

Copyright © 1995, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**VIDEOSTATION: A COMPOSITION PLATFORM
FOR ADVANCED VIDEO SERVICES**

by

Wen-lung Chen

Memorandum No. UCB/ERL M95/7

25 January 1995

COVER PAGE

**VIDEOSTATION: A COMPOSITION PLATFORM
FOR ADVANCED VIDEO SERVICES**

Copyright © 1994

by

Wen-lung Chen

Memorandum No. UCB/ERL M95/7

25 January 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

**VIDEOSTATION: A COMPOSITION PLATFORM
FOR ADVANCED VIDEO SERVICES**

Copyright © 1994

by

Wen-lung Chen

Memorandum No. UCB/ERL M95/7

25 January 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

VideoStation: A Composition platform for Advanced Video Services

by

Wen-lung Chen

Doctor of Philosophy in Engineering—Electrical Engineering and Computer Science

University of California at Berkeley

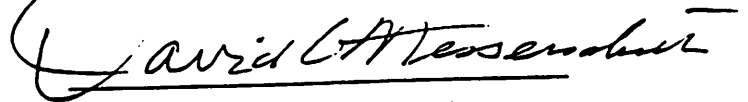
Professor David G. Messerschmitt, Chair

The advancement of the computer technology, broadband networking, and video compression technology makes it possible to support advanced video services such as video telephony, multi-party video conference, tele-seminar, distant learning, video on demand, interactive TV, full motion video games, and virtual reality. In these advanced video services, video materials are generated and/or stored distributively over the network, and shared and retrieved by many users over the network in real-time. To support advanced video services, one important issue is the integration of all pieces of primitive video information from multiple points on the network to produce the final results suitable for personal use. With all the fundamental real-time video supporting technologies today, one missing piece for supporting advance video services is the high-level structure of the video integration that enables efficient use and manipulation of the video elements. This integration process of video materials, called *video compositing*, is the main topic of this thesis.

In this thesis, we propose a *structured video model* to provide a framework to support all kinds of video information compositing. It represents the composited video scene in a hierarchical tree structure while at the same time keeps all the video elements logically separate over the network until the very last stage of video compositing at the users workstation. By doing this, the whole data structure is maintained in a very clean, structural way. All the video elements can also be kept in a simple form that can be most efficient for

data compression, video material sharing and reuse. This makes the network management and the network resource utilization more efficient. The structured video model also provides means to support interactive control for real-time compositing of video information.

Based on the structured video model, we have explored various aspects of the realization of structured video. We have explored both spatial and temporal compositing issues to realize the structured video model. We also designed a real-time compositing platform called VideoStation to demonstrate the feasibility of the structured video model under the limitation of today's supporting technologies. We propose a pipeline architecture for VideoStation to address the memory bandwidth and processing capability bottlenecks in today's memory and processor technology. To explore various video supporting technologies, the VideoStation is implemented in two different approaches — the programmable video signal processing and ASIC design. We have compared the results of these two implementations, and discuss the improvement of video signal processor design to support general real-time video signal processing algorithms efficiently.

A handwritten signature in black ink, reading "David G. Messerschmitt". The signature is fluid and cursive, with a large initial "D" and "M".

David G. Messerschmitt

Committee Chair

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION 1

- 1.1 Motivations 1
- 1.2 Thesis organization 6

CHAPTER 2

ADVANCED VIDEO SERVICES 8

- 2.1 Advanced video services and their requirements 10
 - 2.1.1 Advanced video services 10
 - 2.1.2 Media characteristics 11
- 2.2 Review of the current technology in three major components 13
 - 2.2.1 Multimedia workstation 13
 - 2.2.2 Network 15
 - ATM technology 16
 - Video support of ATM network 18
 - 2.2.3 Multimedia information server 20
 - Requirements 21
 - Technologies in multimedia information server 22
- 2.3 Conclusion 26

CHAPTER 3

STRUCTURED VIDEO MODEL 29

3.1	Objectives	31
3.2	Review of today's standard in multimedia information representation	34
3.3	The definition of a structured video model	39
3.3.1	Video Object	42
	Properties of video object	43
3.3.2	Compositing function	47
3.3.3	Basic compositing functions	49
	Spatial aspect	50
	Temporal Aspect	52
3.3.4	Constraints	56
3.3.5	Events	58
3.3.6	Representation of structured video	59
	Spatial representation	59
	Temporal representation	60
3.4	Advantages of structured video	62
3.5	Conclusion	66

CHAPTER 4

STRUCTURED VIDEO REALIZATION ISSUES

68

4.1	Spatial compositing	70
4.1.1	Anti-aliasing	70
4.1.2	Compositing algorithm	72
4.1.3	Distributed implementation of structured video	73
	Re-structuring of compositing function	76
4.1.4	Possible efficient implementation in spatial compositing	82
4.2	Temporal compositing (synchronization)	86
4.2.1	Synchronization background	87
4.2.2	Clock rate matching	90
	Brief review	90
	Global clock for structured video	92
4.2.3	Many-to-one type synchronization in structured video	93
4.3	Conclusion	96

CHAPTER 5

THE VIDEOSTATION — A PLATFORM FOR VIDEO COMPOSITING 99

- 5.1 Objectives 100
- 5.2 Today's technology for video display 101
 - 5.2.1 Today's memory components 103
 - 5.2.2 Today's display processors 107
- 5.3 Architecture consideration 109
- 5.4 VideoStation design 111
 - 5.4.1 An integrated compositing algorithm 111
 - 5.4.2 Pipeline architecture 113
 - 5.4.3 A modified approach to avoid presorting 116
- 5.5 Conclusion 121

CHAPTER 6

VIDEOSTATION PROTOTYPE 123

- 6.1 System Perspective 124
 - 6.1.1 Graphic Stages 125
 - 6.1.2 Isochronous object stage 126
 - 6.1.3 Control 129
- 6.2 Compositing processor implementation with programmable VSP 130
 - 6.2.1 Programmable VSP overview 130
 - 6.2.2 Philips architecture. 131
 - 6.2.3 Implementation result and discussion 133
- 6.3 Compositing processor implementation with ASIC 137
 - 6.3.1 Design environment 137
 - 6.3.2 Design methodology 137
 - 6.3.3 Simulation, fabrication and testing 140
- 6.4 Conclusion 141

CHAPTER 7**CONCLUSION 144**

- 7.1 Summary of research result 144
- 7.2 Future Direction 148

REFERENCES 150**Appendix A
Video compositing processor implementation with Philip
VSP 156**

- A.1 VSP-8 hardware connection graph(Hard Draw) 156
- A.2 Video compositing algorithm signal flow graph (Soft Draw) 157

**Appendix B
Video compositing processor implementation with ASIC
approach 161**

- B.1 Chip functional diagram 161
- B.2 VCP finite state machine block implementation 164
- B.3 VCP combinational logic block implementation 171
- B.4 VCP scanning register path for testing 172
- B.5 VCP chip pin diagram 173

Acknowledgments

I would like to thank many people who helped me on this Ph.D. work. I could not have finished this dissertation without all of their valuable contributions. First of all, I would like to thank Professor Messerschmitt for his advising, constant support and patience. I would also like to thank Paul Haskell, Shih-Fu Chang, Louie Yun, Chung Sheng Li, Ho-Ping Tseng, Chuen-Chieh Lee and many other friends for their valuable discussions. Paul Haskell also wrote a software emulator for the VideoStation compositing model. Louie Yun contributed a lot of his valuable time to design the VideoStation board using the ASIC chip.

I appreciate the assistance of Philips and Signetics in providing access to the VSP processor and its software tools in the VideoStation prototype design. Kees Vissers and Arthur van Roermund of Philips Research Laboratories have been generous with their time in assisting the project. I also appreciate the Asahi Chemical Co. for fabricating the video compositing chip that I designed. I also thank Mr. Arakawa and his colleague for their valuable time for communicating with us and doing the chip fabrication and testing.

I would like to thank also to my colleagues at Pacific Bell, Yuet Lee, Hon So, and Tom Soon for their support and understanding. Hon So, especially gives me his full support to help me finish this dissertation writing.

Finally I would like to thank my wife Esther and my parents for their constant help and considerate. Without their help, I would not be able to finish this. Thanks!

Wen-lung Chen

CHAPTER 1

INTRODUCTION

1.1 Motivations

Today's technology has finally come to a point that multimedia applications are possible both on the desktop environment and over the network. The technologies for generating, manipulating, and distributing the tremendous volume of data involved in multimedia information are becoming available. Typical examples of multimedia applications are video telephony, multi-party video conferencing, tele-seminars, distant learning, video on demand, interactive TV, full motion video games, virtual reality, etc. Since the definition of 'multimedia' is still somewhat vague, we will use the term *advanced video services* to represent the services we have in mind. The ultimate goal of advanced video services is to make it easy to present, share, manipulate, and reuse various kind of media information, mainly full motion video, in an efficient way among all users over the network. The triggering technologies for this advanced video service consists of three technologies: computers , broadband networks, and data compression.

In computer technology, the increasing capacity of storage devices and increasing processing capability make it possible to store/retrieve and manipulate multimedia information more easily. CPU processing capability and memory device capacity increase every year, thanks mainly to the advances in very large scale integrated (VLSI) technology. In addition to the advance of VLSI technology, parallel architectures are also important, especially whenever the processing speed of the CPU or the bandwidth of the memory device cannot keep up with the need for supporting real-time full motion video. Examples are the design in video read/write access memory (VRAM), and the redundant array of inexpensive disks (RAID), which use special architectures to provide the bandwidth that silicon devices or conventional mechanical magnetic disks cannot achieve.

In broadband network technology, the asynchronous transfer mode¹ (ATM) network has been adopted by CCITT as the standard for broadband integrated services digital network (B-ISDN). Most of today's network systems suffer from many disadvantages in supporting multimedia due to the tremendous bandwidth requirement and the variety of traffic involved in multimedia. Advanced video services, by our definition, will include various kinds of media. Each different medium may have different requirements when it is transmitted over networks. For example, full motion video has a higher requirement for network bandwidth and delay jitter. However, it may be able to tolerate some errors in transport. Transmission of computer data requires very reliable transport, but is not as critical in the delay requirement. Audio streams, on the other hand, are very sensitive to network delays. All these different media impose different requirements. Today's network either does not support sufficient bandwidth for real-time full motion video, or is not flexible enough to support different kinds of traffic. The ATM network is basically a service independent network. Its virtual circuit concept makes it possible to support any kind of media transport or a combination of them in a quite efficient way. This kind of network will be most useful for transporting all the video, audio, graphics, etc. The ATM network

¹ Transfer mode is a term used in telecommunications to describe multiplex and switching techniques

also provides a very high bandwidth. The currently available ATM network supports traffic of 155 Mbps (OC-3 rate) or even up to 600 Mbps (OC-12 rate) on one port. This is equivalent to a bandwidth of about 100 to 400 video streams with MPEG¹ I compression technology of the average bit rate from 1.2 Mbps to 1.5 Mbps. The huge amount of bandwidth capability and the flexibility for supporting multimedia make it the choice for a future B-ISDN standard.

The third important technology is video compression. The uncompressed video information takes a huge amount of bandwidth to transmit. A CCIR 601² video takes a bandwidth of 216 or 270 Mbps for a word length of 8 or 10 bits, respectively. With no compression, a single video stream cannot even fit into one 155 Mb/s ATM port. With compression, however, the video stream can be reduced to a much lower rate for transmission through the network. There are currently various compression standards used for various applications. For low-quality video conferencing, the H.261³ standard can compress the video down to $p \times 64$ Kbps. The video quality is barely acceptable when p is low. MPEG I can be compressed down to 1.2 - 1.5 Mbps with VHS quality. It is used mainly for application using storage devices. The algorithm is designed such that fast forward or reverse is possible. MPEG II is aimed at entertainment, or even higher quality applications. The data rate is varying from 4 - 15 Mbps for a format conforming to CCIR 601 standard, to 60 Mbps for high definition TV (HDTV).

A diagram of the advanced video service environment is given in Figure 1-1, illustrating how the different technologies work together. In this figure, all users are connected with a broadband ATM network for sharing multimedia material. Any information, including video, audio, graphics, still images, and text, are transmitted through the ATM network in a very flexible manner. At the user location, basic features would include video

¹ ISO/IEC/JTC1/SC29/WG11 MPEG working group

² SMPTE recommendations 601, a format standard for studio quality video.

³ ITU-T/CITT Recommendation H.261, "Video Codec for Audiovisual services at $p \times 64$ Kbps."

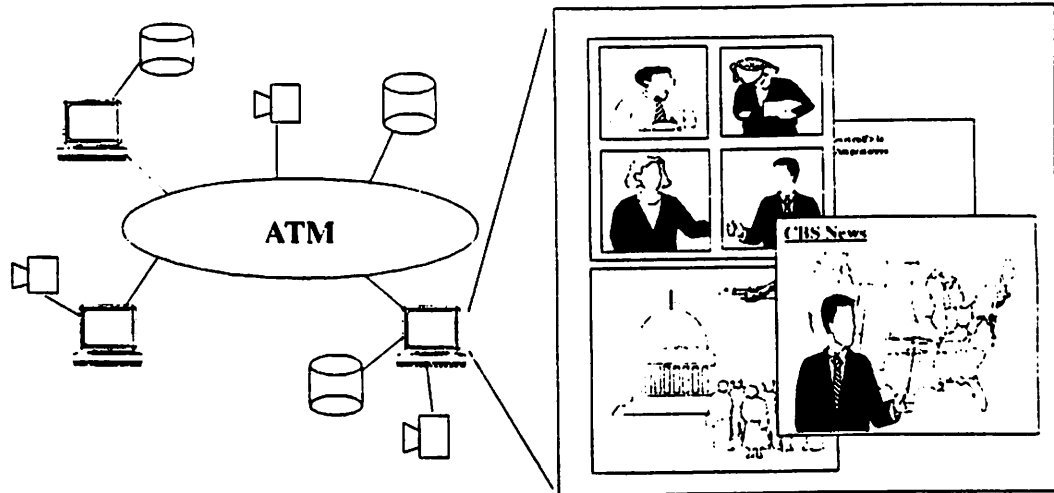


Figure 1-1. Advanced video service environment.

compression/decompression, shooting real-time video sequences with a video camera, retrieving/saving multimedia information from/into a local storage device on the network, composing multimedia information, and transmitting information over the network. Several kind of devices can be provided on the network for serving all the users, such as a large multimedia information server accessible by all users, video bridging or transcoding for providing video stream merging and format conversion.

Even though this kind of service has been discussed for quite a long time, it was never as promising as it is today. The cost of equipment to support them is also decreasing, which helps to make it practical. Today, people are proposing for a set top box for interactive TV with a price around \$300. Such a set top box will include features such as network interface, video decompression, interactive control capability, and remote control capability. Among all the supporting blocks for multimedia, one single piece whose cost has not come down is the production of the contents - mainly video materials. This is mainly due to the difficulties in accessing and reusing existing video materials. Today, we can very easily edit text and graphics information with even a very primitive personal computer. Any single graphic object can be used over and over again in different documents. However, it is usually more difficult to reuse and compose video materials, especially for

casual users. Most of today's video composing activities are performed in video studios with very expensive equipment.

The reason for this difficulty in reusing video material actually lies in the structure of the video information itself. Unlike computer graphics which start from structural manner for efficient presentation, today's video technology develops from quite a different origin. The video technology today basically puts all the information in one primitive video stream with a rectangular screen shape. This single video stream may include many different pieces of information tightly integrated. A more structured format is needed in order to utilize existing video material more easily. The need for a structured format actually already appears in the audio format in Japanese "Karaoke" equipment. In Karaoke, the sound of the original singer is separated from the background music such that we can choose to either hear the original singer's voice or substitute our own voice for his voice. This is a good example of how audio information needs to be more structured to have flexibility in the presentation. A similar situation happens in video. Without any structural information in this primitive video stream, it is usually very hard to retrieve each individual piece of information for reuse. The whole video stream consisting of many pieces of information has less chance to be reused than each individual piece of information because it is less generic for reuse.

The objective of this thesis is to study the structural approach for multimedia information, mainly video information. We will try to use a structural model to make the presentation of video information more efficient, in terms of either representation, transmission, and ease of reuse. With this objective, we propose a structured video model, which we believe will achieve these goals. In addition to better video material reuse, arranging the video material in a structural manner also brings other advantages such as better compression, more efficient resource utilization. Starting from this model, we also look into the current technology that can actually implement the structured video model. Three major components are required to support structured video. They are the workstation, used for

compositing and presenting structured video information; the storage system, used for storing video information; and the network, used for transporting video information. This thesis will focus on the efficient implementation for the video display subsystem of a workstation, which we call a VideoStation. We propose a pipeline architecture that can be implemented with current technology to support real-time compositing and presentation of structured video. We also implement the VideoStation with both an application specific integrated circuit (ASIC) and a programmable video signal processing approach to get more insight into the current technology.

1.2 Thesis organization

The organization of this thesis is as follows. In the second chapter, we describe in more detail the advanced video services that we have just outlined, and study their requirements in terms of bandwidth, processing, quality, etc. We briefly review today's technologies in regard to the three major components for the support of advanced video services — the workstation, the storage, and the network. We also discuss the relationship of the three components to each other for efficient implementation.

In the third chapter, we describe the structured video model that we propose to facilitate efficient video information presentation and reuse. Some major components in the model, such as video objects, compositing functions, constraints, and events are defined. The compositing properties of these components are studied in detail, in both spatial and temporal compositing aspects. Then we discuss the advantage and disadvantage of this structured video model.

In the fourth chapter, we describe the implementation issues of the structured video model. In the spatial compositing aspect, we discuss the anti-aliasing issue, and derive the algorithms for all the compositing functions of interest. We also study the properties of compositing functions to enable restructuring and other ways for more efficient imple-

mentation. In the temporal compositing aspect, we study the synchronization issues. This includes clock rate matching and synchronization among multiple video objects.

In the fifth chapter, we describe the VideoStation that we propose for real-time compositing and presentation for structured video. We start with a review of today's display technology, and describe the bottleneck in this compositing platform. Then we propose our architecture — the pipeline architecture — to enable real-time compositing. We also describe the design of the VideoStation, and some optimization issues.

In the sixth chapter, we describe our prototype of VideoStation. We use two quite different approaches for this implementation. In the first case, we use a fully dedicated approach to show the efficiency of the VideoStation architecture. In the second case, we use a programmable video signal processing approach to get a better understanding of today's video supporting technology. The programmable approach also enables faster implementation and easier debugging. We also discuss the advantages and disadvantages of the architecture of the VSP chips.

In the last chapter, we summarize the results, and describe some important issues in this field that will require further study in the future.

CHAPTER 2

ADVANCED VIDEO SERVICES

As we described in chapter 1, the technologies in computer, broadband network, and data compression together bring up possibilities to support *advanced video services* such as video telephony, multi-party video conference, tele-seminar, distant learning, video on demand, interactive TV, full motion video games, virtual reality, etc. In our definition, advanced video services are video services that provide people with an easy way to retrieve, share, use, and present various kinds of information over the network. The media that we consider are mainly full motion video sequences, but also includes audio, graphics, images, and text. The ultimate goal is to allow people at different locations to easily interact and collaborate among one other.

When advanced video services are commonly used, it can significantly change the way people interact. Full motion video not only lets people present their ideas more flexibly, but also make people feel like the person were in front of them even though they are not. This is most important for human interaction services such as video telephony, video conferencing, and distance learning.

There are three major components to advanced video services — the workstation that processes and presents the information, the network that transports the information, and the storage device that stores the information. Designing these components is a challenge to support all kinds of media, especially full-motion video sequences. On top of these components, an even more important issue is how various kinds of media are organized in a most efficient way to ease the information transport, process and presentation. This includes specifying basic components of information materials and to define the data structure of the basic components best fit for all advanced video services. This information structure issue is a major part of this thesis, and we will postpone the discussion about this until later chapters.

When all these components are put together on the network, there are also issues regarding distributed processing. For example, whether should we store a piece of information locally or remotely to make the overall process and transport most efficient; how the processing is allocated distributively such that the resources on the network are efficiently used, etc. These aspects allow further optimization when we address each specific service. To do this design, we need a better understanding of the characteristic of the information that we want to support.

In this chapter, we give a general review of the advanced video services and the technologies in the three major components. In section 2.1, we list some advanced video services, and describe the characteristics of the media involved in these services. Then we briefly review the technologies in each of the three components to support advanced video services in section 2.2.

2.1 Advanced video services and their requirements

2.1.1 Advanced video services

The advanced video services are categorized into two classes from the users environment as listed in Table 2-1 — the business applications and the home applications. Typical

TABLE 2-1: Advanced video services.

Type of application	Application
Business	Video telephony/Multi-party video conference
	Tele-seminar/Corporate training/Distant learning
	Open/share workspace/Remote Collaboration
	Video mail
Home	Video on demand/interactive TV
	Bulletin board service/Home shopping service
	On-line multimedia magazine
	Full motion video game
	Virtual reality/Virtual trip

business applications include video telephony, multi-party video conferencing, open/share workspace, tele-seminar, corporate training, distant learning, video mail, etc. Video telephony and multi-party video conferencing allow people to interact through the network without moving to the same place. Tele-seminar, corporate training, and distant learning allow people to attend a seminar or a class in either realtime or non-realtime manner. In realtime, people can attend the seminar from different places; have two way communications between the audience and the speaker; and also can have subgroup discussions. In a non-realtime manner, people access the seminar material pre-recorded in some information server, and simultaneously they may ask questions through a connection to the speaker, or have discussions with each other. The class material may involve video, slides, or some interactive exercises, with all various media integrated in a graceful way. The

open/share workspace and remote collaboration are applications that allow people to work on the same project from different sites. It provides a virtual workspace that has the project material accessible to all the participating people. Any changes of the project material by any participant will be reflected to all the rest of people in real-time. Video mail is an extension of the current electronic mail and voice mail. It allows people to compose, store, and distribute video message easily.

In a home user environment, video on demand allow people to order a video directly from the network instead of walking into the video tape rental store, while allows people to still have VCR-like control feature, such as pause, fast forward, etc., over the displayed video program. It also allows people to start watching the movie any time without worrying about the movie schedule. The bulletin board and the home shopping service allow people to browse through the catalogs of many stores in real time, do research about the quality and price of some merchandize, and finally place the order. The virtual reality and virtual trip use 3-D video to enable people to tour a place with a real sense of personal participation.

2.1.2 Media characteristics

Various kinds of media are involved in the applications described above — video, audio, still images, binary data, interactive data, and their combination. The characteristics of all these data types vary and cover the full spectrum of possibilities. Technically, it is quite a challenge to support all kinds of media and integrate them in a graceful way. The characteristics that are of interest are the data rate, the total data volume, and the burstiness of the traffic. Burstiness of the traffic is defined as the ratio between the maximum and the average information rate. There are also characteristics that are application-dependent or implementation-dependent, e.g., the sensitivity to the transport delay or delay jitter. For example, some applications require intermediate response with little delay, while others allow a long delay. The acceptable delay is determined by applications, not by the media

involved. Delay jitter, the variation of a delay, can usually be maintained to be under some limit at the price of some buffering scheme and a longer delay.

