actual content of each frame in the motion sequence. This synthesis has been addressed by the computer

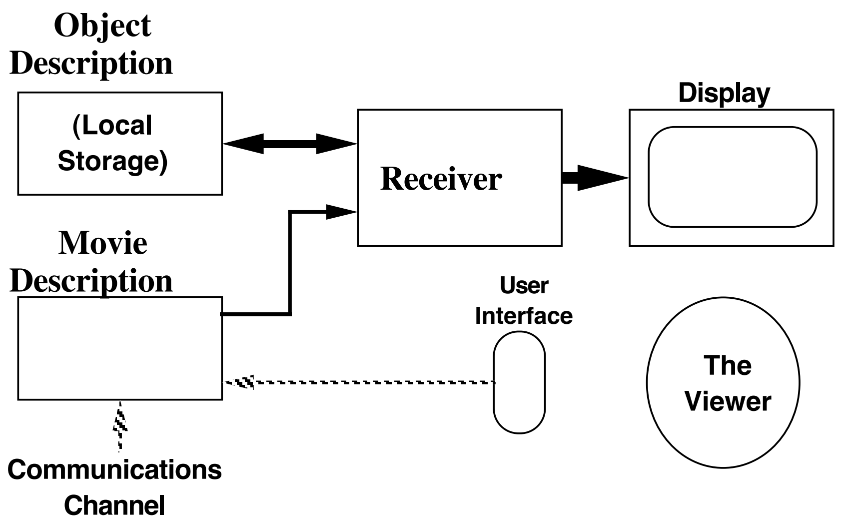

Figure 1.1: A Typical Synthetic Movie Player

graphics community, although due to the complexity normally associated with realistic intraframe synthesis, rarely have the movies been generated in real-time (ie. as they are viewed).

1.2 Reasons for Exploring Synthetic Movies

Synthetic movies are best suited to transmitting certain forms of visual information. This is due to the fact that the image must be reconstructed at the receiver. When the sequence being transmitted is very detailed, or is not being manipulated, synthetic movies are not the medium of choice. An example of this is artistic works, which are usually meant to be viewed in a single, unidirectional sequence. If the interactive transmittal of visual information is desired, however, synthetic movies are ideal.

Synthetic movies, because they are constructed at the receiver, are inherently manipulable. One component of the sequence may be changed, or an alternative one used. In the case of interframe synthesis, the sequence may branch among a wide possibility of cases. In intraframe synthesis, an object/actor on the screen may mimic the sensed motions of a real person[Ginsberg83][Maxwell83]. Implementation considerations usually restrict the manipulability, but it remains as one of the main reasons for developing synthetic movies.

The manipulability of the synthetic movie is not necessary for some applications. There are, however, many uses of image sequences where interactivity is very desirable, or even required. Here are three example uses for synthetic movies:

1.2.1 Interactive Games

Electronic games have become increasingly popular as the cost of the required electronics has plummeted. Games that produce a video display, in particular, have been developed and produced in astounding numbers. The video display generated by one of these electronic games must be viewed as a synthetic movie, because it is an image sequence generated in real time to reflect some internal environment. Object descriptions are usually stored locally in read-only memory, and the movie description is generated by the game's state machine response to user input.

The realism of even the best of these games is rather limited, due to the simplicity of the object descriptions used, but this is changing. Some commercial video games are already using traditional computer graphic techniques to generate the display. The computer processors used in video games are becoming more and more powerful[2].

1.2.2 Low Bandwidth Communication

Synthetic movies may be used for very low bandwidth information transmittal. The "message" of the image sequence is contained in the movie description, which is always transmitted. At the receiver, the movie description is reconstructed using locally stored object descriptions. Once object descriptions compact enough to be transmitted can be reconstructed in real-time, the object descriptions may be transmitted along with the movie description.

The displayed image sequenced will be distorted due to several reasons. One reason is the differences existing between the original object description and the one used for reconstruction. Another reason is the omission of parts of the original image sequence not incorporated into the movie description.

[2] A recent commercial video game release, "Hard Drivin' " from Atari, uses a Motorola 68010 for user input and general processing, an Analog Devices ADSP-2100 floating point signal processor for graphics computation, and a TMS34010 drawing engine from Texas Instruments for rendering to the frame buffer. The performance obtained from this system is approximately 1000 polygons/frame [DSPatch].

1.2.3 The Movie Director

One of the possible future uses of synthetic movies is separating the artistic content of a movie from the limitations imposed by physical filming. A motion picture maker could be freed from the requirement of having the exact lighting required, or the camera[3] in proper location, during the filming. In the editing phase, the movie would be synthesized from the database generated from the filming using any combination of lighting, lens characteristics, and camera position desired. New objects could be added or existing objects removed from the scene. When the desired effect is produced, the movie description for that scene is stored for inclusion into the movie.

The movie distribution could entail transmitting both the movie description and the object descriptions to the receiver, using a digital transmission medium. Alternatively, the movie could be rendered once and recorded for publication and distribution along more conventional channels.

1.3 Synthetic Movies at the User Interface

The earliest computers had no user interface. Batch mode processing limited interaction with the computer to punching programs onto paper cards or tape, then loading them into the computer. Output usually consisted of printing to

[3]This is assuming that a camera capable of recording more than just luminance information is available. See Section 6.4 [Bove89].

a line printer. When the teletype appeared, and simple operating systems were designed to use them; it was a great improvement. Teletypes were later replaced by terminals using cathode ray tubes as their display, although the high cost of memory limited the versatility of these displays to that of an electronic teletype (the DEC VT-100, for example) Due to the relatively low cost of memory today, the modern personal computer/workstation is usually equipped with a bit-mapped display, usually capable of displaying multiple colors. The user interface, however, usually still consists of little more than a window system which allows the user the convenience of having multiple virtual "teletypes".

An exception to this are the object oriented user interfaces descending from the original Smalltalk project at Xerox PARC[Kay77][XEROX Star], such as the Apple Macintosh Desktop. These systems use the graphic capabilities of modern bit-mapped displays to provide the user with a graphically oriented operating system. Files and a hiearchical file system are described using images of applications, documents and filing folders. Common system services (such as the printer, or the routine to destroy a file) are also given icons. If, in addition to being capable of displaying still images, the computer system is capable of supporting moving pictures, how may they be integrated into the user interface? Perhaps the answer may be found by discussing the applications of still images at the user interface.

Still images are commonly used to represent a virtual object which conveys information between the user and the computer. Control bars, buttons, dials, etc., are simple examples of this. Icons are small still images used to represent different elements and functions of the system, such as files and trash cans.

The simplest application of motion sequences in the user interface is an extension of the still image applications. Using interframe synthesis, icons that are themselves image sequences can be made. Although these are interesting, they are limited in the information that can be conveyed.

More interesting applications of motion sequences in the user interface can be found when the content of the image sequence can convey more than one variable of visual information. Intraframe synthesis allows the contents of the frames to reflect the state of many different variables, in a form that is easily interpreted by human users.

Chapter 2

Previous Examples of Synthetic Movies

Synthetic movies have been explored previously many times. Early examples of synthetic movies, with a very limited message, were interactive video games. The graphical output of these video games tended to be of poor quality, with low spatial and color resolution due to the high cost of memory. The next attempts at creating synthetic movies were prompted by the availability of the videodisc, which provided a large, random access frame store. As mentioned earlier, this allows interframe synthetic movies to be generated. Newer videodisc systems include graphic overlays and simple digital effects to allow increased manipulability of the displayed image sequence. The generation of interframe synthetic movies has received a large amount of attention from the field of computer graphics.

2.1 Computer Graphics

Intraframe synthetic movies have been given much attention by the field of Interactive Computer Graphics. Foley and Van Dam, in their classic treatise upon the subject[Foley83], define *interactive computer graphics* to be when the "user dynamically controls the picture's content, format, size, or color...by means of an interaction device". Computer graphic animations are very similar to synthetic movies. The difference between them is the amount of time required to generate the image sequence. A synthetic movie is composed at it is viewed, whereas computer graphic animations are not normally generated in real time. This, however, is a restriction imposed only by the capabilities of the synthesizing hardware. Very expensive flight simulators are capable of rendering images of moderate complexity in real time (ie. as they are viewed). The cost of high-powered graphics engines is decreasing, however, as their ability to generate more complicated real time animations is increasing [1].

Generating a computer animation presents several problems to the animator. The objects in the scene should *look* real, as well as *act* real. The realistic synthesizing (rendering) of images is not simple, requiring that a 2D view be made from a 3D database, incorporating the effects of lighting, lens effects and occlusions. Much work has been done in the field of rendering, resulting in the development

[1] Examples of these are the Stellar Graphics Supercomputer [Apgar88], the Silicon Graphics Workstations[Akeley88], AT+T's Pixel Machine, and UNC's Pixel Planes-4[Goldfeather89]

of techniques that allow realistic generation of synthetic images.