All media can be categorized into two classes — non-realtime and realtime. The non-realtime class includes ASCII files, binary data, still images, etc. The non-realtime media has no hard time limit on when it should be available after we issue the access request. Usually, we simply want to make the access as fast as possible for this class of media. Since there is no hard-time limit on the access, the data rate and the burstiness vary, depending on the application and the implementation. Typically, the burstiness of this class of media is high because of the design of the storage device. For example, the accesses of data from hard disk are always by a whole block. This causes the data traffic to be more bursty. The total data volume of this class of media also varies. They can be a small ASCII file of several bytes, or a very large image file of several mega bytes. Today's computer technology primarily focuses on this class of media, and can support them quite well.

The realtime class includes data types that are usually accessed in a rhythmic way instead of being accessed abruptly in as a short time as possible. They are access periodically with a certain data rate, and have a hard-time limit on when it is available. If some

TABLE 2-2: Real-time media and their data rate.

Types of media	Data rate	Burstiness
Voice	32 kbit/s	2
Interactive data	1-100 kbit/s	10
High quality video telephony	0.2-2 Mbit/s	5
Standard quality video	1.5-15 Mbit/s	2-3
High definition TV	15-150 Mbit/s	1-2

data are available after this hard-time limit, it becomes useless. The transport delay jitter therefore has important impact on the media in this class. The data types included in this

class are video, audio, interactive data, etc. The data rate and burstiness vary depending on the media, as shown in Table 2-2. The typical data volume of this type of media is large.

In essence, the media involved in advanced video services have characteristics that cover the full spectrum of possibilities. To implement advanced video services, we need to support both real-time or non real-time data types. We need to handle the data rate from several kbits/sec up to hundreds of Mbits/sec. The total data volume can be from bytes up to gigabytes. The burstiness also varies a lot. The components that store, transport, and display these media need to handle this heterogeneity in an efficient way.

2.2 Review of the current technology in three major components

In this section, we review today's technology in terms of presenting, transporting and storing the media for advanced video services. Today's computer technology starting from non-realtime data environment has no problem with the non-realtime media. The challenge occurs when the data rate grows higher and when the data volume grows very large. This is mainly introduced by full-motion video. Therefore we will focus on the support of video streams when we discuss the three components. We will also briefly mention how the current technology handles the heterogeneity of media types.

2.2.1 Multimedia workstation

One major component for advanced video services is the multimedia workstation that users directly reach. Through this workstation, a user can access, view, and compose a multimedia material. The function of this multimedia workstation is to receive and decode various kind of media either from a local storage device or from the network, combine the media according to some specified data structure, and present them on the display screen.

There are three components in the multimedia workstation — the media interface that receives and decodes the media, the human interface that eases the manipulation of various media, and the compositing display that combines the media. The media interface relies on existing standards. For full motion video, there are several video compression

formats available for various applications. H.261, motion JPEG, MPEG 1, and MPEG 2 are some examples. H.261 is a constant bit rate video compression standard for video telephony application. Motion JPEG (MJPEG) is a video extension of the JPEG still image standard. It is preferable to some applications that need to support video editing features because of its intra-frame compression algorithm. MPEG 1 is originally proposed for video storage related applications with roughly VHS video quality. MPEG 2 is a higher quality proposed for video entertainment quality. It includes video resolution from NTSC video quality to high definition TV. No matter what standard is used, the media interface should provide appropriate decompression according to the nature of the media.

The human interface is important for multimedia presentation because of the complexity involved. Traditional mono-media applications, e.g., the voice phone, have a very clearly defined media and function and are easy to handle. When the application involves multiple media, however, there are all kinds of combinations and possibilities. The complexity will go beyond people's ability to handle if a proper human interface is not available. One basic principle of designing the human interface is to use the analogy to conventional way of doing things. For example, H. Kamata et. al. proposed in their MONSTER multimedia system a human interface based on the conventional paper document system that people are most familiar today[29]. In addition, the technology in today's computer system should also be incorporated, e.g., the hyper-linking technique that allows easy access to related material by specifying their relationships through a link. The authoring capability allowing users to easily compose a multimedia material is also important. Currently, commercial multimedia authoring systems are already available on the market.¹

The compositing display system that combines the media and shows the material on the display is a major bottleneck in a multimedia workstation. Most of today's worksta-

¹. For example, Gain Momentum from Gain Technology Inc.

tions cannot support full-motion video. With special hardware, some workstation can support very limited full-motion capabilities¹. Most of the existing special hardware of workstation supporting full-motion video simply replaces the original frame buffer with a faster one such that one (or two) rectangular full-motion video window can show on the display. The big disadvantage of this approach is the scalability — they cannot be expanded to accommodate more video streams. In advanced video services, multiple simultaneous video streams are essential. One solution to this is to use a video bridge to merge all the video streams into one stream in advance, such that the user need only to receive and display one video stream. This approach, however, has a very big limitation on the use and manipulation of each individual stream of video material to get the best result of presentation.

In advanced video services, we want to support multiple video streams and manipulate each individual one independently. To maximize the flexibility of video composition and presentation in advanced video services, we also want the video streams to be arbitrarily shaped instead of rectangular such that the video streams can be easily put together and composed into a new video scene. To do this, the compositing display system need to be greatly enhanced to support more elaborated compositing operations for arbitrarily shaped video streams. One goal of this thesis is to define structured video model based on arbitrarily shaped video streams to ease the use and composition of video materials, and to demonstrate the implementation of compositing display system for the structured video. All these aspects will be discussed in detail in later chapters.

2.2.2 Network

In this section, we will review the network technology, mainly *asynchronous transfer mode* (ATM) technology, which supports the requirements of advanced video services. To support advanced video services, the network needs to handle any kind of information.

¹ For example, XVideo card from Parallax Inc. for Sparc stations and Video Blaster card from Creative Lab for PCs.

such as voice, data, image, text, and video, in an integrated manner. The network not only needs to support the required bandwidth of the media, but also to support the heterogeneity of the services — bursty and continuous traffic, interactive and distributive services, connection-oriented and connectionless, point-to-point and multipoint-to-multipoint connections — all be integrated gracefully. The use of highly reliable fiber systems can provide the necessary high bandwidth for advanced video services. To support heterogeneous traffic, the transfer mode of the network need to be considered.

Transfer mode, according to CCITT, is the technique used for transmission, multiplexing, and switching data information over the communication network. To support heterogeneous traffic, the transfer mode must be very flexible to transport a wide range of natural bit rates and cope with services that have fluctuating characteristics in time. The asynchronous transfer mode (ATM)[9][12] technology has been recognized to be the best one to support the heterogeneous requirement. In this section, we will emphasize on the ATM network technology.

2.2.2.1 ATM technology

ATM is a connection oriented packet switching technology that uses packet switching with minimal functionality in the network to allow fast and efficient processing of the switched information. The term “asynchronous” is used because it allows asynchronous operations between the sender and the receiver using independent clocks. In ATM, user information is transmitted between communicating entities using fixed-sized packets (called cells) of 53 bytes. By choosing such a small and fixed cell size, all constant bit rate(CBR) and/or variable bit rate(VBR) services can be easily multiplexed together to share the network resources. Small cell size also allows us to reduce the buffer size in the switch to limit the buffer queuing delays. A low queuing delay is necessary to satisfy the requirement of real time services. The buffer management and switching fabric design are also simplified with this small fixed cell size. There is no link-by-link error protection or flow control in the ATM network. All the error detection and flow control mechanisms,

when necessary, are pushed up to the end-to-end transport layer to keep the switch system simple and fast. With highly reliable fiber transmission, error protection can be omitted without causing much problem.

Today's ATM networks can provide the OC-3 rate of 155 Mbps, and most possibly up to the OC1-12 rate of 622 Mbps soon. With this rate, the ATM network supports multiple standard quality video streams or even high definition TV quality video streams easily with all other audio, data, images. Because of this, ATM has been adopted by CCITT as the standard for the future broadband integrated services digital network (B-ISDN).

Since an ATM network is a service independent network platform, it can be tuned to support specific characteristics of a connection on the network. Similar to the OSI layered reference model, an ATM protocol reference model is also defined by CCITT. Among all the layers in the reference model, the ATM adaptation layer (AAL) is defined to adapt various service information to the ATM streams to provide the required quality of service (QoS) to services. The basic function of AAL includes segmenting/reassembling the information stream into/from cells, maintaining the time/clock recovery information, compensating variable delay and loss cells, etc.

Four AALs are currently defined by CCITT for four different classes of service. The services are classified using three parameters: the timing relationship (realtime or non-realtime), the bit rate (variable or constant bit rate), and the connection mode (connectionless or connection-oriented.) The four AALs are:

- AAL1: constant bit-rate connection-oriented services with timing relation between source and destination. Examples of this class of services are the emulation of constant bit rate channels such as 1.5 Mbits T-1 connections and constant bit-rate video coded connection (e.g., k x 64k).
- AAL2: variable bit-rate connection-oriented services with timing relation between source and destination. Variable bit-rate video coded connection is an example.

- AAL3/4: variable bit-rate connection-oriented/connectionless data transfer services that are sensitive to loss, but not to delay. These are defined for typical data communication.
- AAL5: variable bit-rate connectionless data transfer services. This layer is defined to offer better error detection and less overhead than AAL3/4. The typical service of this class is also data communication.

Although ATM has already been deployed in public networks, there are still issues that need to be resolved to really achieve the goal of serving as a high performance service independent platform. The major issue is currently in the traffic management — how to monitor the traffic, to perform the admission control and traffic policing to avoid the congestion in the network and guarantee the quality of the services for each connection. The other issue is the routing — to select a best path in the network and maintain the correctness, simplicity, robustness, stability, fairness, and reliability.

2.2.2.2 Video support of ATM network

In terms of supporting full motion video over ATM, the major issues are the clock recovery/synchronization and error detection/correction/concealment. The clock recovery deals with the different clock frequency at the video source and the destination. As the name implies, ATM is asynchronous, meaning that the clock at different nodes can be independent. This would cause the receiver to expect data at a faster or slower rate than it is being transmitted. If the receiver runs at a faster clock rate, the buffer may be underflow and the cells are thought to be lost. If the receiver runs at a slower clock rate, then the buffer may be overflow and the cells are discarded. In either case, the quality of video is degraded. To solve this problem, some timing information stamped in the data stream is needed. Using the time stamp, we can recover a clock at the receiver to tightly follow the clock at the transmitter. In this way, the data consumption rate at the receiver and the data generation rate at the transmitter will be roughly the same.

Besides using the recovered clock to avoid buffer underflow/overflow, the recovered clock is also used to generate the NTSC the video signal after the received data stream is uncompressed. The current NTSC specification for video has a very tight requirement on the drift of the recovered clock. Consequently, MPEG II specifies the acceptable clock drift to be under 0.075 Hz/sec. It was shown by simulation by Divicom Inc.¹ that the recovered clock has a clock deviation as large as 10 Hz/sec for a 40-tap digital phase lock loop filter, and 70 Hz/sec for a 20-tap filter, assuming a uniform i.i.d. jitter with a peak-to-peak jitter of 1 msec. This is still two order of magnitude lower than the MPEG requirement. How to find an economical way for clock recovery is currently still a research topic.

Synchronization among multiple video sources at the same receiver is also an issue to be resolved. Using the clock recovery technique, the receiver can only keep track of one video source. How to make all the video sources synchronized is yet to be solved. In this thesis, we will discuss this issue in more detail later in chapter 4.

Error detection/correction and concealment are also important issues for video traffic. Over ATM, the bit error rate is usually low (less than 10^{-8}) due to the use of fiber system, and the link-by-link error correction can be omitted without much impact on the performance. However, some error correction on a per cell basis at the AAL layer is still needed to satisfy the requirement of various video services. A similar situation applies to the cell loss ratio. Typical cell loss rate for a single ATM switch range from 10^{-8} to 10^{-11} . Table 2-3 [10] and Table 2-3 [11] show some recommendations for the requirement on bit error

TABLE 2-3: Recommended BER values for some video applications.

Application	Bit rate	BER [†]	BER [*]
Videophone	2 Mbps	3×10^{-11}	1.3×10^{-6}
Videoconference	5 Mbps	10^{-11}	1.8×10^{-6}
TV Distribution	20-50 Mbps	3×10^{-13}	6×10^{-7}

[†] "Comments on Clock Recovery in the Presence of Jitter", 1993 ISO/MPEG meeting paper.

TABLE 2-3: Recommended BER values for some video applications.

Application	Bit rate	BER [†]	BER [*]
MPEG 1	1.5 Mbps	4×10^{-11}	2.5×10^{-6}
MPEG 2	10 Mbps	6×10^{-12}	1.5×10^{-6}

[†] Without error handling in AAL.

^{*} Single-bit error correction on cell basis and additional cell loss correction in AAL.

TABLE 2-4: Recommended CLR values for some video applications.

Application	Bit rate	Cell loss ratio [†]	Cell loss ratio [*]
Videophone	2 Mbps	10^{-8}	8×10^{-6}
Videoconference	5 Mbps	4×10^{-9}	5×10^{-6}
TV Distribution	20-50 Mbps	10^{-10}	8×10^{-7}
MPEG 1	1.5 Mbps	10^{-8}	9.5×10^{-6}
MPEG 2	10 Mbps	2×10^{-9}	4×10^{-6}

[†] Without error handling in AAL.

^{*} Single-bit error correction on cell basis and additional cell loss correction in AAL.

rates and cell loss rates of various video applications. After applying the bit error correction and cell loss correction, some error concealment technique is used to make the uncorrected bit errors or cell losses less noticeable. Retransmission is usually not possible for video application over ATM because of the large bandwidth of the video traffics and the delay of the transport.

2.2.3 Multimedia information server

The multimedia information server is a very important component for advanced video services. Most of the advanced video services that we described in section 2.1.1, such as video mail, video on demand, bulletin board service, and on-line multimedia magazine, need a large information server to make them realizable. In this section, we discuss the requirements on a server to support advanced video services. We also review the important issues in the realization of such a multimedia information server.

2.2.3.1 Requirements

To support advanced video services, the information server has quite different requirements from a conventional computer file server. These requirements are mainly in three aspects: the storage capacity, the access bandwidth, and the media access mode.

Capacity requirement

The storage capacity needed for full motion video sequences is usually very large. An hour of MPEG 1 video information can easily consume about 1 GBytes of storage space. Typical data storage of a two-hour long video material in various applications is shown in

TABLE 2-5: Data volume of various video applications.

Application	Bit rate	Data volume of 2 hours of video
Videophone	2Mbps	1.8 GBytes
Videoconference	5Mbps	4.5 GBytes
TV Distribution	20 - 50 Mbps	18 GBytes - 45 GBytes
MPEG 1	1.5 Mbps	1.35 GBytes
MPEG 2	10 Mbps	9 GBytes

Table 2-5. Depending on the application, an information server may need to store several hundred pieces of video information. This makes a total capacity requirement on an order of several hundred gigabytes to several terabytes. This is extremely high as compared with today's data file servers.

Bandwidth requirement

The bandwidth requirement on the video server is also usually high. Table 2-5 also shows the access bandwidth requirement on each video stream. As a server, it is necessary to support multiple simultaneous accesses. For example, in video on demand application, a video server may need to support up to, say, 500 homes. This make the total access bandwidth up to 94 MBytes/sec sustained rate, assuming MPEG 1 video streams are used.

Media access requirement

The media access of a multimedia server is quite different from the conventional data server. First, it needs to support access of multiple media. Media such as text, graphics, images, audio, and video streams are all stored in one server. Secondly, it need to support real-time access for all media. To do this, a server needs to record both the incoming data stream and the incoming traffic pattern, then replicate the same traffic pattern when the piece of information is retrieved. Without this traffic pattern replication capability, the retrieved information may be useless for realtime application. This is especially important with variable bit rate video streams.

Note that not only the accesses of video/audio streams have the real-time constraint, the access of text and graphics/images also need to satisfy similar real-time constraint such that the text and graphics are available at a time synchronized with the accompanying video and audio. This kind of I/O regulation/scheduling capability is one important feature of realtime servers. In a conventional data file server, the server reads out a file as fast as possible when the file access command is received. In a multimedia server, however, the server only reads out the file (or video/audio streams) at a certain time with a specified rate, even though it can do it earlier or faster. All these real-time access requirements make the multimedia server quite different from the conventional data file server.

To meet all three requirements described above is a major challenge in multimedia information server design. Typically the requirements are beyond the capability of any storage device available today. Under this situation, multiple devices running in parallel are necessary to satisfy both the space and the bandwidth requirements. In the next section, we will review the architecture and the major technologies involved in a multimedia server design.

2.2.3.2 Technologies in multimedia information server

A typical multimedia information server has an architecture shown in Fig. 2-1. In this architecture, there are four major components: the storage device, the server processor, the

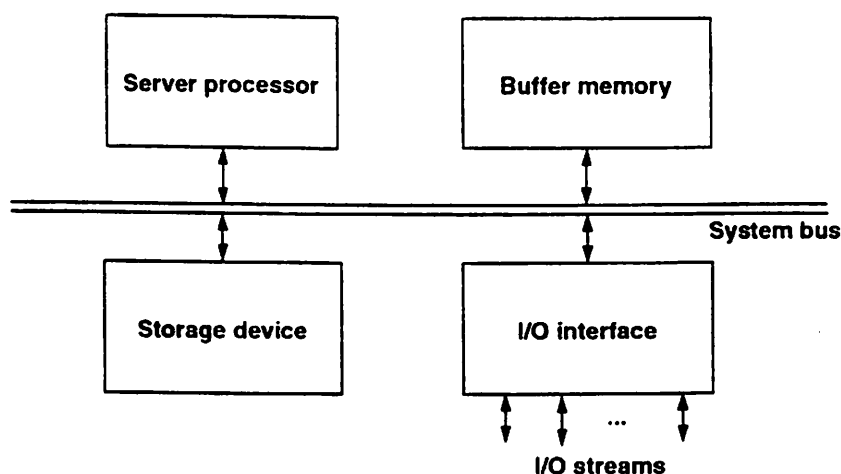


Fig. 2-1. Typical multimedia server architecture.

buffer memory, and the I/O interface. The storage devices are the memory storage devices such as magnetic disk, tape, and semiconductor memory, which keep the data. The server processor is in charge of the receiving I/O access requests from outside the server, performs real-time scheduling, and issue commands to ship data between the I/O interface, the buffer memory, and the storage device. The I/O interface is the data interface between the server and the outside world. The buffer memory is used by the server processor to regulate the I/O traffic from more bursty one at the storage device into a smooth one, or vice versa, at the I/O interface and the outside world. The buffer memory can also be used to optimize the hard disk performance by keeping the disk access block as large as possible to reduce the disk heads movements. Among these components, the storage device and the server processor are more involved, and we will discuss about them in more detail.

Storage devices

The first issue in the design of multimedia information server is to have a storage device that satisfies the bandwidth and the capacity requirements described previously. There are various storage devices available today with different capabilities, as shown

in Table 2-6. Among these devices, semiconductor memory devices provide the fastest

TABLE 2-6: Typical parameters of various storage devices available today.

Storage Device	Access time	Bandwidth	Capacity per unit	Cost per MByte
SRAM	5-50 ns		1-4 Mbit/chip	\$80-120
DRAM	50-100 ns		4-16 Mbit/chip	\$32-40
Magnetic hard drive	20 ms	2.5-4 MBytes/sec	20 Mbytes to several GBytes [†]	\$1-7
Magnetic tape drive*		3 Mbytes/sec	2400 GBytes	\$0.25
Optical drive (R/W)	19-90 ms	5-15 Mbit/sec	Up to 1.3 GBytes	\$5-8
Optical drive (WORM)	40 - 90	4-8 Mbit/sec	600-940 MBytes	\$6

[†] Disk drives range from 1.3" to 10".

* IBM 3480/90 tape drive.

access time and the highest data transfer throughput. However, they are small in capacity and expensive in cost. The magnetic hard drive is slower than the semi-conductor memory device, but with a larger space and lower cost. The magnetic tape drive provides the largest storage space and least cost per megabyte. However, it is also the slowest.

Note that there are tradeoffs between the bandwidth capability and the capacity/cost per megabyte. A faster storage device usually has a smaller capacity and a higher cost. Under the constraint of these tradeoffs between the bandwidth, capacity, and cost, memory caching hierarchy provides a way to reduce the overall cost while also satisfies the bandwidth and capacity requirement. This is done by placing the less frequently accessed data in a lower speed but larger space magnetic tape drive, and swaps the more frequently accessed material from the tape drive into faster devices such as magnetic disk and semiconductor memory buffer. Once the data are placed in the faster devices, they can be repetitively used until they are swapped out of that device when other data are swapped in. In

this way, most of the I/O access will only go to the faster device, and therefore reduce the actual I/O access to the low speed devices. This kind of realization is typical in today's video server for video-on-demand applications.

Among all the storage devices in the memory hierarchy for a video server, magnetic hard disks are the most important to provide multiple real-time video accesses. Note that magnetic tape drive can only support sequential access due to its tape storage. This is not feasible for applications requiring very fast response with very small delay. The sequential access nature also prohibits multiple simultaneous accesses to the same video material. Magnetic hard disks are more appropriate to support simultaneous real-time video accesses in terms of the bandwidth, capacity, random access capability, and the cost.

To provide enough bandwidth and capacity with hard disks, an array of parallel disks is usually used to expand the bandwidth and capacity. The disadvantage is that a disk array usually has a much higher failure rate than each individual disk. The failure of any individual disk will cause the whole array system to fail. *Redundant array of inexpensive disk (RAID)* [13][14] technique is very useful to make the disk array more reliable. The idea is to provide redundancy in the disk array system such that a failure in a disk can be recovered from the information on the rest of the disks. In this way, the RAID technique can provide a large capacity storage device using arrays of inexpensive disks while also provide the reliability higher than a single large and expensive disk.

Server processor

As described previously, the server should support realtime access, record and replicate certain I/O pattern. These are all accomplished by the server processor. In essence, the function of the server processor is to provide real-time control to ship the information around between the I/O interface, buffer memory, and the storage device at a correct time. To do this, a real-time operating system running on the server processor is needed to maintain the realtime operations correctly. The VxWork by Windriver Inc. and the Irix real-

time operating system by Silicon Graphics, Inc. are two of the realtime operating systems currently available on the market.

The other task of the server processor is the scheduling. Scheduling is used to solve the contention among multiple simultaneous accesses on the common resources such as bus, disk, buffer memory, and I/O interfaces. Scheduling is important in real-time access because the access has a deadline to meet. A good scheduling algorithm will schedule I/O tasks to satisfy the deadlines of most of the accesses, and optimize the performance. The scheduling of real-time video access is still a research topic. More details can be found in [15].

The server processor is usually implemented with general purpose CPUs. When the processing requirement of this server processor exceeds the capability of a single central processing unit (CPU), a multiple processors' approach can be used to enhance the processing capability. Under this situation, several processors can either locate on the same bus in Fig. 2-1, as is done in the Silicon Graphics Challenge series of video servers, or be interconnected in a more complicated way, such as the N-Cube approach in Oracle video servers. Exactly which architecture to use depends on the performance requirement and the cost. A single bus approach apparently can be implemented with a lower cost. On the other hand, a more elaborate link N-Cube can provide a higher performance and expansion capability.

2.3 Conclusion

In this chapter, we briefly review the technologies used to build the components in an advanced video service environment. The emerging advanced video services impose many requirements on the workstation, the network, and the storage server. It requires the three components to have high bandwidth/capacity, high processing capability, real-time operation capability, and the capability to handle heterogeneous media traffic. These requirements make the three components quite different from their conventional counterpart.

They all need to be re-designed and enhanced to support the required capability. This is one major bottleneck that prohibits the multimedia services from growing fast. However, most of the needed technologies in these components are getting mature. The cost and performance will also improve quickly.

Besides the optimization in each of the workstation, network transport, and the information server, system level optimization arises when all the three components are integrated together. When these components are connected through the network, it becomes a fully distributed environment. All the distributed processing issues, such as resource management and load balance, appear in this environment too. There are also tradeoffs between the capability of these three components. That is, the performance of one component can affect the requirements on other components. For example, using a more powerful compression workstation, we can reduce the network bandwidth requirement, however, at the cost of longer delay. Using a local storage device to save frequently used and less frequently changed material can also save the bandwidth on the network. It can also save the capacity and bandwidth requirement on the remote server. How to allocate the distributed processing resources over the network to optimize the overall network resource utilization is also an important issue in the system design.

A basic issue that we have not described so far is the information structure that specifies the relations among heterogeneous media to enable efficient information transport, processing, and presentation. This includes specifying basic components of information materials and defining the data structure of the combination of all basic elements for advanced video services. All the components that we reviewed so far need to rely on this information structure to realize the multimedia information support. For example, the workstation needs to use this structure to present various media to the user correctly. The storage server must have this structure built in to efficiently store and retrieve related information. The network also needs to use this structure to satisfy the real-time transport requirements. Apparently this information structure is very important and needs to be

studied in detail. We will explore this issue in detail in the next chapter, and propose our structured video model.

CHAPTER 3

STRUCTURED VIDEO MODEL

The advanced video services described in the previous chapter require a multimedia workstation to receive various information, either video, audio, pictures, graphics, or text, from any resource accessible on the network, and integrate them in a way suitable to personal use. Among these features, the integration of visual information onto a single display (usually a CRT monitor in today's workstation) is most interesting because of the high processing and bandwidth requirements for visual information, especially full motion video sequences. This integration process, which we call *video compositing*, combines several video sources into a single display stream. Typical compositing includes overlapping, clipping (possibly to non-rectangular shapes), scaling, blending, translation, etc.

In conventional broadcast video, compositing can only be performed in a video studio for video program production. All video elements (e.g., foreground news reporter, background weather map, text,) are combined into a single rasterstream in the video studio before the signals are sent out for broadcasting. For advanced video services, video

compositing can be done not only in a video studio, but also at any place on the network or locally at the user's display. In addition, users want to have control over the pictures they manipulate and view. For example, in multi-way video conferencing, a user may want to display only part of the participants, or he/she may want to enlarge the image size of a participant, and simultaneously display another video sequence from a database, while the other user may want to arrange the information in a totally different way.

For this purpose, the most flexible choice is to keep the visual elements logically separate and use a structural representation for the final composed result shown on the user's display. In this structural representation, the compositing operations can be assigned easily to any existing resource, from the video sources through the network to the user's end. The representation also provides an interactive interface to allow the user to dynamically change the structure of the visual information, and thereby change the displayed scene. The structural representation can also be used for efficient performance analysis, resource allocation, network administration, and system implementation. There have been some efforts to standardize the coding and exchange formats for multimedia representations and documents, but not for performance analysis and mapping to practical implementations in networks.

In section 3.1, we first discuss the objectives of the representation model for composite video. Then in section 3.2 we review several existing representation standards developed for multimedia applications, including ODA¹[4], HyTime[3], and MHEG²[1][3]. In section 3.3, we propose our *structured video model* with some detailed description of its components, functionality, and representation. In section 3.4, we discuss various advantages of the structured video model. Then we give a brief conclusion in section 3.5.

¹ ISO 8613: Office Document Architecture.

² ISO/IEC JTC1/SC2/WG12, known as Multimedia and Hypermedia information coding Expert Group (abbrev. MHEG).

3.1 Objectives

In this section, we describe the objectives/features of the representation model we have in mind. There are basically six main goals, as listed below.

- Real-time presentation and interchange.
- Logically separate visual information with structural representation.
- Use of video materials as easy as the use of graphics and text.
- Common coding and representation.
- Primitive compositing functions.
- Performance analysis and implementation optimization.

Real-time presentation and interchange

The model will support the presentation of full motion video compositing in real time. In order to achieve this, it will require as little processing as possible to have the final displayed scene. It is not intended for a full featured editing purpose such as standards like ODA or HyTime, which requires a lot of processing to solve the cross-referencing or hyper-linking before the final presentation is available. It is not intended for co-editing either. On the other hand, the representation model will provide limited editing capability to allow users to interactively change the appearance of the displayed scene in real time.

In addition, the representation model also supports real-time interchange. That is, when a composite scene is to be displayed, all the elements of the composite scene are retrieved in real-time. The element may be available locally, or may be called up through a network connection from a remote site. This is quite different from the document interchange format such as ODA which collects all kinds of information into one single document, and this is then exchanged as a single entity.

Logically separate visual information from structural representation.

To allow users to flexibly arrange the displayed scene, all the visual elements are kept logically separate with the relationships among the various elements specified in some structural representation. Keeping visual elements logically separate has several advantages. First, users can easily change the structure or the relation to rearrange the scene. Second, it allows the flexibility of reusing and editing video information. Usually there are more chances to reuse a simple visual element than to reuse a complicated composite element. Third, it could be more efficient to compress visual informations of different characteristics separately instead of compressing the composite result[7]. Keeping the visual elements logically separate does not imply anything in implementation. The actual compositing operation can be performed at either video source, some nodes on the network, or at the user's end, according to the actual network resources constraint.

Use of video materials as easy as the use of graphics and text

Conventionally, the support mechanism for manipulation and presentation of video and computer graphics comes from two very different technologies. The former one is basically image oriented, with its information arranged in a raster scan format of some fixed frame rate, to be displayed in fixed sized rectangular windows, such as NTSC or PAL standards. The latter one is data oriented, and is basically generated and manipulated by computers. With the video material arranged in conventional format, it is usually hard to reuse and edit without the special equipment found in a video studio. Today, the merging of video and computer technologies requires a more flexible use of video materials.

To achieve this, the representation model will support arbitrarily shaped full motion video sequences. For example, a piece of video information can be a moving person without his accompanying background scene. The shape and size of the video sequence can also change from frame to frame. With this kind of representation, the video material

can be easily reused by overlaying it with some background scene to suit a special need, just as we do with graphics and text today.

Common coding and representation

The representation model will provide a common coding representation for the visual elements such that they can be interchanged easily no matter from what platform/applications these visual elements are generated. To be a useful visual information representation, it must be extensive to cover all possible visual elements, such as bitmap pictures, graphs, texts, video sequences, and flexible enough to allow all possible compression algorithms.

The object oriented approach can be useful in the visual element representation. Instead of standardizing the encoding of every kind of information (i.e., every mono-media piece of information), object oriented methodology can encapsulate data in a flexible way, hide the internal details of various visual elements and provide a uniform higher level interface to the user. It allows the flexibility of using any kind of encoding/compression algorithm as long as the way to access the visual information is described with a "method." The inheritance property of object oriented methodology also allows sharing of common behavior among the contents of different objects. Almost all the current standards use the object oriented approach for object formatting.

Primitive compositing functions

Defining standardized compositing functions makes the use of common modular hardware and software components possible across a variety of applications. On the other hand, it is feasible to keep only the primitive functions that can be realized in real time, considering the constraint of supporting full motion video sequences. Many complicated compositing functions can be realized by the use of combinations of primitive ones. For example, a panning function can be implemented by dynamically changing the translation distance of the *translate* function, possibly together with the *scale* function.

Performance analysis and implementation optimization

For a given displayed scene, there are usually different implementations. The representation model will allow us to analyze the performance of the compositing system. It will also serve as a tool to optimize the implementation according to the actual network and hardware resource constraints, and can also be used to help network management and process allocation.

There is at present some effort to standardize the representation of multimedia information[1,2,3,4]. Most of today's standardization efforts place their major emphasis only on some of the goals described above, especially on the coding representation, but not all. So far, there is not any model that is intended as a tool for performance analysis/optimization. With these considerations in mind, we propose a structured video model, which defines a logical model for any visual presentation in a hierarchical structure. As a logical model, this model does not specify how the compositing system is implemented. Instead, it provides the flexibility of performing the implementation in different ways. The optimization of implementation involves some restructuring of the logical model graph such that it is more efficiently mapped to the existing network with real physical locations. We will describe the model itself in this chapter, and leave the implementation issues to later chapters.

3.2 Review of today's standard in multimedia information representation

In this section, we review some of today's standardization activity, and discuss its relation to the structured video model that we propose. We will review three standards — ODA[4], HyTime[3], and MHEG[1,3]. The various standards proposed represent the community they originate from. In the community of computer users, people are more interested in the hypermedia and the efficiency found in an interactive user interface. In telecommunication community, the emphasis is on high speed communication and synchronization. In publishing, there is more interest in the database structure. In addition to these

standards, there are also proposals for multimedia information structures [2.5.6.7]. Interested readers may refer to the references.

There are some common issues in the multimedia information representation — object formatting, hyper-linking, and synchronization. Object formatting is important in multimedia applications which use various kinds of information with different characteristics. Object formatting not only standardizes the format for information interchange, it also hides the internal details of various kinds of information and provides a common interface to the user. For this purpose, it is common to use the object oriented methodology, which is useful in providing a standard for the object format to represent visual elements as described in the last section. Hyper-linking is used to combine related information and provide interactive capabilities to the user. Bibliographic cross referencing is a simple example which allows the user to search the linked information. More complicated hyper-linking will pop-up the linked information from a window (e.g., on-line help or contextual linking) or play some kind of audio and video streams. Synchronization is used to keep the logical relations in time or space among several pieces of information. The lip sync between the audio and images in a movie is an example of time synchronization. Another example is the triggering of the start of a piece of information by the ending of a second one. The page layout in a document is an example of space synchronization.

All of the standards proposed need to cope with all these three basic issues for multimedia information representation. Depending on its focus, however, each standard may use a simple scheme in some issues and use a complicated scheme for others.

Office document architecture (ODA)

ODA is a multimedia document standard issued by ISO. The ODA document is proposed to facilitate the interchange of documents consisting of text, image, graphics, and sound. It is intended to be used for document preparation, storage, interchange, and presentation of a multimedia document on the page or on a video display. The basic concept

in ODA is *structures*. ODA uses two different structures — logical structure and layout structure. In the logical structure, a document is subdivided into smaller parts according to its meaning. Examples of logical objects are chapters, sections, figures, and paragraphs. In the layout structure, the document is subdivided on the basis of its layout. Examples of layout objects are pages and blocks. Basically, the logical structure is used for input and editing, and the layout structure is used for final presentation. The layout structure is actually a synchronization mechanism — mainly space synchronization for page layout, but also a discrete version of time synchronization by the concept of pages. Using both the logical structure and layout structure allows very complicated editing capabilities and at the same time easy presentation.

The use of both logical and layout structures also creates some disadvantages. There is usually some processing required to convert the document from its logical structure to the layout structure for presentation. The more complicated the logical structure, the more powerful editing features it can support, while at the same time more processing is required for preparing the logical structure into a layout structure. This may not be feasible for real-time interactive editing purposes. Also, the ODA standard does not support full motion video so far.

HyTime

HyTime is a standardized infrastructure for the representation of integrated open hypermedia documents by the Music Information Processing Standards (MIPS). It is derived from a standard music description language (SMDL) developed for music publishing. Because of this, important features in music description such as synchronization and hyper-linking (e.g., repeats and codas in music) are emphasized in HyTime.

For synchronization, HyTime places objects in bonding structures known as *events* which occur at some point in an ordered list known as a *schedule*. Synchronization is the alignment of events within the schedule. The timing model is based on a *virtual time*.

which specifies the duration of objects relative to a master time reference. When the duration of the master time reference is changed, the timing of all the objects are also changed. The same idea also applies to the space domain. To render an object, the model simply places it into a *virtual space*, called *Cosm*, and then performs the projection to convert the object from one space to another. Again, the axis of the virtual space can use a relative scaling unit. For hyper-linking, HyTime allows all information to be linked, whether or not it was explicitly prepared for linking. Also, link addressing is independent of file management or the network architecture of any particular platform. In addition to a regular linking like bibliographic referencing, HyTime also allows an indirect link similar to the pointer constructs in C language.

Similar to ODA, HyTime is also intended to be a document interchange standard whereby all the information is collected into a single entity. In contrast to the input/editing capability in ODA, HyTime places its emphasis on elaborate synchronization and the hyper-linking mechanism. Similar to the limitation of ODA, a complicated mechanism for synchronization and linking may compromise its real-time presentation performance under system limitation.

MHEG¹

MHEG is a standard under development by the CCITT/ISO. It is intended to be a generic layer for objects used by a wide range of applications. Therefore, it is a standard for representing objects, not a document processing or interchange standard. Many applications, such as HyTime and ODA, may use MHEG objects as their basic elements. MHEG allows many data types, such as text, graphics, video, digital audio, to be represented as objects. Each object is manipulated as a single entity. As a standard developed from CCITT, the MHEG standard reflects a communication-oriented way of thinking,

¹ISO/IEC JTC1/SC2/WG12, known as Multimedia and Hypermedia information coding Expert Group (abbrev. MHEG).

which focuses on the multimedia services used in a communication environment such as the interchange of multimedia information through telecommunication networks or by means of a digital storage medium. Under this situation, the real-time presentation, interchange of the objects are the major considerations in MHEG:

- Real-time presentation — MHEG objects are intended for real-time interactive presentation. This requires the support of some real-time synchronization mechanism. Also, the objects are represented as a final form for direct presentation without the additional processing on their structure, such as conversion from the logical structure to the layout structure in ODA.
- Real-time interchange — MHEG is intended to provide mechanisms for real-time interchange with minimum buffering. A serialization mechanism is used to arrange the coding of several related objects in a certain sequence such that the delay is acceptable when objects are interchanged through a limited throughput medium.

Even though MHEG is intended primarily for object representation, it allows each object to go beyond a mono-media object. This is done by the so-called composite object, which designates an object containing some component objects and the inter-object relationship structure. The composite object can grow recursively with the component object itself a composite object. The inter-object relations basically deal with the synchronization specifying where the component objects are to be placed on the time axis and in the space domain.

Structured video and the standards

Among the three standards described above, MHEG is most similar to our structured video model. The basic objective of structured video is to provide a final presentation for real-time multimedia information integration from the network. This is similar to MHEG in that all objects are to be interchanged and presented across the telecommunication network in real-time. On the other hand, the focus on the integration and final presentation of

structured video makes it different from MHEG. MHEG is only for object representation at a lower layer, while structured video can sit on a higher application layer which uses MHEG just like ODA or HyTime does. On the other hand, structured video will not support complicated hyper-linking features such as ODA or HyTime. Instead, structured video will only support limited features that are good enough for general multimedia service use and also simple enough for real-time realization. This is similar to the MHEG composite object, which actually goes beyond the MHEG layer to a higher layer specifying the structure and the relations of several component objects. The features in a composite object are also limited. Structured video defines more involved inter-object relations using primitive compositing functions for output representation as compared with the composite object of MHEG.

Contrasted to the two structures used in ODA, structured video will use one single structure for both input and output. In this way, we avoid the process of converting from the logical structure to the layout structure before presentation. Of course, a single structure may limit the editing capability. Considering that structured video is used for multimedia presentation on the CRT display, the complicated logical structure used in ODA is not needed.

3.3 The definition of a structured video model

In this section, we describe our proposed structured video model. Structured video can be thought of as an extension to the previously proposed *structured graphics* [16]. When the model is extended from still images to full motion video, not only does the compositing in the spatial domain need to be considered as is the case in structured graphics, but temporal domain properties and modelling are also very important. Spatial and temporal domains are the two facets of the structured video. Both need to be carefully considered.

We first illustrate structured video in figure 3-1. In the picture, the composite scene consists of *video objects* of rectangular-shaped bitmaps (a football background), irregular-

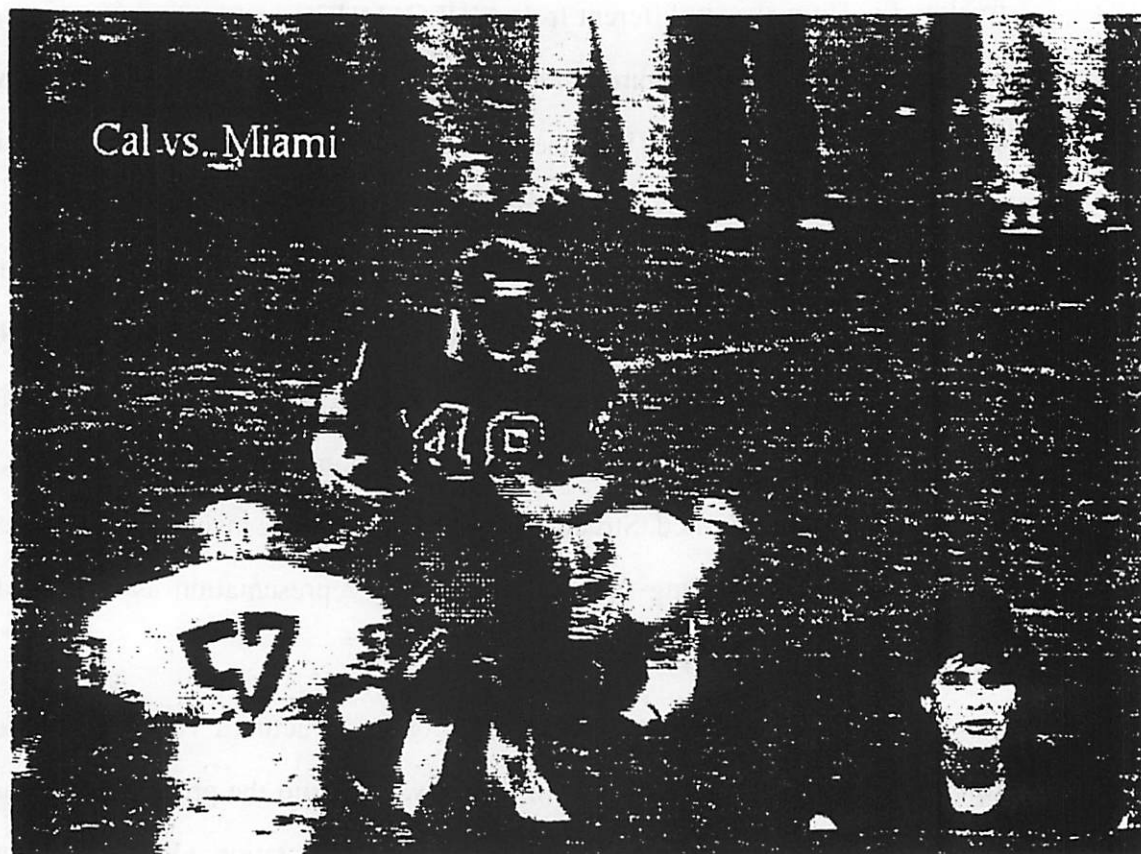


Fig. 3-1. A sample composited picture of structured video.

shaped bitmaps (a news reporter), graphics, and text. The news reporter is opaquely overlapped with the background object, while the graphic object is overlapped with the background in a semi-transparent way. This picture shows only the spatial aspect of compositing. With the temporal aspect, not only is the football background a full motion sequence, but the news reporter is talking as well, with her shape changing from frame to frame. Her location may change from time to time to avoid obscuring important scenes in the background. The graphics and text can appear for a short period of time, and then disappear. Given a set of video objects, there are many different ways to composite them into a displayed scene, both spatially and temporally. It is the goal of our structured video model to define efficient representations for video objects and their compositing rules.

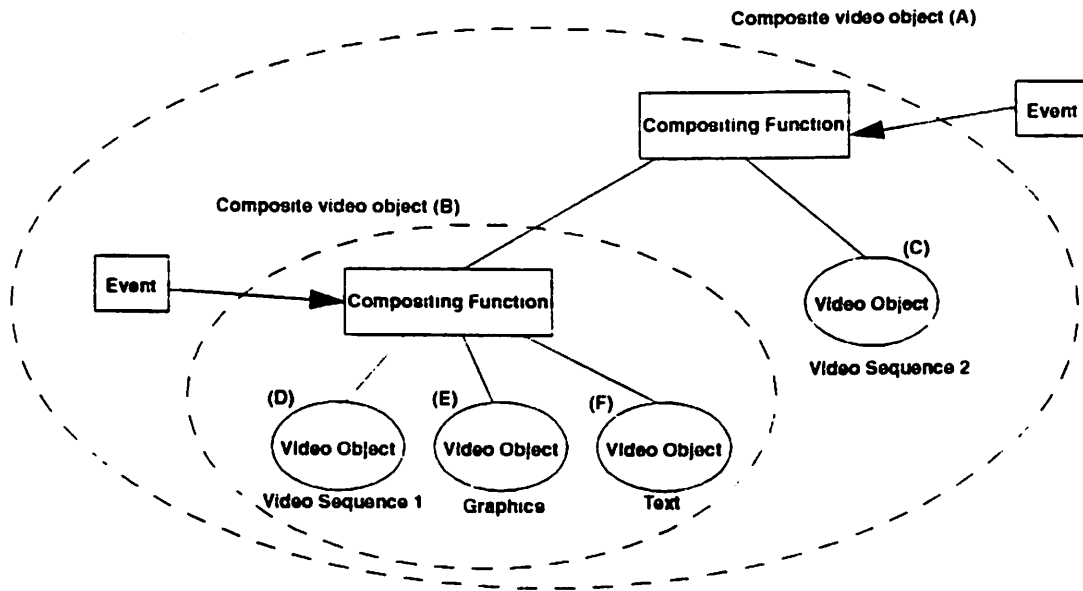


Fig. 3-2. A composited video object in the structured video model.

The structured video model starts with primitive video objects, each of which is a piece of visual information that is generated from a single source, and cannot be further subdivided at downstream sites. *Composite video objects* are formed by combining several video objects (either primitive video objects or other composite video objects) into a single video object with a compositing function. In this way, any complex video object can be represented hierarchically with sets of video objects and compositing functions. An example of a composite video object is shown in figure 3-2. Again, this figure shows only the spatial aspect of the structured video model. When it is expanded in the temporal domain, the graph becomes a 3-D graph, as shown in Fig. 3-3. In these figures, the composite video object consists of a graphic video object and another composite video object, which again consists of two video objects (video sequences) and a text video object. The compositing function specifies how the child video objects are composited, e.g., transparent or non-transparent, scaling, relative locations, synchronization, etc. In addition to video objects and compositing function, a constraints system (not shown in the figure) and events are also used in the structured video model. The constraints system is used to main-

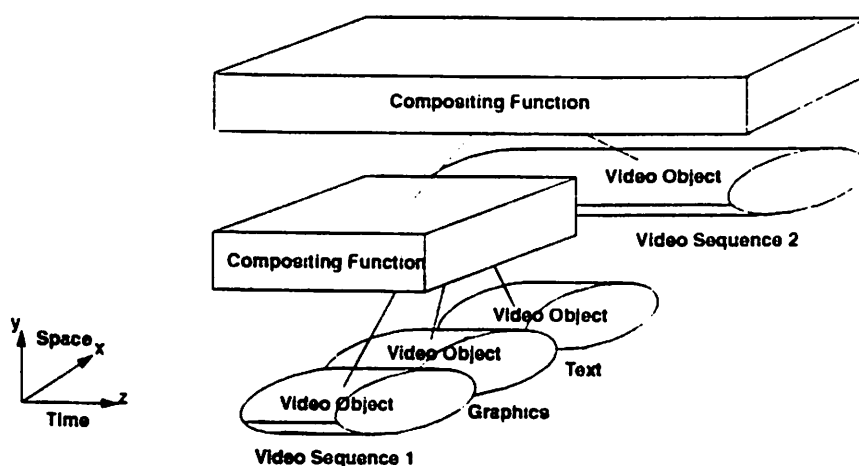


Fig. 3-3. Structured video model expressed in both spatial and temporal domain.

tain the binding relationship among video objects, and the events are used to handle the user interactive inputs (from keyboard or mouse) or exceptions and maintain temporal relations among all the compositing functions. In the following subsections, we will describe the above mentioned components of the structured video model in more detail. Once we understand more about all the components, we will describe the representation of a composite object using a graph such as Fig. 3-3. A 3-D graph is actually difficult to handle. Our approach is to model the spatial and temporal domain separately in two graphs. Fig. 3-2 is an example for the structured video graph in the spatial domain. We will describe both spatial and temporal graphs later.