It is the *acting* real that presents a more serious problem in computer animation. Modeling the physics involved in natural motion, such as a tree waving in the wind, or a dog running down a street, is a difficult task. Once the motion *has* been modeled, a method of controlling and synchronizing the motion of objects in the scene must be found.

The control of the objects in an animation has three basic approaches. These are the *guiding* mode, the *animator* level, and the *task* level. In the guiding mode of control, the positions of the animated objects are explicitly controlled by the animator. In the animator level of control, the motion/behavior of objects is described algorithmically, usually in some sort of programming notation. The task level of control stipulates the motion in the scene in terms of events and relationships between the objects [Zeltzer85].

Unfortunately, the synthesis of realistic images is currently possible only by the use of rendering techniques, such as ray tracing, which are not very amenable to rapid computation. It is for this reason that alternative means of generating synthetic movies merit research and development. At some point in the future, advances both in computer technology and graphic rendering algorithms will likely make computer graphic techniques the medium of choice.

2.2 Bolio

A system for the development of interactive simulation, the Integrated Graphical Simulation Platform (IGSP) developed at the MIT Media Laboratory, provides for simple synthetic movies in response to user input, present constraints and task level animation. The IGSP includes an object editor, and behavior modeling tools for the objects in the microworld. A hand position input device, the VPL Dataglove, is provided for the user to interact in real time (actually handle!) with the objects in the microworld. Another input device is a 6 degree of freedom joystick, the Spaceball. Objects in the microworld may be fairly complex: a roach which is capable of walking around under its own control, for example. Another of the objects developed for the microworld is a robot arm that can "play catch", with a user using the Dataglove, using inverse kinematics techniques. [Brett87][Sturman89][Zeltzer88].

2.3 Interactive Video Games

The introduction of direct computer/user interaction with the appearance of the MIT TX-0[2] was soon followed by the introduction of the first interactive video games. Spacewar, a game developed at MIT in 1961 for the DEC PDP-1, for the first time presented a user with a virtual world with which he could interact

[2]Although previous computers were not devoid of user interface, (they had switches and paper tape for input) the TX-0 was one of the first computers to use a teletype as the user interface.

in real time. The graphics display of Spacewar was very limited, using simple line graphics to present the users with two miniature spaceships that could be manipulated in front of a astronomically correct background of stars[Levy84].

The first video games to be available to the public were commercial video games such as "Pong" and "Space Invaders". These first games were very simplistic, due to the level of computer hardware available at affordable costs. Many other games of increasing complexity, however, soon followed.

The introduction of the video game into the consumer market followed soon after the advent of the first monolithic microcomputer. The microprocessor also fueled the personal computer revolution, providing many people with the means of producing and playing interactive video games. These games tended to have poor spatial and color resolution, due to both memory and system bandwidth constraints. The programmers who created these video games were forced to explore methods of providing real-time graphics performance with very simple equipment. They almost universally used a 2D image representation for their objects.

The ready availability of consumer videodisc players spurred the development of electronic games making use of them. The videodisc games produced tended to combine interframe synthesis with simple user input via the host computer. The resultant motion sequence can diverge at any user query step, although due to videodisc storage considerations the divergent sequences are usually reunited at a later time. These games suffer from the need to have an external videodisc player, which is often as large as the computer itself, nearby. Additionally, extra video equipment is necessary to mix and display the resultant video images. In order to display video while the videodisc is searching for a new sequence, it is necessary to use two videodisc players or a frame storage device.

2.4 The Movie Manual

The "Movie Manual" was a prototype electronic book, incorporating text, sound, images and image sequences. It used computer controlled videodiscs to provide a large, random access frame and data store. This data was accessed and displayed to the user by interactive software using a book metaphor. Central to the "Movie Manual" was the concept that the displayed information was being constructed from the stored information as it was viewed. The information base, and the means of presenting it, could be altered by the reader. The receiver utilized information about the contents of the electronic book in displaying (formatting) it.

The hardware consisted of a host minicomputer, equipped with a medium resolution framestore, two or more videodisc players, and video mixing equipment to superimpose the computer graphics and video. The framestore was used to provide still computer graphics images and text, but was not used in generating image sequences. Although this limited the electronic book to interframe synthesis, the

"Movie Manual" is a good example of a synthetic movie [Backer88].

2.5 The Oxyacetylene Welding Simulator

Another example of a synthetic movie is a training simulator designed by D. Hon of Ixion Systems to teach oxyacetylene welding [Ixion][Brush89]. This simulator uses a videodisc as a large framestore to store the images required for the simulator. The student is coached in the technique of using a welding torch by a personal computer controlling the videodisc player. An actual welding torch (without real gases) is used as an input device. Its three dimensional position, and the state of the two gas valves are used as input to the computer.

The student is first taught the procedure for lighting the torch. The computer uses images from the disk to simulate the flame that would be produced were the student using a real oxyacetylene welding torch. The desired flame is also shown for comparison. Once the student has adjusted the gas flow valves to the proper settings, the simulation continues.

On the horizontal monitor face, an image of two pieces of steel sheeting is shown. The student uses his welding torch to "weld" the two sheets together. As the weld progresses, the controlling personal computer displays its evaluation of the student's work by selective retrieval of images from the videodisc frame store. If the weld being made would be flawed, the computer will display a flawed weld. If the weld is acceptable, a good weld is displayed. The interframe synthetic movie generated reflects the internal model of the weld maintained by the personal computer.

2.6 The Elastic Charles

A very recent example of an interframe synthetic movie, utilizing advances in computer technology, is the "Elastic Charles" project [Brøndmo89]. The "Elastic Charles" is a hypermedia journal, a prototype electronic magazine. It contains a collection of multimedia stories about the history, ecology, current news and usage of the Charles River in Massachusetts. These stories are primarily visual, but are augmented with text, graphics and sound. Hypermedia references (links) may be made between text, graphics and sound and video segments.

The hardware platform for "Elastic Charles" consists of a personal computer equipped with a specialized frame buffer, and a videodisc player. The computer used is an Apple Macintosh II. The ColorSpace II frame buffer allows the Mac to generate an overlay on the output of the videodisc, as well as permitting the videodisc output to be digitized. The videodisc is used for storage of the video sequences, limiting the magazine to interframe synthetic movies.

The traditional hypertext notion of a link is extended in "Elastic Charles" to include references to motion image and sound sequences. Unlike text links, the presence of links from a video sequence are presented in a separate window to avoid disturbing the presentation of the video information. If a link references another video sequence, the link is represented by a “micon”, or moving icon. This icon is a short, miniature sequence of video digitized from the referenced video sequence. Micons are also used to navigate the video sequences, recording the path taken by the user in a recent segments card.

Chapter 3

Basic Components of a Synthetic Movie

The basic components of a synthetic movie are the description of the movie and the description of the objects in the movie. The semantic content of the movie is transmitted in the movie description. This is not to say that the object view generated from the object description does not convey information, but that the presence of that particular view is dictated by the movie description.

The actual structure of the components depends on the implementation of the synthetic movie. For example, a videodisc based synthetic movie uses a very different movie description from a computer graphic animation. If interframe synthesis is used, the movie description will not describe the contents of individual frames, but rather their sequencing. Alternatively, if intraframe synthesis is used, the movie description must contain a description of the contents of each frame.

The object descriptions, typically being relatively large, are usually stored locally at the receiver. They may be pre-transmitted or provided on local mass storage such as videodiscs, CD-ROMs, or hard disks. The movie description may either be included on the mass storage device or transmitted separately.

3.1 The Object Description

The object descriptions contain representations of the objects used in the image sequence. The complexity of the receiver is determined largely by the complexity of the object descriptions. The desirable qualities of the object description are low complexity, small size, and manipulability. Several classes of object representation exist, each with advantages and disadvantages.

3.1.1 3D Representations

A common class of object representations describe an object by specifying a three dimensional surface and the reflectance at every point on the surface. These representations differ in how the three dimensional shape is described. Examples of these representations are polygonal patch descriptions and particle representations[1]. Another common class of object representations attempt to describe objects as

[1] Although some particle systems model the entire volume of the object (voxels), for the purposes of synthetic movies only describing the particles along the surface (surfels) is sufficient[Bove89].

combinations of primitive geometrical objects. Examples of this class are constructive solid geometry (CSG) and superquadric representations.

These representations tend to be very manipulable. The two dimensional view required for display by conventional display devices must be rendered from the three dimensional object description. In this rendering step, any desired object view, using any lighting and lens parameters, may be generated.

The description of the motion (including deformation) of realistic objects presents a difficult problem. This is because most real objects (in particular live objects) are non-rigid, or even more problematic: deformable. The representation typically used for representation of non-rigid objects is an articulated object with links between rigid sub-parts. An acceptable representation of deformable objects has been the subject of much recent research[Terzopoulos88][Platt88]. One new deformable object representation, which is promising due to its low computing requirements, describes the deformations using a modal analysis [Pentland89].