3.3.1 Video Object

First of all, we define the term video object. A video object is the visual representation of part of some physical or logical object such that the entire part can be composited using the same rules. For example, any graphics or text in a video sequence can be a video object as long as any subregion within the video object is composited in the same way as the remaining parts of the video object when the video object is composited with others. If for any entity in the video sequence, a subregion must be composited differently from the rest

of the entity, then such an entity must be segmented into more than one video object. For example, the representation of a person's hand could consist of separate video objects for each finger and the palm, or could consist of just one object if the whole hand moves as a single unit. A video object can be chosen to be the largest possible unit that can be composited with the same rule to reduce the overhead required for the representation and manipulation of objects. For example, if a person's hand does move as a single unit then it should be described with just one video object. Of course, the number of video objects used to represent any real-world object may change as the object interacts with its surroundings.

In addition to segmenting an entity into multiple video objects in the spatial domain, it is also possible to segment an entity in the time domain or the frequency domain. Segmentation in the time domain is frequently found within the video clips in video editing. If segments of video streams are to be played at different times, they need to be treated as separate video objects. Segmentation in frequency domain is less obvious. The basic idea of this segmentation in the frequency domain is from sub-band coding. For example, a real object can be separated into video objects of "high frequency band object," "low frequency band object." In this way, the quality of the object can be changed by either compositing both video objects together or simply by use of one video object (low frequency band object).

3.3.1.1 Properties of video object

To clearly define a video object, we discuss the property of a video object from three aspects — spatial domain, temporal domain, and general properties. From these properties, we will define basic parameters that are needed to specify such a video object. A list of these parameters are as follows:

- Spatial domain parameters — pixel content, shape description, size, etc.
- Temporal domain parameters — frame rate, time stamp, life span.
- General parameters — object type, coding format.

The spatial domain is orthogonal to the temporal domain. Therefore, the spatial domain parameters may vary from instance to instance. On the other hand, the temporal domain parameters usually do not change with spatial location.

Spatial related properties

One spatial related property is the shape of the video object. A video object need not be in rectangular shape. They can be any arbitrary shape such as head and shoulder of a person. It will be too restrictive on video object manipulation and reusability if we assume that all video objects are rectangular. Most graphics and text objects today are non-rectangular, but most video objects from a video camera are rectangular. There are techniques to generate an arbitrarily shaped video object from rectangular-shaped video sequence such as the chroma key technique. In this thesis, we will not discuss how to generate an arbitrarily shaped video object here. Instead, we only assume that a video object can be any shape, and it has the information needed to describe its shape.

Parameters related to spatial domain properties are the pixel contents and the shape description (if video sequences), graphic description file (if graphic), or the text content and fonts (if text). Other parameters are also needed to ease the compositing operations, such as the video object size.

There are also other parameters which are not inherent in video objects; however, they will be needed in compositing operations. For example, these parameters may be the object location in the composite object, the scale factor for the scaling function, the transparent factor for transparent overlapping, etc. These parameters need not be in the video object. When they are put into the video object, they can be used as default values.

Temporal related properties

From the temporal domain perspective, video objects can be classified into different categories: *isochronous/anisochronous* and *retrievable/non-retrievable*. Isochronous video

objects are the objects that must satisfy certain timing constraint in order to make the presentation meaningful. For example, movies are constant 24 frames per second picture sequences. If the presentation does not abide with the 24 frames/sec timing constraint, the movie is distorted. The timing constraint may or may not be fixed frame rate as in the movie case. They can also be non-constant rate, e.g. the computer animation case. This non-constant rate type is useful in describing video objects that does not changes very often.

Retrievable video objects are video objects that are stored in some device. They can be played at any starting time without any difficulty. The life span of the video object is usually known in advance. The video stored in cassette tape or in disks are examples of retrievable video objects. The non-retrievable video objects are objects that need to be presented at the time they are received, or they are lost. The viewer has no control on the starting time or the length of the presentation. The broadcast TV is an example of the non-retrievable video object. Although the difference between retrievable and non-retrievable video objects seems to be trivial, it makes differences when we consider video compositing.

In video compositing, all the video objects need to be presented at a certain starting time for a certain period of time to make the presentation meaningful. In this compositing process, we are under various kind of timing constraint from video sources. For some objects, we have no control on the starting time without using a large and expensive buffer. For others, we cannot change the presentation frame rate. Also, the timing relationship among video objects need to be maintained. For example, the end of a video object triggers the start of the other video object. To maintain all these timing relationship while at the same time satisfy the timing constraints of the video objects is an issue to explore in this dissertation.

General properties

General properties include all information that is related to whole video object, regardless of the time instance or spatial location. Examples are the video object type and the representation format of the video object. The video object type specifies that either the video object is a retrievable isochronous object, non-retrievable isochronous object, or an-isochronous video object. The representation format specifies what kind of format it is using. A video object can use any format for its content, as long as it is specified and the receiver/user's end recognizes it. For example, a video can use either plain pixel values, or some compressed format, such as JPEG, MPEG, or H.261 [17,18,19]. Graphics can use postscript format or others.

Since video objects are intended to be arbitrarily shaped, the coding mechanism may be non-conventional. The simplest solution of describing arbitrarily shaped video objects is to surround the video object with certain pixel values in order to make it a rectangle. Then we can use the conventional compression scheme to code the video object. Apparently it is not efficient to arbitrarily choose the pixel values to be filled. Some studies have been done on the compression algorithm on the content and the edge of an arbitrarily shaped video object.[28] Segmentation coding methods[42] can also be used to code arbitrarily shaped video objects, even though they were not originally proposed for that purpose.

Several video objects can be combined into a single composite video object through a compositing function. In this situation, the composite video object is treated as a single video object that can be used for further compositing to form an even more complex object. A new set of spatial, temporal, or general parameters will be generated to reflect the actual situation of the composite video object. Basically, a composited video object is just like a primitive video object, with one exception. The way a composited video object appears can always be changed through an interactive event sent to the compositing function, because the representation of a composited video object actually consists of the representations of child video objects and a compositing function. Keeping the composited

video object structure gives users the flexibility of manipulating the child video objects separately.

3.3.2 Compositing function

A compositing function can be used to change/composite one or several video objects (either a composited video object or a primitive one) into one single composite video object. The function describes the way the child objects are changed/composited. Examples of compositing functions are clipping, scaling, blending, translation, delay, etc. The number of child video objects (the video objects that the compositing function operates on) can be one or more. When there is only one child video object, it is a conversion of the video object to a new one of different format or property. An example of this unary operator compositing function is the scaling function, which resizes the video object.

We can use a symbolic notation to stand for a compositing function as $V = f(V_1, V_2, \dots, var1, var2, \dots)$, where V_1, V_2, \dots are child video objects; $var1, var2, \dots$ are variables used by the compositing function to decide how the video objects are composited; and V is the composited video object. There are two classes of variables in the compositing function. One is the variables that appear in all compositing function. Examples are locations L (to determine the location of child objects with respect to the parent one) and the start time T (to determine the timing delay of child objects with respect to the parent timing). Every compositing function requires a set of such parameters for each video object. Because of this, these parameters can be put into video objects as default parameters to simplify the representation of the compositing function. For example, an overlap function combines one video object above the other according to the location L and timing T represented as $over(V_1, V_2, L_1, L_2, T_1, T_2)$ can be simplified as $over(\bar{V}_1, \bar{V}_2)$, where \bar{V}_i represents the video object V_i with default setting of (L_i, T_i) . However, we must note that even though the parameters L_i, T_i are put into a video object, they are not characteristics of the video object V_i . They are simply default values used when V_i is composited with other video objects and are meaningless if V_i stands alone. The other class is the variables that appear only in

certain compositing functions. Some examples are the transparency (used to determine the transparency of objects in transparent overlapping) and scale (to determine the scaling factor in a scaling function). A transparent compositing function can be represented as *transparent* (V_1, V_2, τ_1, τ_2), where τ_1, τ_2 represent the transparency factors.

The compositing result of the compositing function is still a video object. The compositing function not only generates the pixel information of the composited video object, it also generates all the parameters inherent in all video objects, such as the representation format, size, described in the last section. In this way, a composited video object can be hierarchically structured from a compositing function and several child video objects, where the child objects themselves in turn are composited video object. Table 3-1 lists

TABLE 3-1: Examples of basic compositing functions

	Name	Description
<i>unary</i>	<i>translation</i> (V, I)	Change the location from the default value by I .
	<i>delay</i> (V, T)	Change the time from the default value by T .
	<i>scale</i> (V, s)	Re-scale object V with the scaling factor s .
	<i>scale_x</i> (V, s)	Re-scale object V in the x-direction with the scaling factor s .
	<i>scale_y</i> (V, s)	Re-scale object V in the y-direction with the scaling factor s .
	<i>flip_x</i> (\bar{V})	Flip the object V in x-direction.
	<i>flip_y</i> (\bar{V})	Flip the object V in y-direction.
	<i>rotate</i> (V, r)	Rotate object V clockwise by r degrees.
<i>binary</i>	<i>over</i> ($V_A V_B$)	Put object A over object B .
	<i>in</i> ($V_A V_B$)	Display the pixels of object A inside object B .
	<i>out</i> ($V_A V_B$)	Display the pixels of object A outside object B .
	<i>transparent</i> (V_A, V_B, τ_1, τ_2)	Composite object A and B transparently according to τ_1, τ_2 .
<i>multiple</i>	<i>sequence</i> ($V_A V_B V_C \dots$)	Combine video objects which are non-overlapped in time domain into one single video object.

some useful compositing functions. A more detailed description of the compositing func-

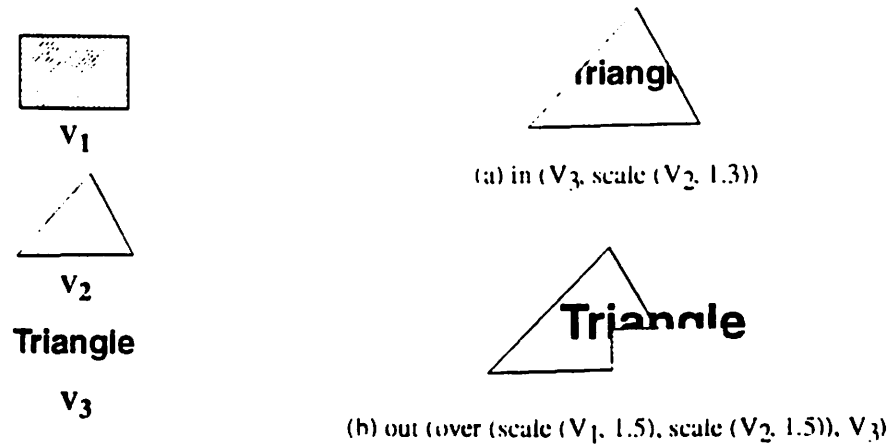


Fig. 3-4. Examples of composited video objects using basic compositing functions in Table 3-1.

tions and their equations is discussed in Chapter 4. Figure 3-4 shows some examples of composited video objects using the basic compositing functions in Table 3-1. Most of the other complex compositing functions of interest can be implemented with the combination of these basic compositing functions. A *zooming* function, for example, can be implemented by dynamically changing the scale size s from frame to frame using the *scale* function. Similarly, a *fading* function can be implemented by dynamically changing the transparent factor τ using the transparent function.

3.3.3 Basic compositing functions

In this section, we discuss basic compositing functions that are useful in video object compositing. We listed the functions in Table 3-1 in the previous section. Here, we show how we came up with the basic compositing functions. The functions in the table basically fall into two categories — *transformation functions*, and *compositing functions*. Transformation functions are the unary functions listed in Table 3-1 that are basically used for changing properties of a video object, such as translation, delay, scaling, and format transcoding. We are more interested in the compositing functions that have multiple video objects involved and only this class of compositing functions will be discussed here.

Again, the compositing function involves both spatial and temporal aspects. Every compositing function must consider both aspects in order to produce a correct compositing result. We will discuss these two aspects separately to clearly define each compositing function.

3.3.3.1 Spatial aspect

over(), in(), out()

In conventional computer graphics, Porter and Duff [35] listed the twelve spatial compositing results between two objects by the way the objects contribute to the four intersect areas, i.e., $A \cap B$, $\bar{A} \cap B$, $A \cap \bar{B}$, and $\bar{A} \cap \bar{B}$. Only one object can contribute to each of the intersection areas. In this way, there are 3 possibilities in $A \cap B$, 2 possibilities each in $\bar{A} \cap B$ and $A \cap \bar{B}$, and only 1 possibility in the area of $\bar{A} \cap \bar{B}$. The total number of combinations is therefore $3 \times 2 \times 2 \times 1 = 12$. Table 3-2 shows the twelve spatial compositing

TABLE 3-2: Spatial compositing results in conventional computer graphics.

Operation	Diagram
Clear	
A	
B	
A over B	
B over A	
A in B	
B in A	
A out B	
B out A	
A atop B	
B atop A	
A xor B	

results derived in this way. Some of these compositing functions are of interest primarily for specialized graphics effect, and some of them are also useful in the compositing of full motion video objects. For example, we may find the use of the function *A in B* in real life when object *B* shows up through a window opening *A*. However, we may not find any use of *A xor B* in real life.

The twelve spatial compositing results of Table 3-2 can be simplified into five binary compositing functions, *over()*, *in()*, *out()*, *atop()*, and *xor()*. Any compositing functions in Table 3-2 are covered by these five functions with proper operand exchange. Among these five compositing functions, *atop()* and *xor()* are less useful and are left out in Fig. 3-1. Choosing only the subset of the compositing function does not limit the flexibility. The *atop()* and *xor()* can still be achieved by the combination of the basic functions. For example,

$$atop(A, B) = over(in((A, B), B)) \quad (3-1)$$

and

$$xor(A, B) = over(out(A, B), out(B, A)) \quad (3-2)$$

In addition to the binary compositing functions, we are also interested in multiple object compositing functions. Unlike the binary compositing functions, which has only twelve functions, multi-operand compositing functions are more complex. The number of compositing functions of a three-object compositing can be shown to be 864, and the number of compositing functions grows rapidly as the number of objects increase. Among the 864 functions, most of them are of no practical use. It is not possible to exhaustively implement all different functions. One possibility is to assume that most of them can be represented as the combination of binary compositing functions. For example, *over(A, B, C)* can be represented as *over(A, over(B, C))* or *over(over(A, B), C)*. In this way, most of the useful multi-operand compositing functions can be implemented through basic binary compositing functions. Note that some of the multi-operand functions arising from

the combination of binary functions are redundant. This issue will be discussed later in the restructuring of structured video in 3.4.2

transparent()

Table 3-2 lists all binary compositing functions where each intersection area consists of only one object. When we consider the case that both objects can contribute to the same area, we come to transparent compositing. The transparent compositing function has been used a lot on TV programs which overlay graphics onto the full motion video background without fully blocking the video background. It is also useful for the case such as looking through a window glass which is not completely transparent. Porter and Duff defined a simple transparent function with a simple *plus* function, i.e., $plus(A, B) = P_A + P_B$. To make the transparent compositing function more versatile, we define the transparent compositing function with an extra transparent weighting factor τ . That is,

$$transparent(A, B) = \frac{\tau_A P_A + \tau_B P_B}{\tau_A + \tau_B} \quad (3-3)$$

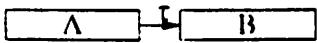
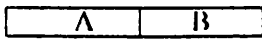
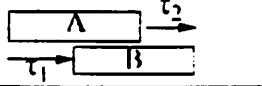
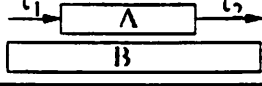
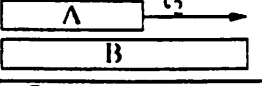
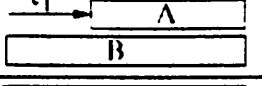
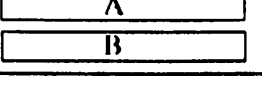
While α values (to be defined in the next chapter) less than 1 could be used to describe semitransparent video objects, they are different from the transparent weighting factor τ in equation (3-3). The τ value is helpful in describing semitransparent video objects, such as windows, glass, and fog, in that the τ value applies to an entire video object rather than to just one pixel.

3.3.3.2 Temporal Aspect

While considering the spatial results of the compositing functions, it is also important to consider the temporal aspect. Every compositing function needs to take care of both the spatial and temporal aspects. Similar to the 12 spatial compositing operations between two video objects shown in Table 3-2, we can also list all the possible compositing relations in the temporal domain. Given any two intervals, there are thirteen distinct ways in which they can be related in time, as described by J. F. Allen[62]. We list only seven out of the

thirteen relations in Table 3-3. The remaining six are simply the inverse of these relations.

TABLE 3-3: Temporal relations from any two intervals.

Operations	Diagram
<i>A before B</i>	
<i>A meets B</i>	
<i>A overlap B</i>	
<i>A during B</i>	
<i>A starts B</i>	
<i>A finish B</i>	
<i>A equals B</i>	

Note that the spatial and temporal aspects are orthogonal to each other. Combining both spatial and temporal aspects, we will have to define $4 \times 7 = 28$ compositing functions. (That is, 4 from the spatial aspects, as described from the previous section, and 7 from the temporal aspect in Table 3-3.)

Instead of performing this combination directly, we choose a simpler way. We will assume a global timing available to all the compositing processors and the sources of the video objects. Each compositing function simply specifies the starting time of the video objects, and uses the life span of the video object to determine how long the object will be shown on the composite result. With global timing, the temporal relations are implicitly specified by the start-time and the life span of all the objects. When only one object is present, the compositing function passes the object to the composite output. When both objects are present, then the compositing function performs the actual spatial compositing. The life span of the composite result is the interval that covers the life span of all the

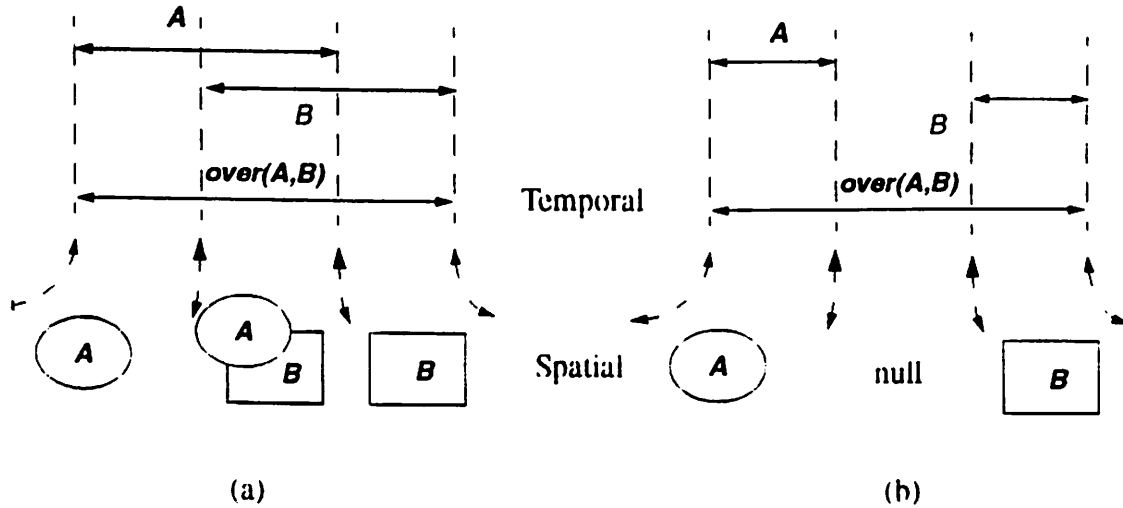


Fig. 3-5. $over(A,B)$ results under different temporal relations.
 (a) Temporal relation: A overlap B. (b) Temporal relation: A before B.

inputs. Under this condition, one spatial compositing function can cover all kinds of temporal relations. The composite results, however, are different depending on the actual temporal relations of the input video objects. For example, the outputs of the $over(A,B)$ function shown in Fig. 3-5 are dependent on the temporal relations of the two input objects.

The life span of the compositing result is also shown in Fig. 3-5. The life span not only indicates the duration of the composite object, but also shows how long the compositing resources are occupied. This is important in scheduling when the structured video tree is mapped into physical compositing resources, which we will discuss in later sections. Note that Fig. 3-5(b) shows a composite result that has a null result in its life span. This is a condition that may not effectively utilize the compositing resource; however, it is a permissible condition.

sequence()

One compositing function that have not been mentioned in Table 3-1 is the *sequence()*. This is a useful compositing function that serializes the input video objects into a sequence

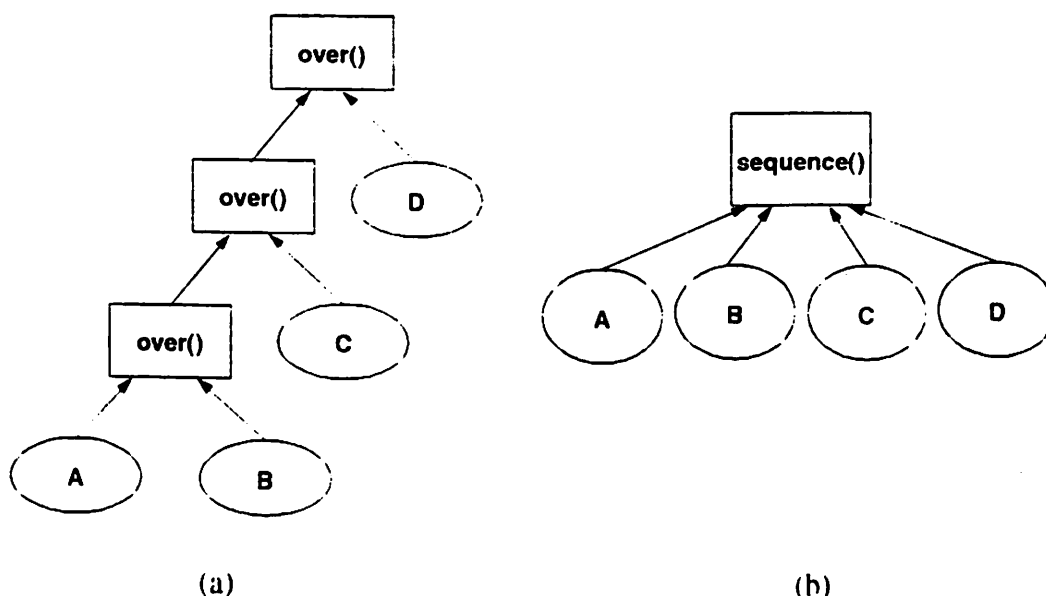


Fig. 3-6. A composite object structure of slide show (a) with and (b) without using *sequence()* function.

in the time domain such that a video object always starts immediately after the previous object, i.e., the temporal relations *meet()* and *before()*. Since the life spans of all the input video objects do not overlap one other, *sequence()* can be used to maintain the temporal relations without needing any spatial compositing capability. One possible use of the function is to organize a slide show, in which the function maintains the slides in some proper sequence. The slide show composite object can still be achieved without using the *sequence()* function, as shown in Fig. 3-6(a). Use of the *sequence()* function, however, will make the tree structure more compact (Fig. 3-6(b)). Also, the use of the *sequence()* function may avoid wasting the spatial compositing resources.

There is one limitation in the use of the *sequence()* function. Since our assumption about using the *sequence()* function is knowing the life spans of all input video object, non-retrievable isochronous objects with an undetermined life span cannot be used as the input of the *sequence()* function.

3.3.4 Constraints

While doing compositing, there are usually some binding relations that must be maintained among the video objects in a composited video object. We call this a *constraint* of the composited video object. For example, suppose that in a weather report program, a weather man is pointing to a map describing the temperature or air pressure. If the weather man and the map are two separate video objects, then the relative location of the two video objects must be maintained, otherwise the weather man may point to the wrong place, and the combination of the two video objects makes no sense. Other examples of constraints are synchronization (time relation), the priority relation (which object is at the top, which one is at the bottom), and scale relation (all child video objects must be kept in the same scale). We will call the video object property specified in the constraint, such as size and location, the constraint variable in the following discussion.

In implementation, these constraints are specified and maintained by a *constraint system*. With the constraints specified, any changes in the constraint variables of a video object (e.g., size, location) will also change the same constraint variables of the whole composited video object, or other video objects bound with this video object. The constraint system is very similar to the constraint model in computer graphics [20,21]. The constraint system keeps track of constraint variables of logically bound video objects and makes the appropriate changes accordingly whenever it detects any changes in any of the bound video objects. With this constraint system, we can relieve the burden of maintaining constraint relations from the compositing functions. Compositing functions simply follow the variables the constraints system generates for it.

There are many ways to model a constraint. We can literally describe the constraint with a list of rules and their related video objects, or we can use a certain data structure to describe the constraint. In order to clearly describe the way in which the video objects interact with each other in our proposed structured video model, we use a tree structure to represent a constraint, which specifies the related video objects and their associated com-

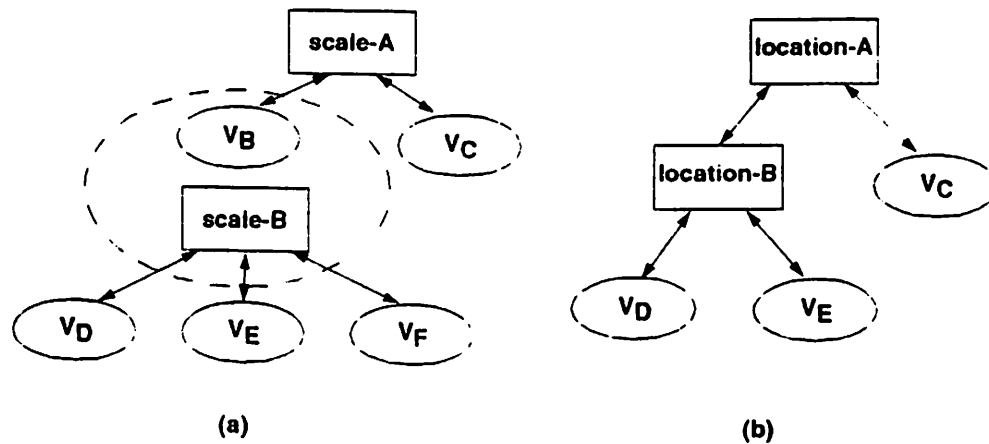


Fig. 3-7. Constraint representation examples for the composited video object shown in figure Fig. 3-2.

positing functions, as shown in Figure 3-7. Each constraint tree specifies one constraint. For example, Figure 3-7 shows two constraint trees for the composited video object shown in Figure 3-2, one for "scale" and another for "location." The constraint tree is a tree reduced from the original tree structure of the structured video, and includes only the related video objects specified in the constraint. Each rectangular block represents a relation among the constraint variables of the video objects linked to it. The literal descriptions of the constraint relations (e.g., *scale-A*, *scale-B*, *location-A*) can actually be considered as mathematical functions, which we call *constraint functions*, of the constraint variables. The constraint function produces new constraint variables if any of its inputs change. The new constraint variables will in turn change other constraint variables if they are linked to other constraint functions, and the effect of the change will be propagated through the whole tree. Each constraint function is associated with one compositing function in the original tree structure of structured video. For example, the *scale-A* in Figure 3-7(a) represents the scale relation among child video objects V_B and V_C of compositing function A. In case the scale variable of object V_C changes, the constraint function *scale-A* calculates the corresponding scale change that V_B should take, and sends the value to object V_B . The constraint function *scale-B* receives the new scale variable of V_B , generates

new scale variables of its child objects V_D , V_E and V_F accordingly, and then sends them to corresponding source sites respectively.

A constraint can bind child video objects of either the same compositing function or different compositing functions. It can bind either all or a subset of child video objects of a compositing function. In Figure 3-7(a), all three child video objects V_D , V_E , V_F of compositing function B are constrained with another video object V_C . In this case, we encircle the composited video object V_B with the constraint function *scale-B*, meaning that V_B is not only constrained by *scale-A*, but also by *scale-B*. This offers an implementation flexibility. The constraint system can either change the composited result at the output of compositing function B , or it can change the child video objects V_D , V_E and V_F separately. On the other hand, in Figure 3-7(b), V_B does not appear with the constraint function *scale-B*. This means the constraint function *location-A* does not operate on the whole composited video object. Instead, it simply operates on the two child video objects V_D and V_E . Under this situation, the constraint can only change V_D and V_E separately, and does not change object V_B or V_F .

Note that the arrows of the constraint system are bidirectional. The constraint function must also be bidirectional; namely, it can receive the parameters from any child video object and make changes to other child video objects. For example, the change of V_C will cause V_D and V_E to change in Figure 3-7(a), and the change of V_D (or V_E) will cause V_C to change also.

3.3.5 Events

Once a composited video object is constructed, some method is needed to interactively change the compositing, to handle exception conditions, and to synchronize the video objects. For these purposes, events are used to communicate between the end user, the constraint system, and the compositing processors. For example, when a user interactively changes the compositing location of a video object, he sends an event signaling this

change to the constraint system. The constraint system then searches through its constraint trees, and sends all the corresponding changes to all related compositing functions. An event can also be used to start a new video object or to delete an existing video object. It can also be used to handle the exception condition when the network or the system breaks down. Another use of events is the communication between the compositing functions and the video object sources for synchronization. To reduce the possibility of buffer overflow/underflow in the transport, a synchronization mechanism is needed to maintain an optimum starting time and frame rate at the video source. For non-retrievable isochronous objects, an event can also be used to notify the receiver of the end of the video object. More details about the synchronization mechanism will be discussed in the next chapter.

3.3.6 Representation of structured video

Ideally, the only way to represent the relations among video objects is to use a 3-D graph such as the one in Fig. 3-3. In a 3-D graph, one axis is used to represent the temporal relations. The rest of the two dimensions are used to represent the spatial compositing relations of the video objects. A 3-D representation graph, however, is difficult to generate and show correctly in a 2-D environment. Our approach is to project the 3-D graph onto the spatial (x-y axis) and temporal domain (z axis) separately. This section briefly describes both spatial and temporal representation briefly. The implementation details and their optimization will be delayed until the next chapter.

3.3.6.1 Spatial representation

A spatial representation has been described briefly and is shown in Fig. 3-2. Basically, the spatial representation uses directed arcs to represent the dependences between all the components such as video objects, compositing functions, and events, and it forms a tree structure. A compositing function is pointed to by all its operand objects through the directed arc. The compositing function together with all the child video objects pointing to it forms a composite video object.

Static or dynamic nature of representation

Since the spatial representation is basically a projection from the 3-D graph, ideally the spatial representation shows all the components involved, no matter when they actually show up. This will form a static tree structure that never changes. A static representation has the advantage of ease of implementation in terms of resource allocation and scheduling. Most of the interactive operations are still available, such as a change in the object location, size, or the delay of a certain video object for a certain period of time. On the other hand, a static representation cannot cover all the situations. From time to time, some interactive operations may add new video objects onto the existing tree structure, or they may change the compositing and change the overall tree structure.

We can adopt the spatial representation in two different ways to handle the dynamic changes of the tree structure. The first is to assume the spatial representation is static. When there is a major change in the tree structure, we terminate the current video object, and start a new one with the new structure. This would ease the implementation of resource allocation and scheduling. However, it has a disadvantage in that it may not satisfy the real time requirement, because tearing down the original object and establishing the new section must be made in real time with no temporal gap. The second way is to assume the spatial structure is actually dynamically changing. In this case, the spatial structure may change when some major interactive operation is issued. The disadvantage of this approach is that the implementation of the dynamic scheduling and allocation may be complicated.

3.3.6.2 Temporal representation

The assumption of the existence of a global clock makes the temporal representation much easier. Based on this clock, all the video object sources can generate video objects at some rate and at a specific starting time. There are other proposals for the timing model that use the fully event driven concept, such as the *Object Composition Petri-Net* (OCPN)[59] model proposed by Thomas Little and Arif Chaffoor for a multimedia storage

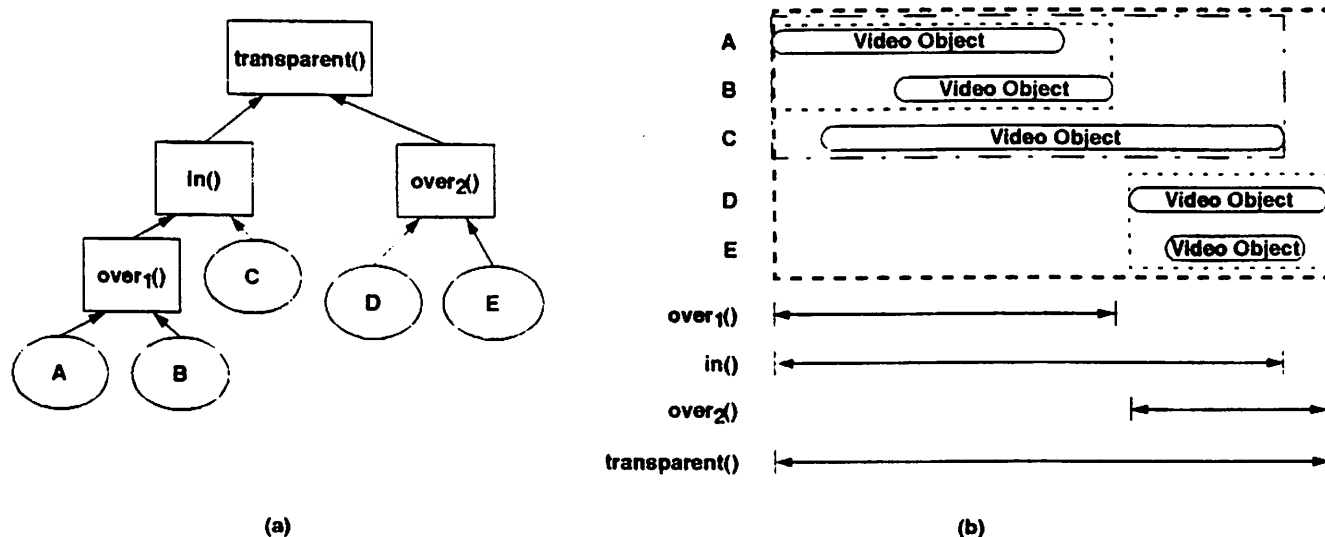


Fig. 3-8. The spatial representation (a) and its corresponding temporal representation (b) of a composite video object.

server. Such a model has the disadvantage of too much overhead in maintaining the synchronization of full motion video sequences with the frame rate. There is no demonstrated video player/display that can maintain synchronization in such a fashion.

Based on the global clock, an example of the temporal representation is shown in Fig. 3-8(b). Its corresponding spatial representation is shown in Fig. 3-8(a). In this representation, the temporal properties (starting time, duration, etc.) are specified by the position and the length of object. The left edge of the video object represents its starting time, and the length of the video object represents its life span. A composite object is represented by a dashed rectangle surrounding all its child video objects. Also shown is the duration of every compositing function. This temporal representation is not especially useful in the resource allocation and scheduling of the compositing functions. For example, Fig. 3-8(b) shows that both `over1()` and `over2()` can share the same compositing hardware since their intervals do not overlap.

Similarly to the spatial representation, a temporal representation may not be static. Interactive operations can change the temporal relations without changing the spatial representation, or vice versa. For non-retrievable isochronous objects without a pre-specified

life span known in advance, the end time of the video object is unknown. The temporal representation should allow this kind of video object as a free-running object which has an indeterminant ending time. The actual end time will be specified at run time by an event sent by the video object source.

3.4 Advantages of structured video

Structured video offers considerable advantages as a model for provisioning of advanced video services. We can divide the service provision into three distinct aspects: the *vendor* or vendors which provide the video object (of course they may also be locally stored), the *transport* or network connection from the vendor, and the *user* who interactively manipulates the video. The essence of structured video is to keep the video objects logically separate, even if they share a common storage or transport, and to support the flexibility of compositing them at later stages throughout the network, such as at the intermediate network node or the user display. The efficient hierarchical structure of the proposed structured video model and the generic representation of the compositing functions enable efficient adaptation of hardware allocations to dynamically changing user resources and application requirements.

Structured video can represent video images over a wide range of applications, for example, a) full-motion high-definition television video, b) conference video, c) videotex, d) interactive video, e) windowing and graphics computer display interfaces, and combinations of these types of representations. A standardized representation enables implementation of display signal processing using a set of common modular hardware and software components across a variety of applications, thereby achieving economies of scale and lower costs. Further, one video display can support a variety of services, depending on the hardware and software modules placed in it.

Similar to the OSI model of data communications, structured video can provide a common model of video shared between vendors, transport networks, and users. Specific

services and applications are provisioned readily by simply parameterizing this model, greatly simplifying the administration of telecommunications networks. A vendor can request the network resources necessary to provision the service, and similarly can query the user's display platform to determine the standardized resources it has and request the resources it needs.

From the perspective of a vendor, keeping video objects separate should enable much greater flexibility and reusability, for example in composing different scenes that reuse common video objects. From the perspective of a user, keeping video objects logically separate all the way to the display enables instantaneous interactive configuration, for example in moving or resizing the participants in a multi-way video conference. The use of standard formats for many image types can support interactive services very easily. Users can combine and customize images from many different people and vendors without requiring that the sources interact at all. A single video stream can be used differently by everyone who receives it without the end users interfering with one other. The control traffic necessary to implement this same interactive functionality at the vendor is eliminated.

Structured video has the potential to enable more efficient compression of complex video representations for storage or transport. Data compression is more effective if done on the separate types of video signals because compression algorithms can be tailored to match the statistics of each data type. It is becoming common to separate video into different regions according to classification algorithms for more efficient compression, but this is avoided if the video objects are kept logically separate in the first place [7,22]. It may also be possible to add to the structured video representation additional semantic information available at the source, such as panning and zooming information, to simplify and improve the compression. Further, if natural scenes are not overlaid with the high-frequency signals caused by text and graphics then standard video compression techniques are more effective. Of course, text and simple graphics can be described very efficiently by

semantic descriptions (fonts, lines, rectangles, arcs) rather than by bitmaps. Graphics and animation sequences can be transmitted much more efficiently by sending the procedure required to generate them rather than the generated images, if there are sufficient processing resources at the user display to execute those procedures.

Structured video can also serve as the basis for adapting services to varying network and user resources, which is an important practical issue. One video display may have limited resources and be targeted at a limited set of services because of the specific hardware and software modules it contains. For example, while an interactive full-motion videotex system would require text, graphics, windows, pull-down menus, and full-motion video mixed together, a computer workstation attached to a low-bandwidth network might require only text, graphics, and windows. The vendor and transport will be able to adjust to varying hardware resources at the display. For example, at the expense of interactive flexibility, object compositing can be done at the vendor or within the transport if the display does not possess sufficient processing capability. This can be done through restructuring of the compositing functions for optimal mapping mentioned in section 3.4.2. In the limiting case, the minimal display can at least accommodate a single rectangular raster-scanned video, which is also supported by structured video.

The computational resources for the final presentation of video can be partitioned arbitrarily along the entire path from production to final presentation, thereby adjusting to economic constraints for a particular service as well as to the available transmission or storage bandwidth. Distribution or broadcast services will tend to place more processing nearer the source, whereas point-to-point or specialized services will generally place more processing near the display. The representation will adapt easily to a) different transmission media (satellite, fiber, cable), b) display devices with varying processing resources, ranging from simple television displays that present only the pixel map representation, to complex workstations that can process all the representations, and c) vendors with widely

varying processing resources ranging from reading pixel maps from storage devices through display and graphics engines.

The subjective quality of presentation also can be adjusted to the transport bandwidth resources. A lower quality can be provided by using only “higher” semantic and graphics representations, with quality up to and including full-resolution HDTV available with the expenditure of additional processing and bandwidth. We also postulate that where limited bandwidth is available, higher subjective quality can be achieved by compressing some video objects more heavily than others, for example in retaining full resolution on a head and shoulders while limiting the resolution of the background.

Structured video also has disadvantages. One disadvantage is the higher cost of display platforms, although as mentioned previously even minimal existing platforms can support a degenerate form of structured video. Another is the expansion of data required to transport the hidden portion of video objects, although this can be reduced with more sophisticated flow-control protocols [8]. There are some limitations in the structured video model relating to its inherently two-dimensional representation (like video itself), as described in more detail in [23].

In summary, the structured video concept of keeping the components used to generate video logically separate past the production process, all the way to the final video presentation if that makes economic sense, is very simple yet powerful. In particular, it will offer flexibility to reuse and modify video material at the user display or at any earlier point. Where pictures are actually assembled from different components (foreground, background, text, graphics, etc.), as will be increasingly the case in the future, it is potentially much more efficient to compress the components before combination than after. Further, structured video allows flexibility to adjust the subjective quality to bandwidth and processing resources (for example substituting a graphically-generated background when bandwidth is not available for a camera-generated background), and the flexibility to adjust processing resources between provider and user (for example in adjusting to band-

width resources). Finally, structured video is consistent with many interactive forms of video, where components generated remotely can be combined easily with locally generated elements to form the final video presentation. Centralized interactive services will segment complex tasks into a) time-critical tasks done locally (graphics and animation), b) reading of high-bandwidth background video too expensive to store locally from a central database, and c) non-data-intensive tasks such as billing performed remotely.

3.5 Conclusion

In this chapter, we propose a structured video model that defines the video material composition in an efficient hierarchical way. The objective is to provide a model such that video information can be easily exchanged, reused, and composed for real-time presentation. The model provides a powerful framework for the provision of advanced interactive multimedia services which can be adapted efficiently based on the varying needs of networks, services, and users. Essentially, the structured video model keeps all the displayed video objects logically separate from sources to the final display, thus enabling instantaneous interactive video configuration, reuse of video objects, and possibly more efficient compression. The model also provides a way for efficient implementation of the compositing system. This is achieved through introducing the basic compositing functions for modular implementation and the restructuring capability of the composite object. The details of composite object restructuring capability has not been discussed yet, but will be studied later in this thesis.

After reviewing the three current existing standards for multimedia information composition, we define the structured video model in section 3.3. We introduced the basic elements of structured video including video objects and compositing functions. We studied video objects and compositing functions from both spatial and temporal aspects to reflect the special requirements of the full motion video compositing. Some compositing functions are listed in Table 3-1 as basic modules. We also introduced a constraint system to maintain the binding relations between separate internal video objects, and events to

handle interactive user inputs, exceptional conditions, and synchronization. Structured video representation graphs in both the spatial and temporal domains are also presented. These graphs are a good means for resource allocation and scheduling in distributed network implementation. In the last section, we also discussed the advantages and limitations of the structured video model.

CHAPTER 4

STRUCTURED VIDEO REALIZATION ISSUES

In chapter 3, we discussed the structured video model for video information integration and presentation. In realization, we want to composite all video information (video objects) in the way specified by the model. The compositing function is used to describe how the video objects are actually integrated. In this chapter, we focus on the compositing function realization issues. When we integrate multiple pieces of video information together, video objects must be composited correctly both *spatially* and *temporally*. *Spatial compositing* means to composite the video objects with correct physical locations on the display, and also to composite according to the rules desired. *Temporal compositing* defines how to maintain the video objects' temporal relations as described in the last chapter. Related video objects, for example a moving weather reporter in the foreground pointing at the background animated weather map, should be combined with correct

timing and frame rate for meaningful presentation. Temporal compositing could also be called *synchronization*.

Spatial and temporal compositing are needed not only in visual information. They are also used for sound information. Spatial compositing in sound refers to the mixing of several audio streams in some proper way. It usually involves gain adjustment before the signals are mixed together. Synchronization is needed between video objects and audio information. Lip sync between the video sequence of a person and his/her sound is an example. Synchronization is also needed between video objects and the events that control the appearance of the composited result.

In the spatial compositing implementation, we first study the *anti-aliasing*, which is needed for compositing arbitrarily shaped video objects in order to have an acceptable video quality in section 4.1.1. Then we discuss the algorithms of the basic compositing function in the structured video model in section 4.1.2. We also study distributed implementation of structured video in section 4.1.3. There may be different ways of video compositing to achieve the same compositing result. One way of compositing may be performed more efficiently than others under the constraint of compositing resources distributed over the network. The study of the distributed implementation discuss how to convert from one way of compositing representation to another to best fit the existing resources. We call this process of compositing representation conversion *restructuring*. In order to do this, we study the generic representation of the primitive compositing functions and their associated properties. Through the study of these compositing functions, we can derive the rules for restructuring. In section 4.1.4, we discuss other alternatives that make the spatial compositing more efficient.

In the temporal compositing or synchronization implementation, we first study the implementation of the global time clock, which is assumed to be available in the structured video model. We also discuss how the synchronization can be maintained among multiple video objects, and propose a synchronization mechanism.

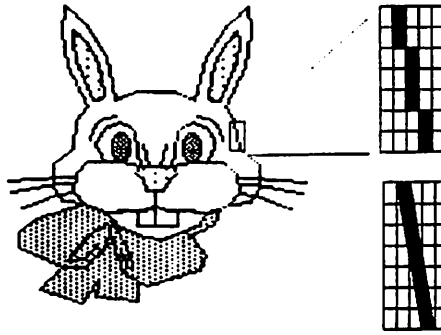


Fig. 4-1. Aliasing effect due to insufficient resolution.

4.1 Spatial compositing

4.1.1 Anti-aliasing

Anti-aliasing is important in video object compositing of arbitrarily shaped objects. Due to insufficient resolution on the display, objects appear jagged at the edge, like a staircase, which we call *aliasing*. An example of this aliasing effect is shown in Fig. 4-1. Porter and Duff proposed a four-parameter ($r\ g\ b\ \alpha$) compositing algorithm for arbitrarily shaped images compositing [12] to reduce the subjective impairment of the aliasing effect. Here, we will review their compositing algorithm.