The rendering stage that gives these representations much of their manipulability is computationally expensive, and well beyond the limited computing power available presently in personal computers. Other representations are currently implementable, and have their own advantages.

3.1.2 A 2D Image Representation

A simpler representation is a limited view representation, where a set of 2D views of the object from a limited number of viewing angles are stored. This representation allows for simple, realistic images and it is very fast as little to no computation is required.

The control of this representation is very limited. The manipulations possible are translation, scaling and simple 2D remappings of the object views. The generation of an object view not explicitly contained in the object description is not a simple problem. If a view similar to the one desired is present in the object description, it could be used. Calculation of an error metric to arrive upon the closest approximation may be impossible, given the large number of possible deformations. Interpolation to obtain views not contained explicitly in the object database may be possible.

The methods of encoding the views differs. Full color (either RGB or YIQ) images may be used, or the images may color quantized to compress them. Additionally, the object view data may be entropy or run-length encoded to further lower the bandwidth. The redundancy present between adjacent views should allow a large amount of compression if more advanced techniques of image compression were used, such as Multiscale VQ, or other multi-channel techniques[Netravali89]. Multiscale (Pyramid) encoding would allow superb overlaying and would simplify the adjustment of the image scale and focus, but would require an extremely powerful computer.

3.2 The Movie Description

The Movie Description describes the position of objects in the scene over time. This description may be generated or modified locally in response to user input or input received from external devices. The movie description is the "essence" of the image sequence. It is what needs to be transmitted to recreate the original image sequence at the receiver. If an image detail (such as *which* flag is flying over the fort) is not included in the movie description, it will not be present in the synthesized movie. The receiver, of course, must have the required object descriptions.

3.2.1 Intraframe Movie Descriptions

Intraframe descriptions specify the sequence of images to be displayed. The images must be one of a large set of images present in the object descriptions. Typically a sequence of images are associated together for display, with the ability to enter and leave the sequence at any particular frame. The movie description describes the order in which the sequences are viewed.

In many intraframe synthetic movies, the movie description is actually a set of constraints and procedures for manipulating the sequences. In Hypermedia, for example, a set of procedures for browsing through information, with the ability to establish and examine links to related information, is provided[Brøndmo89]. The course of the movie is determined by the user input received, possibly prompted by additional visual or aural cues.

3.2.2 Interframe Movie Descriptions

Interframe descriptions describe the actual contents of the image sequence, in a frame by frame manner. The computer graphics community has developed several synthetic movie scripting languages for describing their animations [Reynolds82][Feiner82][Fortin83]. These languages differ in many respects, but they share several details in common:

- The basic unit of time in the movie descriptions is the displayed frame.
- Support for parallel (and synchronized) execution of object motion descriptions.

The movie description may not be explicitly defined in some interframe synthetic movies. Instead, these movies are driven directly by software state machines or constraint systems. Indeed, examples of these may be more numerous than explicitly defined movies. They include interactive video games, flight simulators, interactive graphical simulators[Zeltzer88] and many applications at the user interface.

Chapter 4

Video Finger: An Example Synthetic Movie

This thesis presents an example implementation of a synthetic movie, Video Finger. Video Finger is an application which uses a synthetic movie to convey information about the state of the computer workplace. It interfaces with existing UNIX information servers over a local area network to obtain information about users: their login state, and what they are currently doing. It was suggested by, and named after, the UNIX finger utility[1]. The main use of the finger utility is to find out who is logged in to a computer system, and what they are doing or how long since they did something.

[1] Actually, the process currently being executed by a user is only displayed by a variant of the finger utility, the *w* utility.

```
3:08am  up 3 days, 12:18,  26 users,  load average: 0.70
User     tty       login@  idle   JCPU   PCPU  what
sab      tty12     9:51am  9:01   1:49   1:11  emacs -f rmail
wad      tty16     2:57am            1         finger
kazu     tty36    11:11am 15:21      1      1  rn
chen     tty41    11:07pm  1:34      9      6  rlogin rebel
vmb      tty58     9:23am 10:17   3:25      6  -tcsh
gslee    ttyp2     3:05am           18     17  col
pat      ttyp3     4:14pm 38:17      1         emacs ch2.tex
jh       ttyp6     3:35am         6:12      5  -tcsh
lacsap   ttypa     8:54pm 50:28   1:20      8  more foobar
klug     ttypb     1:43am  1:09     10         tcsh
walter   ttyr0    11:25am 36:15     40      4  emacs isabel
wave     ttyre     2:18pm  1:49   1:14      3  -tcsh
```

Figure 4.1: Typical UNIX finger text output

4.1 Video Finger Description

The traditional output of the finger utility is text in a chart format, listing selected information. An example output is shown in Fig. 4.1. Alternatively, visualize a video interface to the same program: A window on the user's display shows a preselected scene, perhaps a local computer terminal room or set of offices. In this scene are the people sharing that computer system[2]. The people being observed may actually be working at home, or in another country. The task being performed by the people in the scene is indicative of their computer usage. If, for example, they have been idle for longer than a preset threshold, they appear to have fallen asleep. A typical Video Finger window is shown in Fig. 4.2.

[2] I use the term computer system loosely. The widespread popularity and increasing power of personal workstations imply that most systems are actually a number of computers shared by a group, and connected over a local area network, such as Ethernet.

If the window is allowed to remain, the scene within changes to reflect the changing computer environment. When a person logs into the system, they enter the scene. When they log out, they exit the scene. They stop doing one task, say reading the news, put down the paper, and start doing another task, such as writing. The user may select a person by clicking on them with a mouse (or touching a touch-screen), and then ask the application for more specific information than is being shown (such as the exact name of the task being run by the person). The information being presented could be varied. A voice-mail system where the user sending the mail actually reads the mail to you is a simple extension. Or a “talk” utility where the person actually turns around and does that, on your screen.

4.1.1 Video Finger as a Synthetic Movie

Video Finger synthesizes the image sequence displayed in response to changes in remote user activity. The description of the sequence is generated by a routine that monitors the activity of the remote users. The descriptions of the objects/persons in the sequence reside in local storage in the receiver, although they could be obtained from a remote file server over the same local area network used to monitor the remote user information.

The image sequence being synthesized by Video Finger can be visualized as the output of a software camera being aimed at the computer system. Occasionally, when the system is at rest, the output of the camera will be still and nothing in

Figure 4.2: Video Finger Display

the sequence will be changing. At other times, when the many users are logging in or out, the camera will record the five o'clock rush and show much action in the sequence.

4.2 Basic Design Issues

The hardware platform chosen for Video Finger was the Apple Macintosh IIx personal computer, equipped with the IranScan video card. The Mac II was chosen because it is a personal computer, yet it supports a graphics oriented user interface well suited to integration with image sequences. The IranScan video card is the result of a joint effort by the Media Laboratory and Apple Computer to develop a system capable of decoding and displaying compressed image sequences. IranScan is designed to decode vector quantized images, but may be used to display color quantized images with up to 4096 colors as well.

4.2.1 Hardware Capabilities

The Macintosh IIx is a 32 bit microcomputer, with a 16 MHz Motorola 68030 and up to 8 MByte of RAM. It has a floating point coprocessor, the Motorola 68882. A SCSI mass storage interface allows the use of hard disks and CD-ROM optical drives. An expansion bus, the NuBus, allows additional memory, processors, and peripherals to be integrated into the system. Video display devices are connected to the NuBus. System expansion normally includes separate, specialized processors connected to the NuBus to accelerate sound and graphics synthesis and processing. Although the theoretical data transfer rate over the NuBus is in excess of 35 MByte/sec., actual data rates obtained between the Mac II processor and a peripheral on the NuBus are much lower, on the order of 4 MBytes/sec [Baumwell88]. Due to the lack of a DMA controller, the Mac IIx is not capable of sustaining this rate.

The IranScan frame buffer (shown in Fig. 4.3) consists of two frame buffers, one of which may be superimposed over the second. The "top" frame buffer (Ollie) is used by the Macintosh operating system to display the Desktop, and supports

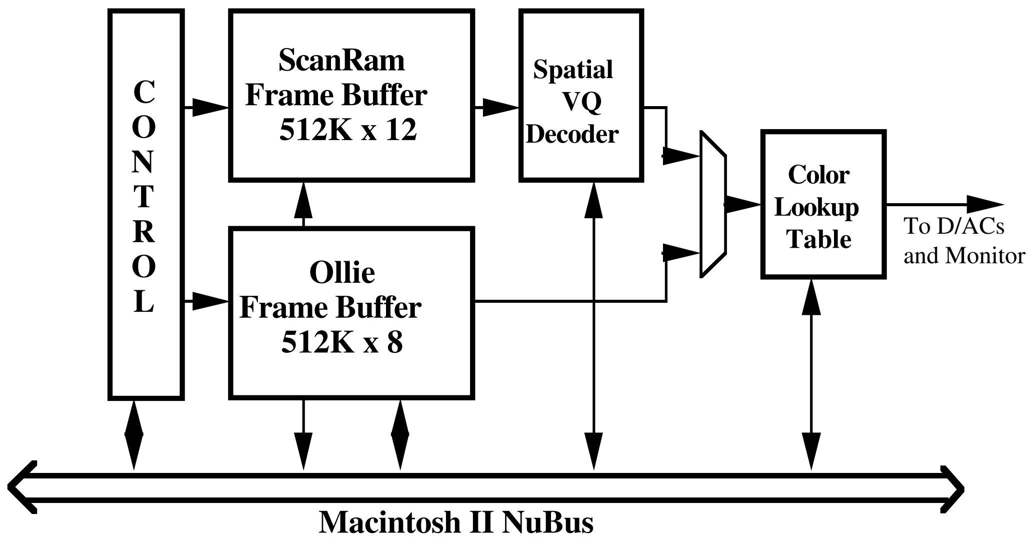

Figure 4.3: Block Diagram of the IranScan Frame Buffer

pixels from 1 bit to 8 bits deep. The "bottom" frame buffer (ScanRam) is used for display of image sequences. It supports pixels from 8 to 12 bits deep. IranScan was designed as a tool for exploring vector quantization of image sequences. In this application, however, I am using IranScan because of flexibility that ScanRam provides for display of more conventional image sequences. An image displayed in a ScanRam window may be double buffered without redrawing the entire screen. This requires less screen memory, as well as allowing faster screen updating. In addition, the pixel depth provided by ScanRam allows the use of realistic color [Heckbert82]. The color lookup table (CLUT) hardware also supports double buffering, allowing it to be updated without disturbing the displayed image.

The display window generated by Video Finger has a content region of 320x240 pixels, which is one-quarter of the screen. This size reduces the drawing requirements to allow real-time animation, yet is still capable of conveying much information.

If a Macintosh IIx can draw 1 Mpixels per second to the screen, it should be capable of displaying thirteen frames of video (320x240) per second. If it is equipped with 8 MBytes of RAM, it can display around seven seconds of such video. Actual applications introduce overhead, however, making this predicted frame rate optimistic.

4.2.2 Object Representation

There are several possible representations for the object data, varying in complexity and manipulability. Unfortunately, the Mac II is not powerful enough to implement even a simple three dimensional rendering in real time, requiring a larger, less complex representation[3]. The object representation chosen for Video Finger is the 2D image set representation. This representation, although restrictive and memory intensive, is very simple to display. If desired, the data for the object representations could be rendered from a more complete, computer graphic database. This representation also allows a simple description of motion.

[3]A commercial rendering package, Swivel v2.0, running on a Mac IIx, requires around 15 sec. to render a simple (8000 polygons) image with simple shading. If the image is rendered anti-aliased, it requires over one minute.

4.2.3 Motion Description

The concept of using image views as a description of the object's appearance suggests a similar approach to motion description, the one used by Video Finger. Object motion/deformation is described by a series of views, called a "task". Each object has associated with it a set of tasks, which represent a significant motion/deformation for that object. An example of the views composing a simple task is presented is Fig. 4.4.

The tasks are defined using a simple interpreted language which defines which object view to use and any independent object motion. The use of a "task" motion representation constrains the motion of objects to be simple translation, existing tasks and unions of those tasks.

A "task" can only define the motion of one object. The Task Dispatcher, along with a signaling mechanism incorporated into the language, provide for the concurrent object motion necessary for intraframe synthetic movies.

Unfortunately, object descriptions must contain all the views and tasks desired. In Video Finger, this is not a significant problem, as the required object motion is limited. Only about a dozen object tasks were considered necessary for Video Finger[4].

[4]To be more precise: standing, sitting, turning from side to side, falling asleep, waking up, sleeping, starting to read, reading and stopping reading.

Figure 4.4: The Image Views comprising an example Task

4.2.4 Depth

Given the two dimensionality of the object representation being used, other methods must be used to impose the perception of a projection of a three dimensional image. Video Finger provides for simple occlusion, as well as perspective. Each object may have one of 256 discrete depths. The depth indicates the order of drawing: the objects with larger depth (distance into the screen) are drawn first. This provides for simple occlusion, but complex occlusion[5] requires that the object be divided into two parts, one in front of the occluding object and the rest behind. This would be a simple extension to the existing software, but it was not implemented.

[5]Complex occlusion is where an object is both occluding and occluded by another object.

In addition to occlusion, the size of an object is changed to correspond to the depth. This simulates the lens parameters used in "recording" the scene. The scaling value used is derived from the formula for a simple lens: $M_T = f/z_o$, where f is the focal length of the lens being simulated and z_o is the depth of the object [Hecht79]. The focal length used in drawing a frame is determined by the background image being used. The focal length used is the same as that used to record the background, so that objects superimposed over the background appear normal with respect to the background.

Chapter 5

Video Finger: Software Description

The software development environment used for the development of Video Finger was the Macintosh Programmer's Workshop (MPW). The majority of the software was written in MPW C. Some hardware dependent or speed restricting parts of the drawing routines were written in 68020 assembly language. A public domain library of routines, TransSkel [DuBois89], which abstracts out the interface to the Macintosh operating system, was used to simplify program development. The User Interface Toolbox of the Macintosh, along with the Operating System Utilities, provided support for windowing, mouse actions, menus, and user dialogs. This allowed the program to easily support those features, while maintaining a user interface consistent with other Macintosh applications.

5.1 Basic Software Overview

The Video Finger software system is diagrammed in Fig. 5.1. The Task Dispatcher executes any tasks that are being performed, then calls each object to redraw itself. One of the tasks being performed is the User Task, which polls the state of the computer environment via the local area network, modifying the movie generated to match. Each type of object has a software routine (an *object handler*) associated with it that is called to manipulate objects of that type. A common nucleus of drawing and CLUT management routines are provided for use by all the objects. Parameters of the Video Finger display, such as what background image is being used, or the update rate of the display, are changeable from user menus.

5.1.1 Object Handlers

Every instance of a given type of object executes the same object handler routine. The handler routine is called with a pointer to the data structure defining the object[1], a command, and a pointer to optional arguments. All object handlers recognize a small set of basic commands: open, close, and draw. The open command opens an object, loading in necessary view and task information from local storage if needed. The close command closes the object, freeing the associated memory, and the draw command calls the proper drawing routines to draw the

[1]The data structure for each object contains a pointer to the appropriate object handler.

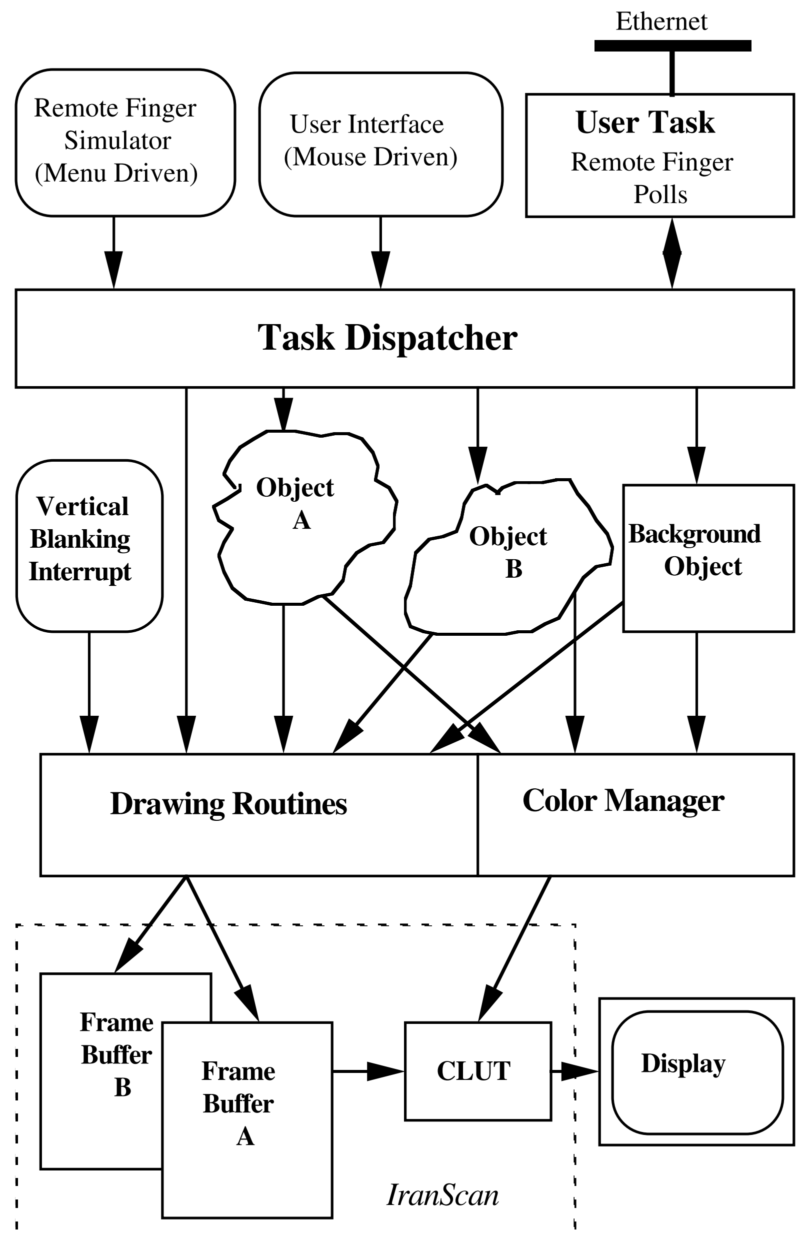

Figure 5.1: Video Finger Software Overview

current object view into the frame buffer.