To represent arbitrarily shaped objects placed over an arbitrary background, Porter and Duff[12] use an additional α channel to represent a pixel in addition to the three colors (either RGB, YUV, or any other format.) The α value indicates the percentage that an object covers a specific pixel location. An α of 0 indicates the object does not cover that specific pixel and an α of 1 means the object fully covers the pixel. At the boundary of an object, an α between 0 and 1 is used to reflect the partial coverage. The shape of a video object can be derived from its α values: the boundary is the area where α becomes greater than 0. An example of the α mask of a triangle is shown in Fig. 4-2.

In essence, the α value is a mixing factor used to control the linear interpolation of foreground and background colors so that the boundary appears smooth. The best choice for this mixing factor is the percentage of a pixel that a video object would cover if we

object and generate the α values at the same time. In this thesis, we will not explore further the derivation of α , and simply assume that it is available together with the pixel values.

4.1.2 Compositing algorithm

Following the 4-channel method by Porter and Duff, we can derive the compositing algorithms for the binary functions in Table 3-1.

$$\bullet C = \text{over}(A, B)$$

$$\begin{aligned} p_C &= \frac{\alpha_A p_A + (1 - \alpha_A) \alpha_B p_B}{\alpha_A + (1 - \alpha_A) \alpha_B} \\ \alpha_C &= \alpha_A + (1 - \alpha_A) \alpha_B \end{aligned} \quad (4-3)$$

$$\bullet C = \text{in}(A, B)$$

$$\begin{aligned} p_C &= p_A \\ \alpha_C &= \alpha_A \alpha_B \end{aligned} \quad (4-4)$$

$$\bullet C = \text{out}(A, B)$$

$$\begin{aligned} p_C &= p_A \\ \alpha_C &= \alpha_A (1 - \alpha_B) \end{aligned} \quad (4-5)$$





$$\bullet C = \text{transparent}(A, B)$$

$$\begin{aligned} p_C &= \frac{\alpha_A \tau_A p_A + \alpha_B \tau_B p_B}{\alpha_A \tau_A + \alpha_B \tau_B} \\ \alpha_C &= 1 - (1 - \alpha_A) (1 - \alpha_B) \end{aligned} \quad (4-6)$$

In the above equations, we also generate the α value of the composited result, so that the composited object can be further composited with other objects. This derived α value represents the overall portion occupied by the composited pixel. There are some approximations used in the derivation of equations 4-3 to 4-6. First, the derivations of the α values are only approximations. Actually, without knowing the detail edge information inside a pixel, we cannot get the exact α value of the composited result anyway. Our assumptions

are shown in table Table 4-1. Let α_A and α_B be the α value of pixel A and B . The intersec-

TABLE 4-1: α value derivation assumptions

Label	pixel	α value
A		α_A
B		α_B
$A \cap B$		$\alpha_A \alpha_B$
$A \cup B$		$1 - (1 - \alpha_A)(1 - \alpha_B)$

tion of the portions occupied by both pixels are $\alpha_A \alpha_B$. The union of both pixels is $1 - (1 - \alpha_A)(1 - \alpha_B)$. This is an orthogonal relationship assumption between the edge of the two pixels. Similar results can be derived for the cases of $A \cap \bar{B}$ and $\bar{A} \cap B$.

Similarly, without knowing the detail edge information of the pixels, we cannot get an exact result for transparent compositing, either. The assumptions in Table 4-1 can apply to the transparent compositing function. However, it makes the equation quite complicated because it segments a pixel into four portions and calculates the result separately. In equation 4-6, we use another simple assumption that contributions from both pixels are not only determined by the transparent factor τ_A , τ_B , but also by α_A and α_B . We are using the product of α and τ , i.e., $\alpha_A \tau_A$ and $\alpha_B \tau_B$ as the weighting factors. The reasoning behind this is that a pixel occupies a larger area (larger α) and ends up contributing more in this transparent compositing function. The simulation result shows that both of these approximations are quite acceptable.

4.1.3 Distributed implementation of structured video

In this section, we discuss the flexibility of structured video implementation in a distributed network system. The structured video model is general enough to represent any kind of video compositing, and is useful in allocating hardware resources for implementation in a distributed environment. In distributive implementation, components of the struc-

tured video model, such as the video objects, compositing functions, and events, are mapped to physical nodes or transmitted on links in a network. There are different mappings (and therefore different implementations) under the existing resource and network configuration constraint. The performance of each mapping can be determined by the usage of communication links and the compositing resource.

For example, in figure 4-3, we show a simple model of 3-way video conferencing. Figure 4-3(a) shows the composited video objects to be displayed. Figure 4-3(b) shows the mapping to three different implementations. In Figure 4-3(a) a single compositing function is performed at a certain location only. The same composited result is broadcast to all participants. Any change of the compositing from any user will change the composited results on all other user displays. This is exactly the situation for video sharing/co-editing. In Figure 4-3(b) three different compositing functions are performed at the same location. Each produces a composite scene for a user. In this situation, three compositing functions are used, implying 3 times the hardware complexity as compared with Figure 4-3(a). However, each user is given the flexibility to fully control his/her composited video objects. The transmission bandwidth of this implementation is roughly the same as the (b.1) case, as shown by the arrows of the graph. The only difference is that in (b.1) the composited result is broadcast back to three users, while in (b.2) three different composited results are sent separately to the users. Broadcasting may save some network resource as compared with using three different transport entities. In (b.3) three compositing functions are put at the user ends. The hardware complexity (by counting the number of compositing function blocks) is the same as that for case (b.2). However, the required transmission bandwidth may exceed that for case (b.2).¹

There are two different approaches to optimizing the mapping of structured video into a distributed network. The first is to optimize the compositing function scheduling. In

¹ Case (b.2) needs $2 \cdot N$ links while case (b.3) needs $N \cdot (N-1)$ links, where N is the number of users.

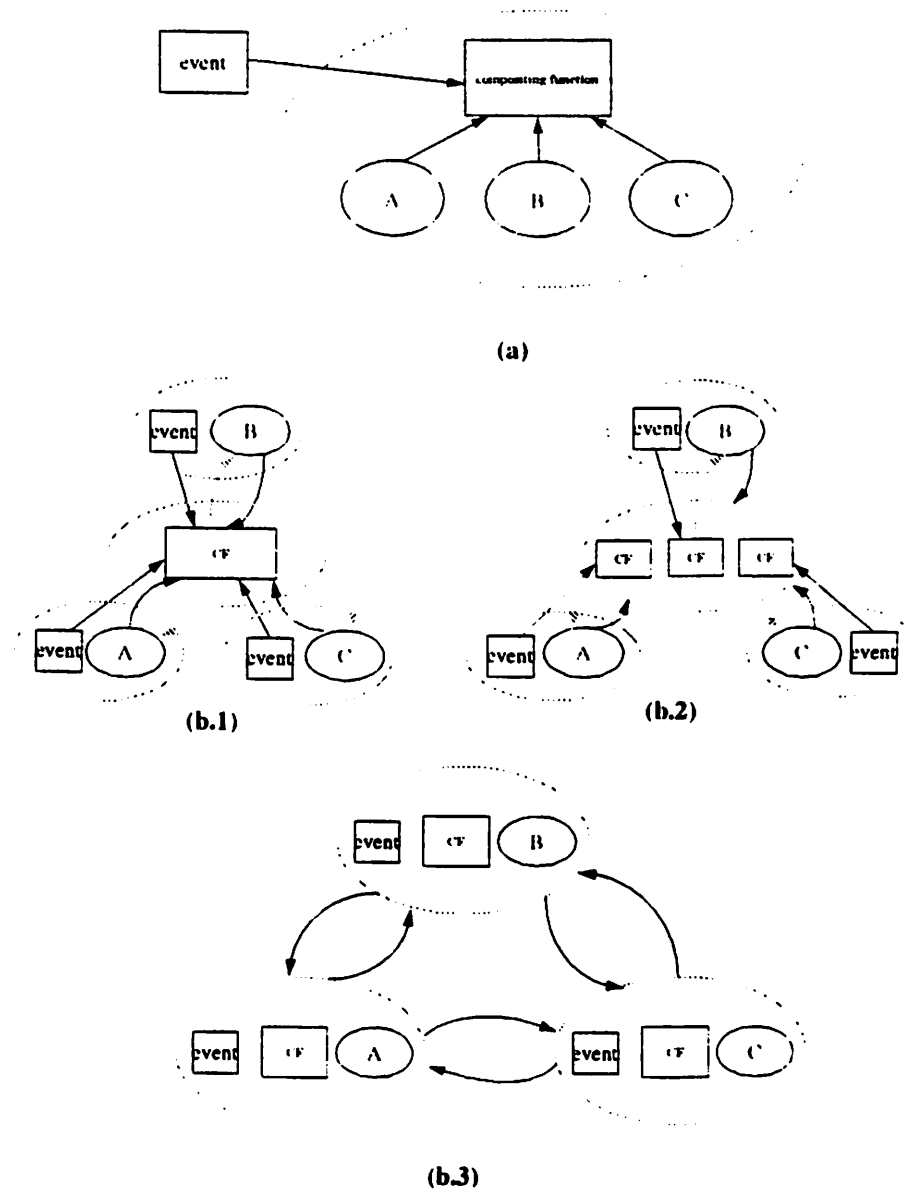


Fig. 4-3. A 3-way video conferencing example. (a) the structured video representation (b.1-b.3) three different mappings to networks.

structured video, most of the composite objects have a limited life span. That is, the compositing processors are used only for a certain period of time. It is possible to schedule several compositing functions into the same compositing processor, as long as the intervals of the compositing functions do not overlap. This scheduling is a very difficult problem, and has been studied extensively in the digital signal processing field. We will not go into much detail regarding scheduling except to make the comment that both static

and dynamic scheduling are of use in structured video scheduling. The choice of either static or dynamic scheduling depends on what kind of video objects are being composited. As described in previous sections, there are various kinds of video objects. For retrievable isochronous objects and an-isochronous objects whose life spans are known in advance, static scheduling is useful. For applications which use nonretrievable isochronous objects or which have a lot of interactive operations, dynamic scheduling is required because no prescheduling can be done until run time. Combining both static and dynamic scheduling may make the scheduling of structured video a very challenging issue.

The second way of optimizing the mapping is through hierarchical tree restructuring. As described in section 3.3.3.1, multiple-operand compositing functions can be formed with the combination of many binary compositing functions. In such a process, several different combinations may have the same final result. In other words, there are possibilities that several different hierarchical trees of composite objects have the same compositing output, and therefore represent the same composite object. On the other hand, different hierarchical trees can provide us with different views of mapping, and also improve the performance. In this section, we will focus on this restructuring issue. To do this, we will study the properties of every compositing function listed before, and find the restructuring rules. In this way, we can enable efficient manipulations of the compositing functions for optimizing the mapping performance.

4.1.3.1 Re-structuring of compositing function

Given a set of video objects, $\{V_i\}$, a compositing function, F , maps them into a composited object, $V_{\text{composited}} = F(V_1, V_2, \dots, var_1, var_2, \dots)$, as described in section 3.3.2. When this compositing function is performed with the combination of multiple unary and binary compositing functions, there can be many ways to complete the compositing functions. For example, overlapping of three objects can be completed with either the first two or the last two objects overlapped first, as shown in Fig. 4-4. Mathematically, this can be represented as $over(over(A,B),C)$ for case (b) and $over(A,over(B,C))$ for case(c) in Fig. 4-4.

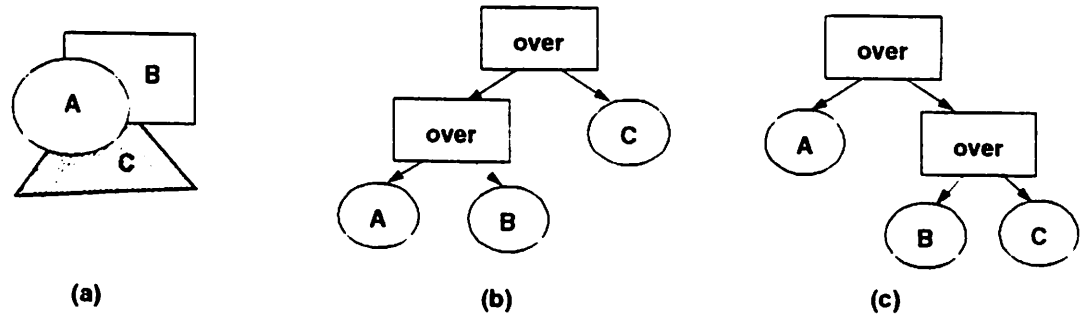


Fig. 4-4. Different implementations for the same composited video object and their tree representations

The mathematical representation will help us analyze the properties of a compositing function in a more systematical way. An example of a more complicated compositing function is shown in the following equation.

$$V_{\text{composited}} = \text{over}(\text{scale}(\text{over}(V_1, V_2)), \text{over}(\text{scale}(V_3), \text{scale}(V_4))) \quad (4-7)$$

The parentheses indicate the processing order for completing the whole compositing function. It is this stiff restriction we want to break in order to support the flexibility for efficiently adapting implementations to the dynamically changing application requirements and network configurations. We achieve this flexibility by exploring the characteristic properties of the generic representations for compositing functions, such as *associative*, *commutative*, and *distributive* properties of the compositing functions. Based on these properties, we can restructure the compositing process so that the final structure implies the optimal implementation. For example, the composited object shown in equation 4-7 can be restructured to a simpler one as follows (if we assume all scaling factors are the same.)

$$V_{\text{composited}} = \text{scale}(\text{over}(\text{over}(\text{over}(V_1, V_2), V_3), V_4)) \quad (4-8)$$

In the following, we discuss these important properties for restructuring the compositing process and their possible effects on the implementation.

A binary compositing function, F , is *associative* if

$$F(F(V_1, V_2), V_3) = F(V_1, F(V_2, V_3)) \quad (4-9)$$

Examples of associative compositing functions are *over()*, *transparent()*, and *in()*. The compositing function *out()*, is not associative. The significance of the associative property is that it allows trade-offs between sequential vs. parallel implementations. For example, if video components are generated incrementally along a bus network, it is most natural to composite these components objects one by one sequentially. On the other hand, if sources of component objects are grouped into several locations, parallel compositing may be more efficient. In terms of hardware complexity, sequential compositing may facilitate efficient implementations like pipelined compositing architectures. But if the sequential compositing process is distributed among many different locations, the end-to-end latency will be long.

A binary compositing function, F , is *commutative* if

$$F(V_1, V_2) = F(V_2, V_1) \quad (4-10)$$

An example of commutative compositing functions is *transparent()*. *Overlap*, *A in B*, and *A out B* are not commutative. When combined with the associated property, the commutative property can determine whether or not multiple video objects can be composited in any arbitrary order. This is important since video objects may come from many different remote locations and their geographic location does not necessarily match the specified compositing order. If the specified compositing order is unchangeable, video objects may need to be sent to a central node for compositing, or some distant video objects may need to be sent to the same node and composited together before they are transmitted to another distant location. Note, for unary operations, that the commutative property implies $U1(U2(V)) = U2(U1(V))$, where $U1$ and $U2$ can be the same operation. For example, scaling and translation are commutative.

The distributive property is defined upon a unary or binary function with respect to a binary function. Operation U is *distributive* with respect to a binary function, G , if

$$U(G(V_1, V_2)) = G(U(V_1), U(V_2)) \quad (4-11)$$

Binary function, F , is *distributive* with respect to a binary function, G , if

$$F(G(V_1, V_2), V_3) = G(F(V_1, V_3), F(V_2, V_3)) \quad (4-12)$$

and

$$F(V_1, G(V_2, V_3)) = G(F(V_1, V_2), F(V_1, V_3)) \quad (4-13)$$

The first advantage of using the distributive property is the reduction of redundant operation by extracting the common operations shared by two video objects. This means the reduction of computations in most cases, although the actual amount of computation reduction depends on the specific operations applied. For example, simple nonoverlap combinations of two objects does not reduce the total data and thus does not reduce computations when a common operation is extracted. On the other hand, using the distributive property to apply the same operation on both component objects before they are composited together may be advantageous in some cases. For example, scaling down the component video objects before they are transmitted to a network node for compositing can save some transmission bandwidth compared to transmitting the full-sized video components and doing down-scaling at the network node.

A table summarizing the basic properties of the compositing functions is shown in Table 4-2. These basic of associative, commutative, and distributive properties show us

TABLE 4-2: Basic properties of compositing functions for restructuring. A: associative, C: commutative (C_b for binary, C_u for unary), D: distributive (F or U wrt. G)

		Binary (G)				Unary			
		Over	Transpa rent	In	Out	Scale	Translat e	Rotate	Flip
Binary (F)	Over	A	D	D	D	(not defined)			
	Transparent	D	A, C_b	D	D				
	In	D	D	A	D				
	Out	D	D	D	+				

TABLE 4-2: Basic properties of compositing functions for restructuring. A: associative, C: commutative (C_b for binary, C_u for unary), D: distributive (F or U wrt. G)

		Binary (G)				Unary			
		Over	Transparent	In	Out	Scale	Translate	Rotate	Flip
Unary (U)	Scale	D	D	D	D	C_u	C_u		C_u
	Translate	D	D	D	D	C_u	C_u		C_u
	Rotate	D	D	D	D			C_u	C_u
	Flip	D	D	D	D	C_u	C_u	C_u	C_u

[†] $(S1 \text{ out } S2) \text{ out } S3 = S1 \text{ out } (S2 \text{ over } S3) \text{ or } S1 \text{ out } (S3 \text{ over } S2).$

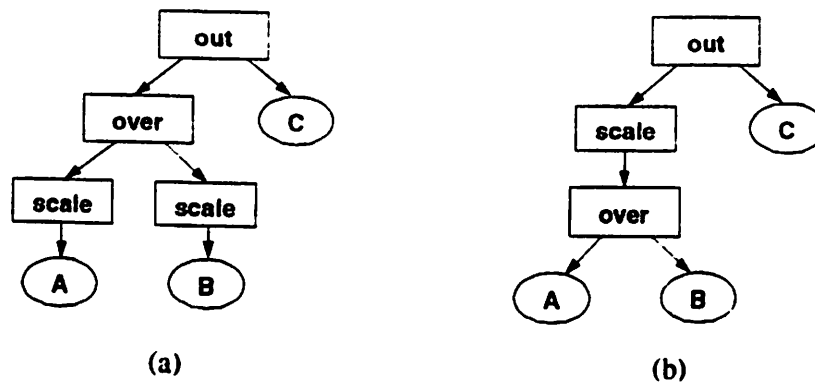


Fig. 4-5. The corresponding action of applying distributive property on the implementation tree, from (a) to (b) — merging, from (b) to (a) — splitting.

various possibilities for implementing the same compositing operation in different tree structures. A more efficient mapping can be found through this restructuring process using these properties. Basically, the associative property allows the moving of a compositing block from one branch to another branch of a node. The commutative property allows the switching of two objects under a compositing block or the switching of two compositing blocks on top of a single video object. The distributive property is equivalent to *merging* two identical underneath blocks into a single block on the top of another binary compositing block, as shown in figure 4-5, or the other way around (*splitting*).

These basic properties for restructuring the compositing functions also facilitate an efficient and systematic approach to matching the structured video implementations to dynamic application requirements. For example, if transmission bandwidth is the most important resource, we should perform compositing at nodes closer to the sources and complete rate-reducing operations (e.g., down scaling) as early as at the source sites. If the computational complexity is of most concern, then finding the simplest computation form should be pursued. On the other hand, if users want the most responsive control, then keeping video objects separate all the way down to the user sites should be most beneficial.

Besides these considerations, there is the challenging issue of optimizing the mapping of several structured video representations for many different services at the same time. A suboptimal mapping for single-user services may become the optimal mapping when multiple services are considered together. In a multi-user heterogeneous-service situation, these restructuring techniques are useful for finding the sharable compositing processes among different users and different services to reduce the overall implementation cost.

To provide more flexibility for restructuring, it is possible to change the basic properties of the compositing functions through different definitions of the compositing function. For example, the definition of $over(A, B)$ places object A at the top of object B . Therefore it is not commutative. However, if we define a new function $over(A, B, z_A, z_B)$ which either places A over or under B depending on the priority values z_A and z_B , then $over()$ becomes commutative. Note that other properties may also change through this new definition. In order to keep the $over()$ function associative, the priority value z needs to be per-pixel based, instead of per-object based. If it is per-pixel based, the composite object has the same priority for the whole object, and the associative property cannot be maintained. This can be seen from the fact that in $over(over(A, B), C)$ the $over(A, B)$ is first performed then composited with object C . Once $over(A, B)$ is performed, it becomes one single video object. We cannot tell the portion in A from the portion in B because a single priority value

is used for this composited object. Under this situation, C cannot be inserted between A and B . This is apparently different from $over(A, over(B, C))$ in the case that A over placed over B over C . By assigning a priority to each single pixel, we can maintain both associative and commutative property for $over()$. Since $over()$ is the most frequently used among all compositing functions, maintaining both commutative and associative properties will provide a great advantage in implementation. It also provides a means to avoid presorting the video objects according to object priority. We will discuss this later in the discussion of the pipelined architecture of the VideoStation.

4.1.4 Possible efficient implementation in spatial compositing

The algorithms described in the last section are a direct implementation of video compositing in the plain pixel map domain. There are ways to make the compositing more efficient to reduce the processing requirement and/or reduce the transmission bandwidth requirement. One possibility is a technique called *compression domain compositing* [26][27]. The other is to use the flow control method. In this section, we will briefly review the compression domain compositing technique, then we discuss the flow control algorithm.

Compression domain compositing

It is common today to use some kind of compression method to reduce the data rate required for video information. With the reduced data rate in compressed domain, it is possible that the compositing takes less processing because there are less data to be processed. On the other hand, when all the inputs and the output of the compositing are in the same compressed domain, e.g., compositing to be performed at some node on the network, performing the compositing in the same compressed domain can reduce the overhead of format conversion which is required if the compositing is performed in the plain pixel map domain.

Compressed domain compositing has been studied in some detail in [26][27]. A brief review of the results shows that efficiency depends on the compressed algorithm, the characteristics of the video objects, and the assumption on the input/output format. Compositing in a simple DCT transform domain is usually more efficient than compositing in the pixel domain. The motion-compensated DCT transform, however, may or may not be more efficient depending on the compression rate. More details can be found in [26][27].

Flow control technique

The other possibility is to use some kind of flow control technique to save the total transmission bandwidth requirement. Usually a composited object consists of objects overlapping each other. There are usually some portions of objects that are obscured and cannot be seen. If it is possible to send this obscurity information to the video object sources, then the obscured portion need not be sent in the first place. This can save some transmission bandwidth.