Optional commands supported by most object handlers are: position, view, and request-task. The position command allows objects to be positioned either absolutely or relatively. The view command request a new view of the object, specified using some characteristic of the view, at a certain scale. The request-task command queries an object to obtain the description of a particular task for that object.

The number of object types is quite small. Only one type is used to represent all the human figures in Video Finger. The other types include background objects, prop objects, and test objects.

5.1.2 Task Dispatcher

The task dispatcher is called whenever there are no system events that require handling. These events are generated by user input, or operating system actions (such as window refreshes). The dispatcher first traverses a linked list of tasks currently being executed, executing each one in turn. Most of these tasks are interpreted by the Basic Task Language (BTL) interpreter. Next, it traverses an ordered linked list of objects in the scene, calling each object with the draw command. The list is ordered such that the objects farther back in the scene are drawn first, and overlayed by later (ie. closer) objects. After all the objects have finished drawing, the task dispatcher sets a semaphore signaling that drawing has finished on the new frame.

5.1.3 Drawing and Color Routines

Common drawing routines provide flicker free generation of the Video Finger window. The window is maintained by the Macintosh Window Manager, which interfaces with the drawing routines. The common drawing routines transparently provide double buffering, as well as decoding of the transparency run-length encoded images as they are drawn. Since the IranScan video card architecture includes a second frame buffer which may be re-positioned without affecting the first frame buffer, double buffering is done in the actual frame memory.

The color map is partitioned and maintained by the color manager. Whenever an object is opened, the object handler requests the needed number of slots from the color manager. The color manager allocates and reserves the requested slots, returning the index of the first slot allocated. The object is then responsible for calling the color manager to load the slots allocated when required. If more than one CLUT is used by the object views, then the object handler is responsible for calling the color manager to update the color map as needed. The small size of the typical object CLUT allows sixteen objects to share the IranScan color map. Since the amount of local memory available usually limits the number of object displayed to less than sixteen, the color manager does not currently degrade gracefully upon running out of free slots.

A routine is installed in the Vertical Blanking (VBL) Queue for the IranScan card which refreshes the display buffers. When called, this routine checks to see that a semaphore indicating the availability of a new frame has been set by the Task Dispatcher. If it has, it repositions the display base address of the ScanRam to display the drawing buffer and hide the previous display buffer. Routines in the VBL Queue for a video card are called by the operating system upon receiving a hardware interrupt from the video card signaling a vertical blanking period. The number of video frames between calls to the VBL routine may be controlled from one of the user menus. It functions as a Fast Play/Slow Play control.

The internal representation of the object views is significant for several reasons. Both the drawing time and the memory required for the view are dependent upon the representation used. In addition, the time required to manipulate the view is dependent upon the representation.

The statistics of the image view data (around 50% transparent pixels) imply that run-length encoding the data would provide a significant gain. Due to the large number of colors allowed each object, however, the non-transparent parts of the image were not amenable to run-length encoding. The final compromise was to run-length encode the transparent portions of the image, allowing the drawing code to draw any length of transparency in a very small, constant time. The non-transparent portions are simply packed into pixels, adjusted for the actual location of the view's CLUT, and transferred to frame memory[2]. The data was actually stored in a non-encoded format on disk and encoded upon being read into local memory. This allowed easy testing of different encoding techniques.

5.2 Basic Task Language Interpreter

The basic unit of object motion, the "task", is defined using a very simple interpreted language, the Basic Task Language (BTL). The task description is obtained by calling the appropriate object handler, and then interpreted using the BTL Interpreter. A command (**btlFrame**) is provided to mark discrete frame boundaries. Each frame, the BTL Interpreter sequentially executes the commands in the task description until a frame marker is reached.

The Basic Task Language consists of a stream of commands with optional arguments. In order to facilitate interpretation, commands are of fixed size (4 Bytes), as are the optional arguments. A list of the BTL commands with a brief description of each is provided is Table 5.1.

The BTL commands are the minimum required to describe an object and its motion in the 2 1/2 D world of Video Finger. The **btlDepth** command adjusts not only the depth, but the perceived scale of the object. The **btlSignal** command provides a simple way of synchronizing the end of one task with the

[2]The simple drawing routine was drawing 1 pixel/3.1 μSec (including transparent pixels). The transparency run-length decoding algorithm can draw 1 pixel/1.8 μSec. It requires 5 μSec to process a length of transparent pixels. For an average object, the drawing time was halved.

Command	Description
btlFrame	Signals the end of the commands for the current frame.
btlView	Indicates which object view should be displayed. Requires one argument indicating the resource number of the object view.
btlPosition	Indicates a relative movement of the object. Requires two arguments indicating the horizontal and vertical offsets.
btlDepth	Indicates a relative depth movement. Requires one argument indicating the depth offset.
btlSignal	Provides a mechanism for communicating between tasks. When a task is executed, a signal handler is specified. This signal handler is passed the value of the one argument.

Table 5.1: Basic Task Language Commands

start of another. When a task is executed, a pointer to a signal handling routine is provided to the BTL Interpreter. If the task contains a **btlSignal** command, the signal handler is called with the command's argument. In addition, the end of a task is automatically signalled to the signal handling routine by the interpreter.

5.3 View Scaling

One of the restrictions imposed by using object views as the object representation is that the views have a particular scale. One of the more useful object manipulations that can be implemented is the scaling of these object views.

Each view in the object descriptions has a scale specified in an accompanying header. When the object view is being drawn, the object handler compares the scale with which the object is currently being drawn with the scale of the view to be drawn. If they are different, an interpolated view is generated and used for display.

A major obstacle to interpolating the image views is their internal representation. Currently the views are stored as transparency run-length encoded color quantized images. The view image data being interpolated is reconstructed using the image's CLUT to RGB values. After interpolation, the resulting RGB value is mapped back into the image's color table using an inverse color lookup table.

The RGB value being mapped is first hashed by taking the most significant 5 bits from each channel and using them as an index into a 32 KByte inverse color table. The number of bits per channel used in the hashing is important in obtaining acceptable image quality. Anything smaller than 5 bits/channel appears very quantized[3], yet larger inverse tables (6 or 7 bits/channels) consume too much memory (256 KBytes or 2 MBytes respectively).

The inverse color table is generated using a full search algorithm to find the most accurate rendition of a particular region of color space among the colors in the object view CLUT. The inverse tables are currently stored as part of the

[3]The degradation imposed by to small of an inverse CLUT is large enough that the benefits of using interpolation over pixel replication disappear. This is due to the regular mapping of the inverse table not being fine enough to discern the low color gradients observed in real images.

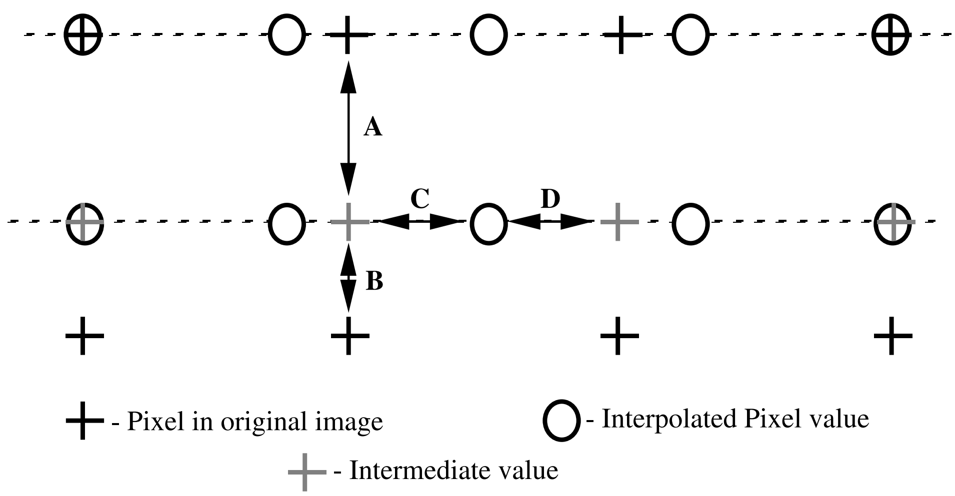

Figure 5.2: Pixel Interpolation

object description to save time when initializing the object. Alternatively, they could be generated from the object's color lookup table.

The interpolation algorithm used allows a rapid scaling of the image by an arbitrary quantity. A depiction of the interpolation algorithm is found in Fig. 5.2. A set of intermediate values is first calculated, using a linear interpolation between two adjacent scan lines of the original image:

$$M_{i,j} = AO_{i,n} + BO_{i,n+1} \tag{5.1}$$
$$B = 1 - A \tag{5.2}$$

In these equations M is the array of intermediate values, O is the array of original pixels, and N is the output array of interpolated pixels.

The final output is then calculated by linearly interpolating horizontally between the two closest intermediate values:

$$N_{i,j} = CM_{m,j} + DM_{m+1,j} \tag{5.3}$$

$$D = 1 - C \tag{5.4}$$

A and C are computed incrementally, by accumulating the vertical and horizontal spacing, respectively, of the new sampling grid N in the original image O. Only the two intermediate values required for calculating the current pixel are stored. All calculations are done in fixed point math, using 16 bits of integer and 16 bits of fraction.

A special interpolation routine decodes the run length encoding of the image's transparent portions while interpolating the view. In addition, the output it produces is itself transparency run-length encoded. The interpolation is performed independently on all three color components of the image: R, G, and B. A typical pixel interpolation requires 12 multiplies and 14 additions per interpolated pixel. If the scaling factor is less than 0.5, only 9 multiplies and 11 additions per interpolated pixel are required. Unfortunately, even using this simple algorithm, the time required to interpolate a normally sized object view using a Macintosh IIx is around 0.8 sec.

5.3.1 View Caching

Since the interpolation algorithm is processor intensive, the interpolated views are cached. Every time an object handler decides that interpolation is necessary, it first checks to see if the desired view (in the desired scale) is present in the View

Cache. If not, it caches the desired view in the cache after interpolating it.

The View Cache is organized as a fully associative cache. It is typically 1 MByte in size, divided into 8 KByte pages. Both these values were obtained by empirical methods. The cache replacement algorithm utilized is a Least Recently Used algorithm. This is implemented by maintaining for each cache entry a count of cache fetches since its last usage. When more pages are needed in the cache, the entry with the largest count is removed.

Without the view cache, the time required for interpolation limits the usefulness of view scaling. In order to fully utilize the view cache, however, a pre-fetching routine should be utilized. This routine would anticipate the use of an object view that required scaling and utilize idle processor cycles to interpolate the view and store it in the View Cache before it is needed. Such a routine could utilize the "task" structure of motion representation in Video Finger to anticipate future need.

5.4 Network Interface

The network interface software allows Video Finger to communicate with other machines sharing a common Local Area Network. The network interface chosen was the Transfer Control Protocol/Internet Protocol (TCP/IP), due to its prevalent use in the Media Laboratory's computing environment. In addition, the UNIX finger utility is already a supported service of the TCP interface software on many machines. The actual software package used was Kinetics TCPort for MPW C v3.0. The physical network interface used was thin-line Ethernet.

Upon initialization, Video Finger starts the User Task, which is responsible for maintaining the User State Tables. These tables record the remote state of every user being monitored. The State Table entry for each user contains a login flag and a state variable. The valid values for the state variable represent the current actions of the user, such as editing, reading news, compiling, idle, etc. The User Task polls the finger server on remote machines for information necessary to maintain the User State Tables. The rate of this polling is limited by the response time of the remote host. It usually requires fifteen to thirty seconds for most hosts to generate a reply.

Chapter 6

Video Finger: Data Preparation

The object representation used for Video Finger requires a set of object views, from which the desired task sequences can be assembled. This set of object views may be generated from a computer graphic database or from actual image data obtained using a real subject. Since one of the goals of Video Finger was to incorporate real imagery into the graphic interface, the object descriptions used were constructed from actual image data.

The object view must be separated from the background present in the original object image. There are many ways of implementing this separation; the most common of which is "blue-screening". In "blue-screening", the object to be filmed is placed in front of a monochromatic matte background. The color of the background is highly saturated, usually blue (hence the name). This allows the background to be separated by post-processing, either by special analog circuitry (a chroma-keyer) or by software. This technique was chosen due to the relatively high resolution segmentation produced, and the ease of implementation. Other techniques considered were not suitable for use with image sequences (laser range camera) or currently provide results with too little resolution (depth-from-focus, depth-from-shading, etc).

The subjects for the prototype Video Finger were asked to stand in front of a blue background, and perform a number of tasks. In order to allow natural motion, the subjects were recorded using a constantly running video camera and video tape recorder. The tasks were each repeated at least twice in order to assure that at least one sequence of each task would be acceptable[1]. The tasks that the subjects were asked to perform varied from subject to subject, but all included a small subset of common tasks, namely twirling around in a chair, standing up and leaving, entering and sitting down, and falling asleep. Other tasks recorded included walking, reading, and writing.

6.1 Filming

The bluescreen consisted of 3/8" plywood, covered with several coats of chroma key blue paint. The subjects were placed approx. 24" in front of the bluescreen, and illuminated by light sources placed inline with (actually about 4" away from)

[1]Acceptable in terms of being representative of the task being recorded. This implied that the tasks were neither performed to rapidly for acurate recording on an NTSC video system, or not separable into single tasks.

the camera lens. This was done in order to reduce shadowing. The camera used was an Ikegami NTSC camera. It was connected directly to an Ampex 1" videotape recorder used for taping the subjects. Although recording the images in NTSC was suboptimal, the fact that the images were band-limited and decimated in later pre-processing lessened the artifacts.

6.2 Pre-Processing

The recorded images of the subjects were examined in order to determine which frames would be selected to represent the subject. Once selected, the views were decoded from the NTSC signal from the videotape recorder into RGB components using a Faroudja NTSC Decoder. Each component was digitized separately using a Datacube frame buffer mounted in a Sun workstation and transferred to a microVax III for further processing. The resolution of the input images was 768x480. The images were then cropped to 640x480, filtered using a half-band lowpass 13-tap 2D Gaussian filter, and decimated by two in each dimension to provide a 320x240 RGB original image set. This image set was then color quantized to provide the actual images used in the object representation. The color lookup table was generated by applying Heckbert's median cut algorithm[Heckbert82] to a histogram generated from a small number of representative images from the image set.

Although the software supports objects having multiple CLUTs, the color statistics of the object views are relatively constant, allowing most objects to be reasonably encoded using one 256 entry color table. This is not to say that the color statistics of a single view are representative of the entire sequence. Instead, a small number (three or four) of representative views should be sampled in generating the CLUT.

6.3 Segmentation

The RGB original image set contained images of the objects against a background. The background region of the image was identified using color segmentation routines developed for this purpose [Watlington88b]. The particular segmentation algorithm used was a region merge algorithm which attempted to form clusters in a two dimensional color space. The color space used for the segmentation was normalized red vs. normalized blue. The output of the segmentation algorithm was an image showing the different regions that had been identified. The background regions (the background was usually identified as three to five separate regions) were then combined to generate a mask image. The mask image was used to set the background region of the color quantized images to a zero pixel value. Zero was the pixel value used to denote transparency.

The edges of an object are usually a problem with a chroma based image segmentation algorithm, due to object interreflectances, pixel averaging (lens blurring), and shadows. If the object edges are not properly identified, different yet