The basic idea of this flow control technique is to use the information about an object being partially obscured to save the transmission bandwidth required for the video compositing. Our discussion will base on a simple model shown in Fig. 4-6. Fig. 4-6(a) shows a composited object C consisting of object A and B with the compositing function $C = \text{over}(A, B)$. Fig. 4-6(b) shows the video sources S_A, S_B of object A, B respectively, the compositing processor C_G and the connection links between them L_A, L_B . The use of the flow control technique is to reduce the total transmission bandwidth requirement on the link L_A and L_B . With this simple modeling, the bandwidth usage are B_A, B_B on link L_A and L_B respectively. Let's use a weight function W_i to represent the cost of the link i . The total cost function F of the transmission will then be:

$$F = W_A B_A + W_B B_B \quad (4-14)$$

The idea is to inform each video source that some portion of its video object will not be seen, and therefore need not be sent. For example, object B is partially obscured by

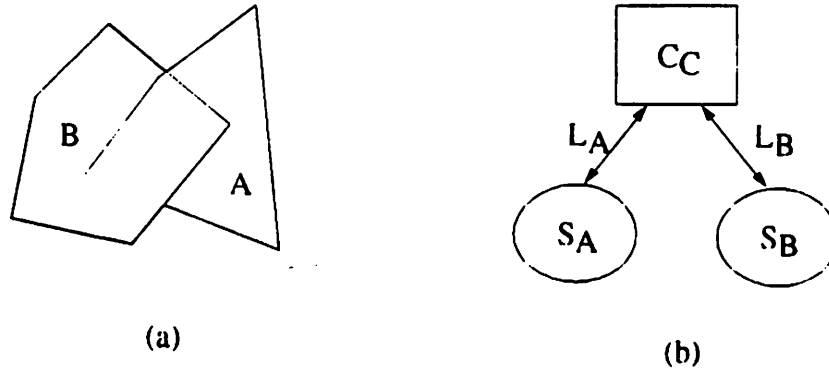


Fig. 4-6. A simple model for flow control scheme using object obscuring information. (a) The composited result. A portion of object B is obscured by object A. (b) The network configuration of video sources S_A , S_B and Compositing processor C_C . They are connected with links L_A , L_B . The results will be sent out through L_C .

object A. With the obscurity information, instead of sending an object B to compositing processor C , video source B sends to C the object $out(B, A)$, the unshaded portion of object B in Fig. 4-6(a), where $out()$ is the compositing function described in the last chapter. Let B_B represent the bandwidth usage of $out(B, A)$. The total transmission cost using the flow control will be:

$$F = W_A B_A + W_B B_B + W_B B_O \quad (4-15)$$

Equation 4-15 assumes the weighting factors of sending information in both directions on link L_B are the same, i.e., $W_B B_O$ is the bandwidth of the obscurity information sent from C to B . In order for the flow control mechanism to be useful, the cost function \bar{F} must be less than F . That is,

$$B_B - B_B - B_O > 0 \quad (4-16)$$

The left-hand side of equation 4-16 represents the final bandwidth saved from sending only the nonobscured portion, including the overhead of the obscurity information. This bandwidth saving must be greater than zero in order for this flow control mechanism to be

useful. The first two terms, $B_B - B_{B'}$, are roughly equal to the data rate of the unshaded portion of object B . The third term, B_O , equals the data rate of the edge information of object A . The exact value of the bandwidth saving depends on the video itself, the overlapped size of the video objects, and the algorithms used to code the video object content and edge information. It is hard to give an exact value for the bandwidth saving. In general, however, the data rate of the code for video object content is several times higher than the code of edge information. Table 4-3 shows some typical ratios of the data rate

TABLE 4-3: Data Ratio of Video Object Content/Edge[†]

Content coding/Edge coding	W x H: 100 x 10	W x H: 50 x 50	W x H: 10 x 100
uncompress/uncompressed bit mask	16	16	16
JPEG/Run-Length-Coding	11.82	7.12	1.492
MPEG/Run-Length-Coding	4.55	2.7	0.574

[†] Assuming the compression ratio of JPEG and MPEG are 10 and 26, respectively.

between the video content and the edge. It shows that for uncompressed video objects, the edge code is only 1/16 of the content codes. If we use run-length coding for the edge, the exact value depends on the size of the video object. Table 4-3 shows three different sizes of video objects. Note that the width and height of the video object shown in Table 4-3 only roughly describe the shape of the video object, because the video objects are arbitrarily shaped instead of a rectangle. The figure in Table 4-3 shows that, for most cases, the data rate of edge information is several times less than that of the content, and it is worthwhile to send edge information to trade for the saved bandwidth from the overlapped shaded portion.

The model described above for flow control is actually oversimplified. Practically there are still issues remaining to be considered:

- First, there are some processing overhead at the video sources to derive the object $out(A,B)$. To make it simple, instead of sending the exact shape of the object A , we may want to send only a rectangle describing the area to be covered by object A . This reduces the processing complexity, and also reduces the bandwidth used to carry the video object edge information. On the other hand, this also decreases the unsent portion of object B .
- The synchronization of both video objects in time needs to be solved. From the model in Fig. 4-6, object B cannot be sent to compositing processor C until object A has arrived at C and the edge information of object A is sent from C to B . This causes a delay of object A s being displayed, and requires more buffer for object A . A possible solution of this problem is that all video objects send their edge information several frames in advance, so that this edge information can be sent to video sources in time to do the flow control. This is reasonable because many coding algorithms today use interframe compression, e.g., MPEG and H.261. To derive the current frame, we need the frames in the past or in the future. It is therefore possible for the coder to put the edge information several frames in advance without much difficulty. In the next section, we will discuss video object synchronization issues further.

4.2 Temporal compositing (synchronization)

Synchronization in general means to maintain events in some temporal order. Structured video uses a temporal representation to maintain this synchronization among all the video objects involved in a structured video compositing. Under the structured video model, all objects are distributed all over the network, and are transported over to the compositing processor when needed. In such a distributed environment, it is important to study the issues concerning how the temporal relations are kept in a most efficient way. In this section, we first review some background of network delay and synchronization, and discuss a source control mechanism that can be used to maintain the synchronization among all video objects, and also minimize the buffer requirement at the receiver. Then we will discuss the global clock timing recovery that is needed for the structured video model.

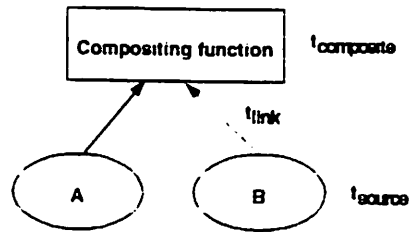


Fig. 4-7. Delays for structured video.

4.2.1 Synchronization background

In a distributed network, delays and delay jitters are inevitable due to the nature of the network. Delay is the duration between the time the signal is sent from the transmitter to the time when the receiver receives the signal. A long delay is not acceptable in applications that require fast response time to interactive operations. The delay jitter results from the changes of the delay. Delay varies due to reasons such as queueing, different routing, retransmission due to error, etc. Because of variable delay, buffers are used to keep the information that cannot be used immediately, and also regulate the traffic into a smoother pattern. The goal of synchronization is threefold: (1) to maintain the correct temporal relations among all video objects, (2) to minimize the overall delay and delay jitter at the receiver, and (3) to minimize the buffer size required at the receiver.

In the structured video model, we can model the delay of any link from video object to the compositing functions, as in the following equation. The corresponding graph of the structured video is shown in Fig. 4-7.

$$t_{delay} = t_{source} + t_{link} + t_{compositing} \quad (4-17)$$

In this equation, the t_{source} is the time used to generate the video object. It includes the query evaluation, seek, and access time (if it is saved in a storage device), the coding time (if it needs to be coded before it is sent to the compositing function), and the packetization time. The t_{link} is the time used to send the video object to the compositing function

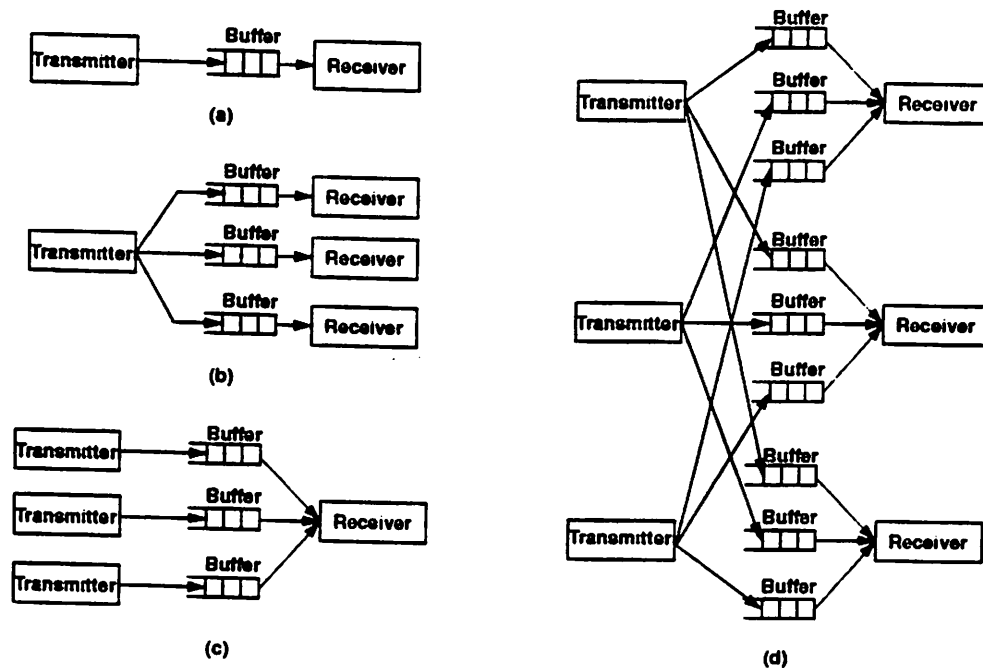


Fig. 4-8. 4 types of synchronization: (a) one-to-one. (b) one-to-many. (c) many-to-one. (d) many-to-many.

through the transport. It includes the transmission delay and the queuing delay (if it is buffered at any switch in the transport). The $t_{compositing}$ is the time used to compose the video object. It includes buffering delay, depacketization, decoding, and the compositing processing time. In all these different delays, we will make no assumption about the source coding, packetization/depacketization, the link or the compositing processing. The emphasis of our discussion is placed on how to reduce the buffer and buffering delay in the last term $t_{compositing}$.

There are basically four types of synchronization. They are one-to-one, one-to-many, many-to-one, and many-to many-synchronizations as shown in Fig. 4-8. The one-to-one type is the most basic that any communication link needs to consider. Among all the four types of synchronization, the many-to-one type of synchronization best fits into the structured video model in that many sources (transmitters) send their video objects to the compositing function (receiver) for compositing. All objects need to arrive at the same time in order to reduce the required buffer. The many-to-many is also possible when video objects

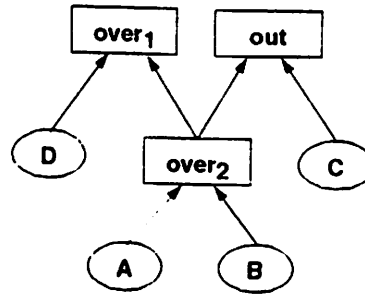


Fig. 4-9. Structured video model sharing the same composite object for a different compositing function.

are shared for some reason. Fig. 4-9 shows an example of a many-to-many type of synchronization. In this case the compositing result of object A and B ($over_2()$) is shared between two receivers ($over_1()$ and $out()$) to save the compositing resource. This case is different from the multiple many-to-one type of synchronization because only one copy of object A and B are sent out.

Two important issues need to be studied. The first is the clock rate matching between the transmitters and the receiver. When the clock rates are different, the buffer between the transmitter and the receiver will overflow or underflow. There are mechanisms for this rate matching. In the structured video model, we assume there is a global timing clock available. It is required that the structured video model use some kind of mechanism to maintain a global clock among all the sources, destinations, and any intermediate compositing processors so they all use a common clock and timing. In section 4.2.2 we will discuss this clock rate matching and global timing issue.

The second issue is the synchronization among all video objects sent to the same compositing function. This is required in both many-to-one and many-to-many types of synchronization, which are the cases that apply to structured video. With either type of synchronization, video objects should arrive at their destination just on time. If an object arrives too early as compared with other video objects to be composited, this video object should be kept in the buffer at the destination. Similarly, if it arrives too late, it keeps other video objects waiting. Under this situation, some kind of source flow control mechanism

is needed to maintain the best timing for video objects arriving at the destination to minimize the buffer requirement between the video sources and destinations. Without using the timing control mechanism, a very large buffer may be required because the data rate of full motion video sequences is usually very large. In section 4.2.3, we will discuss this source flow control mechanism.

4.2.2 Clock rate matching

4.2.2.1 Brief review

In a distributed system, there is usually no single clock available to all nodes on the network. Usually, each node operates on its own local clock. If the clock rate drifts from node to node, problems appear. For example, when the transmitter transmits a video stream at a rate slightly higher than the rate that the receiver can consume, the video stream accumulates at the buffer, and the buffer periodically overflows no matter how large the buffer size is. Similarly, when the transmitter transmits at a lower rate, the buffer periodically underflows, and the receiver is always waiting for the video stream data. There are basically four solutions that can be used to solve this clock rate discrepancy problem.

The first is so called “*slip buffering*.” The idea is not to change the clock at either the transmitter or receiver. Instead, the receiver throws away frames when the buffer overflows, and repeats the previous frame when the buffer underflows. In this way, the receiver can always catch up the clock rate of the transmitter. The disadvantage of this approach is the degradation of the video quality. For example, with a 0.01% drift in the crystal clock, a frame is slipped every 6.5 minutes, assuming the video sequence is 30 frames/sec. There is a goal in the MPEG community that no more than one glitch appear every 15 minutes¹. That is a much more stringent requirement because the “glitch” includes any single bit

¹. From ATM forum SAA group e-mail discussion.

error. In slip buffering, every time the buffer overflows, it is not a small glitch; it is a freeze or jump of the whole frame.

The second approach is to monitor the occupancy of the buffer at the receiver and send this information back to the transmitter to adjust its transmission rate accordingly. This mechanism, however, may not be useful for real time video when the delay on the link t_{link} is large. With such a large delay, the receiver already accumulates a lot of data before the transmitter begins to correct its rate. The buffer needed at the receiver may become too large. In the one-to-many type of synchronization, rate matching through monitoring the buffer is not viable because different receivers may run at a different rates, and the transmitter can only match the rate of one of the receivers.

The third approach is to send the time stamps together with the video sequence, and recover the clock rate at the receiver end. This approach does not need any feedback from the receiver, and is implementable for real-time video. MPEG I and II both use this approach to recover the clock at the receiver. However, this synchronization mechanism let a transmitter determine the clock by its own. When multiple transmitters are involved, different clocks are used. In this way, it does not work for the many-to-one type of synchronization because the receiver can only recover the clock rate of one transmitter. For structured video, many-to-one synchronization is most representative. Matching only the clock of one transmitter results in a sacrifice of video quality of other transmitters.

The fourth approach is to provide a global clock available to all components. This approach is appealing to applications with complicated structure such as the structured video model. Providing an exact global clock to all distributed components may not be easy, however, providing "approximate" global clocks based on the same reference clock, however, is possible. In the distributed computation system, a system using a global clock is called a *synchronous system*. In fact, a global clock can be implemented even in the presence of failures[63][64][65]. In the next subsection, we will describe our global clock approach for structured video.

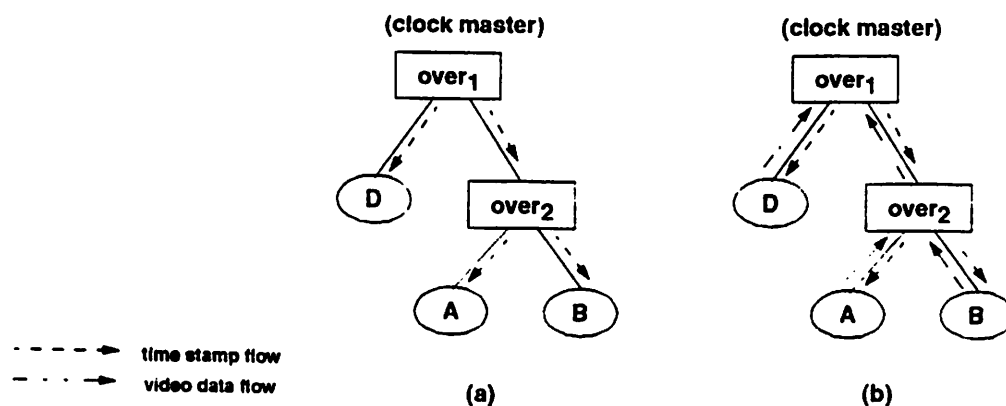


Fig. 4-10. Global clock time stamps flow in two steps: (a) establishment session and (b) compositing session.

4.2.2.2 Global clock for structured video

Our solution to synchronization for structured video is to provide a global clock available to all the components in the tree structure. By "global clock," is meant only global to the whole tree, not the whole network. The clock rate between different trees can be different. This is different from creating a totally synchronous distributed system in [63][64][65].

The global clock synchronization process includes two steps: the establishment session and the compositing session. The global clock must be provided before the normal compositing starts. To do this, an initial establishment session is used to exchange clock information before the normal compositing starts. In the establishment session, we assign the root compositing function of the tree as the clock master, as shown in Fig. 4-10. The clock master sends out time stamps through the tree structure to all the components in the tree. The time stamps are sent out continuously from the start of the establishment session until the end of the compositing session. All the components in the tree then recover a local clock based on this reference clock.

Note that this time stamp is not sent directly from the root to all the leaves. Instead, it is sent through the intermediate level of compositing functions, then to the leaves. In this way, the synchronization can be done from link to link in a more distributed manner, and

also the children nodes can maintain a better synchronization with their intermediate parent node. This mechanism can also cover the many-to-one synchronization, in which the clock master can be assigned to one of the root compositing functions in the structure. This clock master sends the time stamps to all the rest of the roots. Each root then sends the time stamps to all the components in its own tree. Note that this approach is different from the MPEG approach, in which the time stamps are embedded in the video signals. Our approach uses a separate channel for time stamps going in the opposite direction of the video signals. The extra channel may or may not use more communication resources. For example, in an ATM network, all the transports are in logical virtual channels. An extra channel does not cost anything except an extra identifier (VCI).

The rate of the reference clock can be chosen according to the need of applications. When possible, a lower rate can be used to save the bandwidth for transmission of the time stamps. This is depending on the application, though. For example, the reference clock rate in MPEG I is 90 KHz, while MPEG II uses a 27 MHz clock. The global clock derived in this way is only an approximation of the reference clock due to the different delays of time stamps (delay jitter). This approximated clock, however, will keep the buffers needed to a limited amount.

4.2.3 Many-to-one type synchronization in structured video

The second issue in synchronization is the arriving time matching among all the video objects of the same compositing function. Using more buffer at the receiver is not a good solution for this problem because of the immense amount of data involved in real time video. A better way is to do some kind of source control at the establishment session so that the video sources send out the data at the exact time to guarantee that data from all video sources arrive at the receiver at roughly the same time. To achieve this, it seems obvious that two parameters need to be decided. One is the relative difference between the origins of the time basis of the components. The other parameter is the delay measurement of the links in the tree. These two parameters are related to each other. To derive one

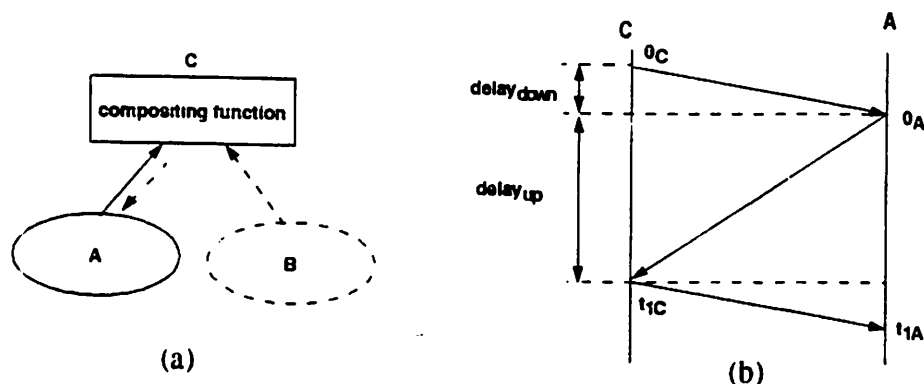


Fig. 4-11. Relative origin and delay determination. (a) Tree structure. (b) Time trace of the negotiation process.

parameter, we need to know the other, and vice versa. In this section, we show a very simple method to make video objects synchronous without knowing these two parameters.

Before we describe the detail, we will first make some assumptions about delay. In our discussion, we assume that delays are constant throughout the whole compositing process. Apparently, the source control to synchronize the arriving time handles only the first order characteristic of delay. It will not handle delay jitter. However, it is reasonable to make such an assumption on this source control model if the delay jitter issue is handled by the rate matching mechanism described in the previous section with some buffers. Secondly, we assume the delays in two directions on the same link may be different. This is a reasonable assumption because both directions may use different routing, and may undergo different traffic situations. Thirdly, we assume that the only unknown delay is the link delay. The processing delay such as coding/decoding, compositing, and accessing delays are all known to the local device. We can ignore such delays in our discussion. Taking them into consideration is easy since those delays are explicitly known.

Under these assumptions, we consider the model shown in Fig. 4-11. Here, we study synchronization on one link of Fig. 4-11(a), i.e., the link between C and A. A similar result applies to the link between C and B. Also note that in Fig. 4-11, component C may or may

not be the clock master. When it is not the clock master, it receives the reference clock from its parent and passes it on to the child nodes.

The global clock described in the previous section only gives all components a correct clock rate. It does not tell the transmitters when to start sending video. In fact, it is very common for every component to have its own origin for its time base, as is shown in Fig. 4-11(b). In this figure, 0_A and 0_C represent the origins of the time basis of A and C, respectively. In the figure, it is assumed that component A assigns an origin when it first receives the message (time stamps) from C at the origin of time basis of C. It is difficult to discover the discrepancy between these origins because the transmission delays between the two components are unknown. However, it is still possible to make objects synchronous without knowing both their time origin and their delay.

Take a closer look at Fig. 4-11(b). A packet sent from A at time 0_A in A's time basis arrives at the time t_{1C} of C's time basis. Similarly, any packet sent at time t_A in A's time basis will arrive at $t_A + t_{1C}$ in C's time basis. This is actually enough information for us to make video objects synchronous at the receiver. To make video objects synchronous, we only need to know the sending time in A's time basis and the arriving time in C's time basis. It does not matter what the sending time is in C's time basis or the arriving time in A's time basis. Therefore, whenever compositing function C requests a video object to arrive at time t_C in C's time basis, the video source A simply sends out the video object at time $(t_C - t_{1C})$ in A's time basis. This value t_{1C} can be taken as a logical delay (t_d) combining the effect across different time bases and link delay. Such a logical delay can be derived easily in the initial setup of the global clock in three passes of message passing, as shown in Fig. 4-11(b).

The overall tree synchronization can be done easily through the global clock recovery and the logical delay. To achieve synchronization, we need to derive the logical delay for every link in the whole tree. The process of the whole tree synchronization is as follows:

1. Initialize the global clock as described in the last section. All the components recover their clock base on the same global reference clock, so that all the components proceed at roughly the same rate.
2. Once the global clock is available, the components chose their origin of time basis as follows:
 - (i) Starting from the root, the root determines it own time origin arbitrarily.
 - (ii) Once the time origin of the root is determined, it passes messages to all its immediate children nodes.
 - (iii) The children node sets its origin of time basis when it receives this message from its parent. Then it sends a message back to its parent notifying its receiving of the message. At the same time, it sends messages to all its children nodes to initiate their time basis origin.
 - (iv) When the parent node receives the response from its children node, it records the arriving time of this message. This arriving time is the logical delay between itself and this children node. Then it sends this logical delay back to this children node.

The steps in 2.(iii) and 2.(iv) are performed recursively until all the components in the tree have been set up correctly. Once the time basis and the logical delay is determined at all components, the synchronization at the parent node can be easily maintained. Note that this process can be done in a fully distributive manner. It starts from the root, and spreads serially to all the leaves.