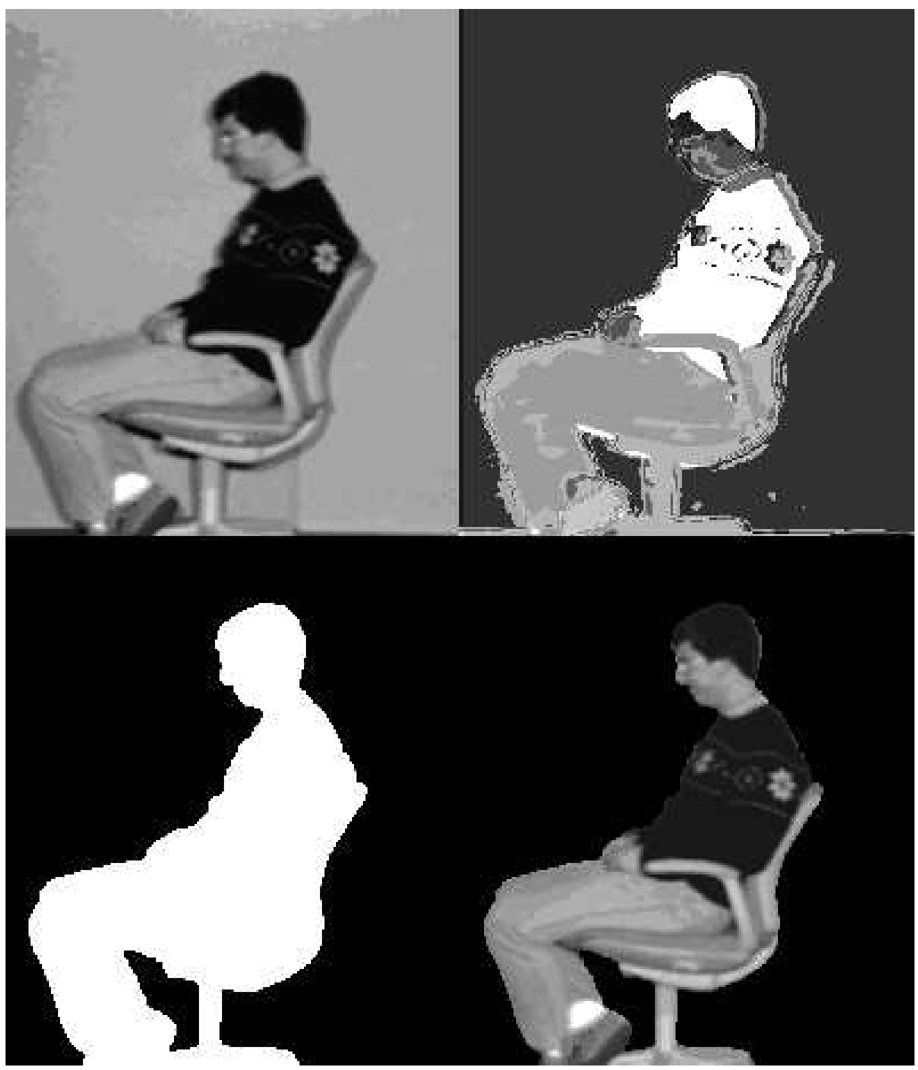

Figure 6.1: Object View at several different stages during processing

similar views will show different object edges where the edge should be the same. When the images are animated (played), the resulting artifact is an object with edges that appear to wiggle back and forth.

The color image segmentation algorithm used performed relatively well, yet still required two steps of hand editing in producing the mask image. One step was identifying which segmented regions actually belonged to the background, the second was eliminating edge location errors around the object. An example object view during several stages of processing is shown in Fig. 6.1. The images are, in clockwise order from the upper left corner, the original image view, the segmented regions, the mask generated from the segmentation information, and the final object view, separated from the background.

These problems could both be alleviated if additional information about the object view was utilized. If another segmentation algorithm is employed, perhaps utilizing motion over time, or utilizing both range data and luminance data, the first step of hand editing could be eliminated.

The images being processed were often highly similar. Due to the use of a sequence of object images to represent motion, many of the images differed only in the motion of one small section. Errors in segmentation could be reduced by using the information about the object edges and areas of shadow gained by the segmentation of the previous images in the sequence.

The segmentation routine was implemented on the Mac II, as part of PicTool, an image processing utility. PicTool was vital in assembling the image sets, as it allowed the images to be easily viewed, segmented, cropped, and retouched as necessary. Additionally, PicTool provided a programming environment for the quick testing of the view manipulation algorithms used in Video Finger.

6.4 Alternative Means of Data Preparation

The object description may be generated automatically, if additional information is recorded by the camera(s) being used. One possibility is the model building camera developed by V. Michael Bove[Bove89]. This camera records range information derived using depth-from-focus methods as well as luminance information for an image. A sequence of these images, or the images recorded by multiple cameras viewing the same scene are used to generate a three dimensional particle representation of the object [Linhardt88]. This database may be converted into other forms for rendering.

Although the computing requirements of rendering a view directly from the object database generated by the camera are too large to consider doing it on the Mac II, the representation used by Video Finger could easily be generated. Unfortunately, a color version of the camera required is not currently available.

Chapter 7

Results

Video Finger has been functioning as a Macintosh application for five months now, in one form or another. The early versions were used to test the object representation, both external and internal. The later versions have included the ability to generate the movie description and animate the display. A small, but representative, set of object descriptions has been prepared, including several backgrounds. The final function to be added was the network interface. When Video Finger first started working, I was startled more than once by the sight of a person entering the window, informing me that a friend had logged in.

7.1 Improvements

There are several areas of the Video Finger implementation that need improvement:

- Currently the object descriptions are maintained in memory, not on disk. This was done due to the time required to transform the view data into the internal representation used by Video Finger. Now that a suitable internal representation has been found, the objects may be stored on disk, and pre-fetched into main memory by the same mechanism proposed for pre-loading the view cache. The memory usage by large (ie. useful) objects is currently the limiting factor is determining the number of objects in the movie.

- The use of a color quantized frame buffer should be re-evaluated. The difficulties introduced when interpolating color values make the use of an RGB frame buffer desirable. The psychophysical limits of the human visual system suggest that a frame buffer using a YIQ representation for color would be more appropriate [Zworykin54]. The peripheral bus bandwidth would be doubled, however, if a "true" color frame buffer is used, requiring a more powerful computer.

- The object edges should be anti-aliased. Unfortunately, anti-aliasing can only be done as the view is being transferred into the display buffer. This could be implemented, using the same technique to interpolate between two color quantized images as is used in view scaling.

- The time required to interpolate the image views is restrictive. In the current implementation, the first few frames of the sequences tend to be slow, until the required views have been cached. The use of a coprocessor on the NuBus to perform the interpolations would accelerate the display, but several very powerful coprocessors would be required to perform the object interpolations in real time.

- The current object libraries are quite sparse. The acquisition and processing of the images required for a single object description is currently a formidable task. The means for improving this have already been discussed, in Sections 6.3 and 6.4.

7.2 Future Work

A more intellegent means of monitoring the external user state is needed. I propose adding a server at the monitored machines for providing the user status. The server could be notified of the receiver's desire to monitor a particular user. Thereafter, the receiver would be notified by the server upon a change in the state of the monitored user. This would decrease the network traffic generated by Video Finger (although this is already quite minimal, especially when compared to a packet video movie !), and reduce the remote processor workload.

The background image is not restricted to still images; it can be a motion image sequence itself. This yields the digital video equivalent of filming a movie scene in front of a screen upon which is projected the scene background. Similarly, the background may be used to convey additional information; perhaps a window in the background, showing the current state of the weather, somewhere in the world.

The addition of sound to synthetic movies is a necessity. Ideally, the sounds associated with a synthetic movie must themselves be synthetic. Two different types of synthetic sound are being investigated: video synchronized speech synthesis and object associated sounds. The synchronization of a synthetic actor with synthetic speech (usually generated by a dedicated speech synthesizer) is not a new idea, it has been illustrated previously. The association of sounds in the soundtrack with particular objects in the video sequence, however, requires interframe synthesis. Video Finger provides a starting point for experimenting with object associated sounds.

7.3 Future Hardware

Video Finger is a very simple example of a synthetic movie, due to the restrictive object representation chosen. A more complex object representation would require a correspondingly more powerful hardware system.

Such systems exist now [Akeley88][Apgar88][Goldfeather89], but the specialized hardware required is very costly. This is changing, however, thanks to advances in VLSI technology. As the chart in Fig. 7.1[1] shows, the personal com-

[1] The data in the chart was obtained from Morris & Dyer[Morris88]

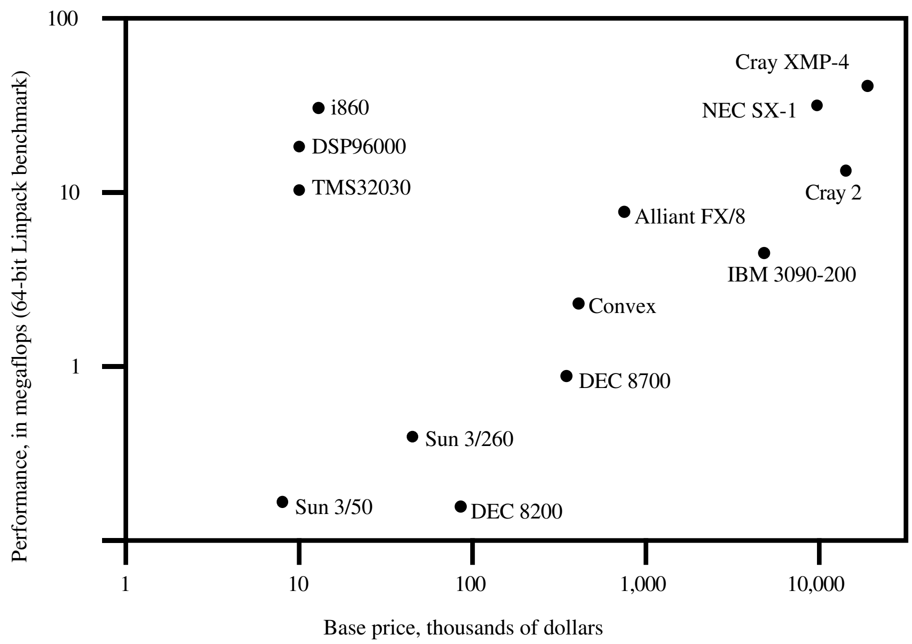

Figure 7.1: Computer Price/Performance Comparison - 1990

puters of the early 1990s will benefit from microprocessors that outperform many large computers of the 1980s. The development of more realistic synthetic movies on personal computers will be helped tremendously.

Chapter 8

Conclusion

This thesis was an exploration of synthetic movies. Synthetic movies are not a new phenomenon, having been present in one form or another since the appearance of the first computers. Until recently, however, synthetic movies had been limited to either entertainment (games) or to relatively expensive, dedicated systems. The entrance of the microprocessor, accompanied by cheap high-density memory and optical storage media, marked the beginning of a new era for synthetic movies.

The digital medium is ideal for the transmission of synthetic movies, allowing a manipulability not found in analog media. Synthetic movies may now be integrated into the user interface and applications of the personal computer. The future is bright.

Bibliography

[Akeley88] K. Akeley, T. Jermoluk, "High Performance Polygon Rendering", ACM Computer Graphics, Vol. 22, No. 4, Aug. 1988, p.239.

[Apgar88] B. Apgar, B. Bersack, A. Mammen, "A Display System for the Stellar Graphics Supercomputer Model GS1000", ACM Computer Graphics, Vol. 22, No. 4, Aug. 1988.

[Backer88] D.S. Backer, "Structures and Interactivity of Media: A Prototype for the Electronic Book," PhD Dissertation, Massachusetts Institute of Technology, June 1988.

[Baumwell88] M. Baumwell, C. Birse, R. Collyer, "NuBus Interrupt Latency (I was a teenage DMA junkie)", Macintosh Technical Note #221, Apple Programmers and Developers Association, Dec. 1988.

[Bove88] V.M. Bove, "Pictoral Applications for range sensing cameras", Proc. SPIE, 1988.

[Bove89] V.M. Bove, "Synthetic Movies Derived from Multi-Dimensional Image Sensors", PhD Dissertation, Massachusetts Institute of Technology, June 1989.

[Brett87] C. Brett, S. Pieper, D. Zeltzer, "Putting It All Together: An Integrated Package for Viewing and Editing 3D Microworlds", Proc. 4th Usenix Computer Graphics Workshop, Cambridge, Massachusetts, Oct. 1987.

[Brøndmo89] H.P. Brøndmo, G. Davenport, "Creating and Viewing the Elastic Charles - A Hypermedia Journal", MIT Media Laboratory, Film/Video Section working paper, July 1989.

[Brush89] Discussion with Dr. George Brush, College of Aeronautics, New York, July 1989.

[DSPatch] —, "The Lamborghini of ADSP-2100 Applications", DSPatch, Analog Devices, No. 11, Spring 1989.

[DuBois89] DuBois, P.,"TransSkel: A Transportable Application Skeleton", TransSkel v2.0 documentation, Wisconsin Regional Primate Research Center, Feb. 1989.

[Feiner82] S. Feiner, D. Salesin, T. Banchoff, "Dial: A Diagrammatic Animation Language", IEEE Computer Graphics, Vol. 2, Num. 7, Sept. 1982, p. 43.

[Foley83]
J.D. Foley, A. Van Dam,Fundamentals of Interactive Computer Graphics, Addison Wesley, Reading, Massachusetts, 1984, p. 3.

[Fortin83] D. Fortin, J.F. Lamy, D. Thalmann, "A Multiple Track Animator System for Motion Synchronization," Proc. ACM SIGGRAPH/SIGART Interdisciplinary Workshop, Toronto, Canada, April 1983, p. 180.

[Ginsberg83] C. Ginsberg, "Human Body Motion as Input to an Animated Graphical Display", S.M. Thesis, Massachusetts Institute of Technology, May 1983.

[Goldfeather89] J. Goldfeather, H. Fuchs, S. Molnar, G. Turk, "Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning", IEEE Computer Graphics, Vol. 9, No. 3, May 1989, p. 20.

[Hecht79] E. Hecht, A. Zajac, Optics, Addison-Wesley, Reading, MA, 1979, p.112.

[Heckbert82] P. Heckbert, "Color Image Quantization for Frame Buffer Display," ACM Computer Graphics, vol. 16, no. 3, pp. 297-305, July 1982.

[Ixion] Information booklet, Ixion Systems, Seattle, Washington.

[Kay77] A. Kay, A. Goldberg, "Personal Dynamic Media", Computer 10(3), March 1977, p. 30.

[Levy84] Steven Levy, Hackers: Heroes of the Computer Revolution, Anchor Press/Doubleday, Garden City, New York, 1984, p.45.

[Linhardt88] P.M. Linhardt, "Integration of Range Images from Multiple Viewpoints into a Particle Database," MSVS Thesis, Massachusetts Institute of Technology, Feb. 1989.

[Lippman80] A. Lippman, "Movie Maps: An Application of the Optical Videodisc to Computer Graphics Laboratory," Proc. ACM SIGGRAPH, 1980.

[Lippman81] A. Lippman, "The Computational Videodisc," IEEE Trans. on Consumer Electronics, vol. CE-27, no. 3, Aug. 1981.

[Lippman87] A. Lippman, W. Bender, "News and Movies in the 50 Megabit Living Room," IEEE Globecom, Tokyo, Japan, Nov. 1987.

[Maxwell83] D. Maxwell, “Graphical Marionette: A Modern-Day Pinocchio”, MSVS Thesis, Massachusetts Institute of Technology, June 1983.

[Mohl81] R. Mohl, “Cognitive Space in the Interactive Movie Map: An Investigation of Spatial Learning in Virtual Environments,” PhD Dissertation, Massachusetts Institute of Technology, Sept. 1981.

[Morris88] L.R. Morris, S.A. Dyer, “Floating Point Signal Processing Chips: The End of the Supercomputer Era ?”, IEEE Micro, Vol. 8, No. 6, Dec. 1988, p. 86.

[Netravali89]
A. Netravali, B. Haskell, <u>Digital Pictures: Representation and Compression</u>, Plenum Press, New York,1989.

[Pentland89] A. Pentland, J. Williams, “Good Vibrations: Modal Dynamics for Graphics and Animations”, ACM Computer Graphics, Vol. 23, No. 3, July 1989, p. 215.

[Platt88] J. Platt, A. Barr, “Constraint Methods for Flexible Models”, ACM Computer Graphics, Vol. 22, No. 4, Aug. 1988, p. 279.

[Pratt78] W.K. Pratt, <u>Digital Image Processing</u>, John Wiley and Sons Inc., New York, 1978.

[Reeves83] W.T. Reeves, “Particle Systems - A Technique for Modeling a Class of Fuzzy Objects”, ACM Trans. on Graphics, vol. 2, no. 2, April 1983, p.91.

[Reynolds82] C.W. Reynolds, “Computer Animation with Scripts and Actors”, ACM Computer Graphics, Vol. 16, No. 3, July 1982, p.289.

[Schreiber86] W.F. Schreiber, <u>Fundamentals of Electronic Imaging Systems</u>, Springer-Verlag, New York, 1986.

[Sturman89] D. Sturman, D. Zeltzer, S. Pieper, “The Use of Constraints in the *bolio* System”, Course Notes, Tutorial #29, ACM SIGGRAPH 1989, Boston, Massachusetts, August 1989.

[Terzopoulos88] D. Terzopoulos, K. Fleischer, “Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture”, ACM Computer Graphics, Vol. 22, No. 4, Aug. 1988, p.269.

[Watlington87] J. Watlington, “A Decoder for Vector Quantized Color Motion Image Sequences”, S.B. Thesis, Massachusetts Institute of Technology, May 1988.

[Watlington88a] J. Watlington, "Movie Player Technical Description", MIT Media Laboratory, Movies of the Future Group, Internal Memo, Fall 1988.

[Watlington88b] J. Watlington, "Color Image Segmentation for Separation of Background Surfaces", MIT 4.906 Final Project Report, Fall 1988.

[XEROX Star] "Designing the Star User Interface", Byte, 1982, p242-282.

[Zeltzer85] D. Zeltzer, "Towards an Integrated View of 3-D Computer Animation", Procs. Graphics Interface '85, Montreal, May 1985.

[Zeltzer88] D. Zeltzer, S. Pieper, D.J. Sturman, "An Integrated Graphical Simulation Platform", Submitted for publication, November 1988.

[Zworykin54] V. K. Zworykin, G. Morton, Television, 2nd Edition, John Wiley & Sons, Inc., New York, 1954, p. 817-818.