4.3 Conclusion

In this chapter, we discuss most of the important issues in the implementation of structured video compositing. In spatial compositing, we start with Porter and Duff's anti-aliasing algorithm, extend it, and derived the compositing algorithm of all the compositing functions that we listed in the previous chapter. We also study the implementation flexi-

bility of the structured video model in allocating video compositing functions to a distributed network environment. The structured video can provide a means for studying various possibilities of implementation, and provides as a tool for its performance analysis and optimization. To do this, we explored the generic structure of compositing functions and their useful properties such as *associative*, *commutative*, and *distributive*, which enable easy manipulations of the compositing functions and restructuring of the structured video representation of a video service and its associated implementation. More discussion regarding the actual mapping of the tree structure into a physical network configuration and the trade-offs can be found in [32].

In temporal compositing, we discuss two important synchronization issues, which are the clock rate matching and the multiple object synchronization. In rate matching, we propose to use a global timing clock for all the components in a composite video object. This kind of global clock mechanism uses separate channels for transmitting timing signals, and may use more network bandwidth. However, this bandwidth usage is small when compared with the bandwidth of full motion video sequences. With the complexity of a tree structure, it is difficult to use traditional methods such as slip buffering or the buffer monitoring method to match the clock rates among all the components. Since the goal of structured video is to support real-time full motion video, it is best to use a global clock to allow each component to run freely without other intervention. To maintain the synchronization among multiple video objects, we use a simple mechanism to set the origin of the time basis of all the components, and measure the delay of the links in the composite object structure.

In a distributed implementation, we did not study optimization of resource allocation in this thesis. We only study properties that enable more flexible resource allocation. The scheduling is another issue that needs to be solved. The temporal representation proposed in chapter 3 can be used as a foundation for scheduling. However, the scheduling can actually be more involved because structured video includes not only limited life span video

objects, but also undetermined life span objects and interactive operations. It is sure that some kind of dynamic scheduling scheme is needed. When the optimization considers both the resource allocation in the spatial domain and the scheduling in the temporal domain, it becomes a very challenging problem, and deserves further detailed study in the future.

CHAPTER 5

THE VIDEOSTATION — A PLATFORM FOR VIDEO COMPOSITING

Today's displays, such as the video subsystems of high-performance workstations, cannot implement the video compositing algorithm of structured video in real-time. In particular,

- Workstation displays are designed to support an-isochronous objects only, with the assumption that each pixel is updated only infrequently. Rectangular video windows can be placed on the screen using an analog RGB switch, which is very inflexible because it does not allow even simple operations like imposing a text label onto a video window.
- Televisions are designed under the assumption that only a single rectangular object is displayed, although they do update each pixel with each frame.

While workstations can implement structured video successfully in software, video compositing must be implemented in hardware with today's technology because of its high processing rates. In this section, we examine the nature of the limitations of today's display architectures. Then, as an example of an architecture that can support structured video compositing in real time with very few limitations, we present our VideoStation hardware architecture. As a demonstrative example, the VideoStation will cover only part of the compositing functions listed in chapter 3. We use a compositing algorithm modified from the algorithms in chapter 4 to cover the chosen compositing functions.

5.1 Objectives

Today's display systems can do a reasonable job with an-isochronous objects, in the sense of updating the display within reasonable human visual system response times, but are inadequate to handle isochronous objects without severe limitations. As a compositing display platform for structured video, we envision a display which can support compositing functions of any kind of video objects, including computer graphics as well as video in various combinations. Some useful display objectives, if they can be achieved at an affordable cost, would include the following:

- It can display a combination of isochronous and an-isochronous objects, with full update speed for isochronous objects.
- There should be no architecture-imposed limit on the number of arbitrarily-shaped video objects. More video objects can be supported by simply inserting more hardware.
- Parameters in compositing functions can be interactively changed in real time. For example, parameters such as the location (L) in the function *translation()*, the overlapping priority (z) of the function *over()*, and the transparent factor (τ) in the function *transparent()*, etc., can all be changed easily from frame to frame.

- A variety of compositing functions can be realized in real time. In our first implementation, we only implement four basic compositing functions in table 3-1, i.e. *over()*, *transparent()*, *translation()*, and *delay()*.
- The rest of the unary functions in table 3-1 can be performed individually on each video object before compositing, and therefore are not included in this compositing platform.

In essence, the implementation of compositing functions deals with the placement and combination of multiple video objects at proper spatial locations and temporal timing on the display according to the compositing algorithms as described in chapter 3. Since video objects are full motion objects with their location changing from frame to frame, the compositing processor must guarantee real time operation such that the location of the video objects appears correctly in every frame. This is difficult in conventional display because of the limitation of processing capability and memory bandwidth. The fact that only the window boundary shows up when we select and move a window in today's workstation reveals the limitation to supporting full motion videos in conventional display. In addition, the video objects require the support of compositing arbitrarily shaped objects, which requires more processing. To properly handle the compositing, we need a special design on the display architecture and implementation. In this chapter, we deal with these processing and memory bandwidth limitations and propose an architecture which enables real time compositing. Before presenting our architecture, we first review the technology limitations in today's video display technology.

5.2 Today's technology for video display

The single frame buffer display is the most popular display architecture used today as is shown in Fig. 5-1. In this architecture, any point on the display screen corresponds to a pixel value stored in the frame buffer. The frame buffer holds a 2-dimensional memory array module with its size equal to the size of the display. The video refresh controller constantly reads out the data stored in the frame buffer and repeatedly scans the CRT monitor

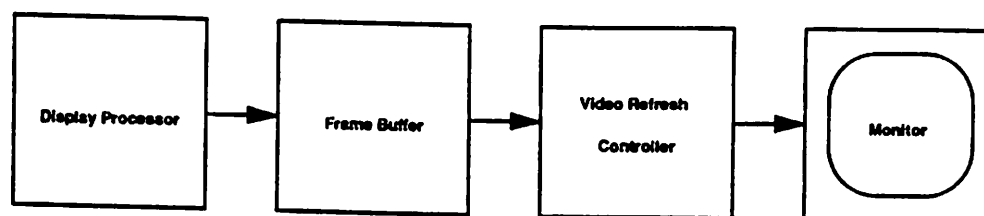


Fig. 5-1. Display Architecture with single frame buffer.

at a typical refresh rate of 30 times or 60 times per second. In this architecture, objects are rasterized and written into certain address of the frame buffer by the display processor to show up at the corresponding location on the display. The display processor can be either a general purpose CPU, or a special hardware which enhances the rasterization of frequently used objects such as lines, rectangles, etc. The frame buffer also records an image data structure that is used by compositing operations. For example, to perform a *translation()* operation, the display processor uses the image data structure to remove the original object from the frame buffer, and then write it into a new address.

This kind of single-frame buffer architecture is well suited to the display of an-isochronous video objects, such as graphics and texts, which are only updated very infrequently. Under this condition, most of the frame buffer bandwidth is used by the video refresh controller. To support full motion video compositing, it is apparent that the frame buffer bandwidth needs to be high enough to support the 30--frames-per-second updates of the video objects. The display processor needs also to be fast enough to process the compositing when the video objects changes from frame to frame. When the number of video objects involved in the compositing increases, the requirement on both frame memory bandwidth and processing capability becomes even more stringent.

It is the limitations of frame memory bandwidth and the display processor capability that prohibit the direct implementation of the structured video compositing algorithm in this kind of single-frame buffer architecture. There are actually some trade-offs between

the frame memory bandwidth requirements and display processor requirements. For example, to perform the *over()* function of two objects, we can either first write the object at the bottom into the frame buffer, and then write the object at the top to overwrite the portion at the bottom. Or we can write only the unobscured portion of the bottom object and top object into the frame buffer. The former approach requires more frame buffer bandwidth, but is easier done. The latter approach saves the frame buffer bandwidth usage but requires more capable display processor to calculate what is the obscured part of the bottom object — which may be very complicated. With this trade-off, it may be possible to apply this single frame buffer architecture directly if we have either very fast frame buffer, or a very fast processor. However, both frame memory bandwidth and processing capability are limited to support real-time full motion video compositing with today's technology, not to mention the compositing of arbitrarily shaped video objects. The next two subsections review the memory components and processors available as the candidates for implementing video compositing display.

5.2.1 Today's memory components

In traditional single frame buffer display systems, the limited memory bandwidth of the frame buffer is tolerable because of its asymmetric access pattern between read and write. Most of the memory access bandwidths are used by the video controller for refreshing. To support full-motion video objects, however, each pixel value is updated every 1/30 to 1/60 of a second. The access pattern becomes symmetric between read and write. In fact, it is quite possible that the write accesses to the memory be higher than the read ones in multiple video objects compositing. Under this situation, the bandwidths of the frame buffer are shared by the frame buffer update and the video refresh. To investi-

TABLE 5-1: Video timing for various video format

Visible Area Pixels x Lines	Pixel time in ns (30Hz refresh rate, interlaced)	Pixel Times in ns (60 Hz refresh rate, non-interlaced)
512 x 485	102.8	48.41

TABLE 5-1: Video timing for various video format

640 x 485	82.3	38.73
512 x 512	96.7	45.14
1024 x 768	32.37	16.52
1024 x 1024	22.57	11.42
1280 x 960	19.62	9.95
1280 x 1024	18.06	9.13

gate the bandwidth requirement on the frame buffer, we list the video timing for various video formats in table 5-1[57]. The number in the table shows the access requirement for the video refresh purpose only. We can take the full motion video bandwidth requirement on the frame buffer as two times the bandwidth listed in table 5-1 when the update access is comparable to the video refresh access. The typical specification for the capacity and

TABLE 5-2: Typical specifications of commercial memory products as of 1992.

Memory Type	Capacity	Configuration (Words X Bits)	Operating mode	Access time [†] (ns)	Cycle time [*] (ns)
DRAM	256K	262,144x1	Page mode	120	230
		65,536x4	Page mode	120	220
	1M	262,144x4	High speed page mode	80	160
	4M	1,048,576x4	High speed page mode	80	140
SRAM	16K	2,048x8	-	100	100
	64K	8,192x8	-	100	100
	256K	32,768x8	-	100	100
	1M	131,072x8	-	100	100
FIFO	4.5K	512x9	-	15	-
	9K	1024x9	-	15	-
	16K	256x36x2	-	-	25
VRAM	1M	276,480x4	for EDTV	50	60
	1M	189,360x8	for VCR	65	88

[†] Access time is the minimum time required for data to become valid at the output of the chip after the chip is addressed.

^{*} Cycle time is the minimum time between successive accesses to the same chip. Depending on the operating mode, there may be multiple words of access in one cycle time.

accessing speed of today's memory components (from data book of 1992) in commercial products is shown in table 5-2[66] The current existing memory components basically include dynamic RAM (DRAM), static RAM (SRAM), video RAM (VRAM), and first-in-first-out memory (FIFO).

Among all the memory components, DRAM is the most used in today's display system. It is the least expensive one, and can support more capacity than other types of memory component. The low cost makes it the most common one for a frame buffer. The access time of the DRAM, however, is also the longest. From table 5-2, we find that DRAM actually cannot keep up with the video refresh time of any video format listed in Table 5-1. To catch up with the pixel time, it is very common to use multiple chips so that multiple pixels are accessed simultaneously. A DRAM needs to periodically refresh the chip so that the data stored in the chip is not lost due to charge leakage. This is done by accessing every memory cell on the chip periodically. This chip refresh is different from video fresh. In video systems, the video refresh that accesses every location of memory for display can easily meet this chip refresh requirement. Therefore, chip refresh is usually not an important issue in a display system.

SRAM provides a faster cycle time than DRAM because it does not need to refresh the memory periodically. On the other hand, it consumes more power and the capacity is usually smaller. The cost is also much higher than with the DRAM. Considering that chip refresh is easily satisfied when DRAM is used as a frame buffer, SRAM does not provide too much advantage over DRAM when used as a frame buffer. However, SRAM is still faster and is still used in some high performance graphic stations. FIFO has the fastest access time with smallest capacity. It is too expensive to be used as a frame buffer. Its capacity is also too small. To be used as a frame buffer, many chips are needed since dis-

play systems usually require a large frame buffer. These limitations in capacity and cost prohibit the use of FIFO in any display system today.

VRAM is a memory specially designed for a frame buffer purpose. It is a dual-port DRAM that allows random access from one port, and serial access from the other one. A shift register is used at the serial port to shift out the data serially. To access through the serial port, a whole row of memory data is moved from DRAM to the shift register. Once the data resides in the shift register, it can be shifted out at a much faster cycle time than accessing the DRAM. At the same time, the random access port can still read/write data into the DRAM. The contention between two ports occurs only when the data is being moved from DRAM to the shift register. VRAM is very useful for a video refresh controller that requires periodical access of serial data. However, VRAM is designed mainly for graphics applications. The random access port bears a bandwidth much lower than the serial port. It cannot support the update access which is used as the frame buffer for full motion video display. The cost of VRAM lies between the SRAM and DRAM while its capacity is a little bit less than DRAM.

Comparing the figure from both table 5-1 and table 5-2, we find that DRAM cannot satisfy the pixel time requirement of any video format. Even SRAM, VRAM and FIFO can only barely make it, depending on the video format. To support full-motion video compositing, the update access requires a comparable amount of bandwidth, or even higher as listed in table 5-1. None of the memory components listed in table 5-2 except FIFO can support twice the bandwidth of the video refresh rate in table 5-1 to cover a comparable video update. Even the VRAM cannot support either. Another popular VRAM from Texas Instruments has a 33 MHz burst rate, which is marginally fast enough to handle one real-time video object but certainly not more[13]. As the number of full-motion objects that a system displays increases, the system's memory bandwidth requirement increases proportionally.

Today's memory technology advances in three directions — to improve the memory size, to improve the speed, and to lower the power consumption. The advances of size expansion have progressed much faster than the other two. Since today's frame buffer uses multiple parallel chips to enhance the bandwidth capability, the size improvement is actually a drawback because it reduce the number of chips needed for simultaneous access.

5.2.2 Today's display processors

The job of the display processor is to update the frame buffer. The major functions of the display processor are address generation, pixel block transfer (bitblt), window clipping, and object generation. Some of the functions not only involve write access to the frame buffer, but also read access. For example, a pixel block transfer includes read-modify-write operation. These functions can be performed by a general purpose processor (CPU), or with a separate display processor controlled by a CPU to accelerate the update speed. A general purpose CPU may be slow for all these functions, and is becoming rare today. Use of the separate display processor is most common.

The display processor used depends on the actual applications. Most of today's display processors are for graphics purposes. They accelerate the generation of a range of geometric objects, such as lines, circles, text characters, polygons, filled objects, and also the bitblt. The general purpose CPU uses simple I/O commands to instruct the display processor to generate a specific object and update the frame buffer. Today, a graphic display processor can be implemented in one single chip which includes all the video refresh controller, memory interface, and graphic generation function. These processors, however, usually only support graphics and texts in a non real-time manner.

With the speed pushed to the extreme, some of the graphic display processors can also support full-motion video overlay. This video card does not get the video stream from the CPU. Instead, it receives the video streams directly from the external interface. The dis-

play processor maintains a pixel-by-pixel mask (a single bit α value), and uses a special mechanism to filter the video stream through the mask, and overlay on other graphics.

This kind of video card is available today on both personal computers¹ and workstations². The weakness of these boards is twofold. First, they can overly only rectangular video windows. The display processor does not consider arbitrarily shaped video objects when it generates the mask. The screening mask is generated in a non real-time sense. The mask does not depend on the content of the video stream, and does not change frequently. When the shape of the video objects changes from frame to frame (which is required by the structured video), the mask also changes from frame to frame. The mask generation becomes a real-time job, and is not affordable with the display processor described above. The second drawback is the expandability problem. Using a single frame buffer with single display processor limits processing capability. This kind of architecture does not have the flexibility to be expanded to accommodate more video objects. For example, the XVideo board from Parallax can process only two video objects, and no more.

The nature of the video objects in structured video is more like graphic objects updating from frame to frame. To compose video objects, the display processor cannot simply put them on top of all the rest of graphic objects without considering their contents. Following the approach described above, the display processor needs to read in the content of the video objects and generate a mask for every video object from all the information read in. All this needs to be done in a single-frame period of time, only achievable through a multi-processors approach.

Thus, both memory bandwidths and display processors have limited capability to support structured video compositing. A multi-chip frame buffer and multi-processor display processor approach are required. With these considerations in mind, the next question is

¹. For example, the Video-Blaster board from Creative labs.

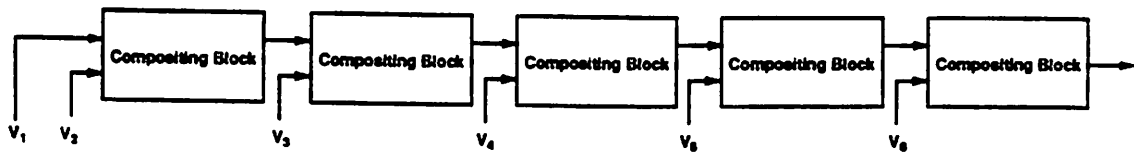
². For example, the Xvideo board from Parallax.

what architecture can best organize all the memory chips and the processors. In the next section, we will discuss this issue.

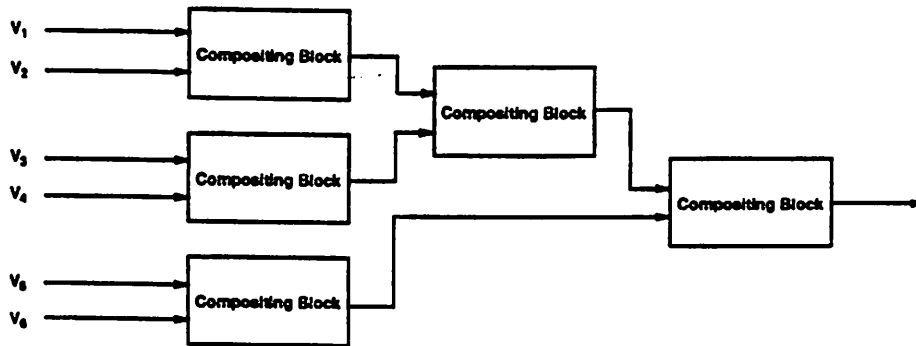
5.3 Architecture consideration

For real time video compositing, it is obvious that our display system must use an architecture with multiple hardware compositing modules — i.e., multiple compositing processors (or display processors) and memory modules to break the bottleneck described in previous sections. In this section, we discuss how the compositing function and/or the display processor can be decomposed into multi-processing modules and their architecture. There are basically three different ways to decompose. The first is *space division*, meaning that the space on the display screen is divided into multiple separate areas. Each area has an associated display processor and frame buffer to process the compositing of any portion of objects fallen into that area.[68][69][70]. The extreme of this approach is to associate a processor with each memory chip[69] or even a pixel[70] in the frame buffer. The second approach is the *function division*, meaning to divide the processing from the function domain. For example, in high speed graphic display, the function of transformation, polygon drawing, edge processing, etc., are pipelined and put into separate modules.[71]. The third way is *object division*, meaning to divide the processing into multiple modules with each module handling different video objects. Out of these three approaches, only the third is expandable to accommodate the increasing number of video objects involved in compositing. The first two approaches will reach their limits when the number of objects increases. For this reason, we will use the third approach in our Video-Station. It is also possible to apply *functional division* and *object division* simultaneously when many complicated compositing functions are used. That is, we decompose the processing into multiple modules so that every module handles only one compositing function and one video object.

There are two basic architectures base on the object division — linear array and tree-based architecture as shown in Fig. 5-2. Tree-based architecture covers many different



(a) A Linear Array Architecture



(b) A Tree-Based Compositing Architecture

Fig. 5-2. Two modular architectures for objects division compositing.

variations. Linear array is actually also a special case of tree-based architecture. We simply call any variation which is not linear array tree-based. These architectures decompose the compositing function into multiple binary compositing blocks. Each compositing block is a binary compositing function. We choose binary compositing function as basic compositing elements just for simplicity. In general cases, it is possible that a compositing block composes N objects instead of only two.

To compare the two architectures, we consider two factors: compositing image quality and complexity/cost. The image quality is basically the same for both cases as long as the same compositing algorithm is used, except for latency. The tree-based system performs compositing with less delay than the linear array because each input is processed by $O(\log(N))$ compositing stages rather than $O(N)$. However, the delay of each compositing stage is on the order of a multiple pixel duration when a fully pipelined implementation is used. So, both tree-based and linear array compositing systems should yield latencies

much shorter than a single-frame time. The cost includes the compositing processor complexity, the memory size, the I/O bus, and also the design cost. Simply comparing the number of compositing blocks used reveals that both architectures use the same number of blocks ($N-1$). However, this does not reflect the actual complexity/cost. Basically, the compositing block in linear array architecture is simpler than the tree-based architectures in terms of compositing processor complexity and the memory size. This is because one of the inputs of the compositing block in linear array is always a simple video object, instead of composite video objects as in the case of tree-based architecture. To combine a video object onto a composite video object can be made simpler than combining two composite video objects. For example, the tree-based architecture compositing block needs to buffer both inputs. In the linear array case, it is possible to feed one of the inputs directly from the previous stage without buffering by treating this input as a full-screen size rasterscan stream to make the synchronization easier, and therefore to save one buffer memory.

5.4 VideoStation design

5.4.1 An integrated compositing algorithm

To design the VideoStation base on the linear array architecture, it is important to formulate the compositing function in a way that best fits in the architecture. With object-based linear array, it is possible to have an extensible architecture that can accommodate more video objects by simply inserting more hardware, and also keeps the complexity low. In this section, we develop a compositing algorithm based on the compositing function algorithms described in chapter 4. We try to define one integrated algorithm that covers several compositing functions.

In our first prototype of the VideoStation, we only implement *over()*, *transparent()*, and the unary compositing function *translation()*. The discussion hereafter will focus only on the algorithms of these two binary compositing functions only. For simplicity, we introduce a parameter in addition to the 4 channels (r,g,b, α) used by Porter and Duff. It is the

object priority z , which is used simply to keep track of which objects cover other objects. The priority z is similar to the z value in z -buffering 3D rendering systems [4, 6], but our z value is simply a depth priority, not a distance. Also, our z value describes a whole video object, while the z value in z -buffering can change with every pixel in an object. Objects with larger z values will be composited in front of other objects with smaller z values.

The z value is also used to control the way in which video objects are composited. Instead of specifying explicitly what compositing function is to be performed, we assume that *transparent()* is performed whenever the z value of the operands are the same, otherwise *over()* is performed. Under this assumption, we will merge both *over()* and *transparent()* in equation 4-3 and 4-6 into the equation as follows.

$$\begin{cases} P_C = \frac{\alpha_A P_A + (1 - \alpha_A) \alpha_B P_B}{\alpha_A + (1 - \alpha_A) \alpha_B} \\ \alpha_C = \alpha_A + (1 - \alpha_A) \alpha_B \\ z_C = z_A \end{cases} \quad \begin{matrix} z_A > z_B \\ \\ \end{matrix}$$

$$\begin{cases} P_C = \frac{\alpha_A \tau_A P_A + \alpha_B \tau_B P_B}{\alpha_A \tau_A + \alpha_B \tau_B} \\ \alpha_C = 1 - (1 - \alpha_A) (1 - \alpha_B) \\ z_C = z_A = z_B \end{cases} \quad \text{if} \quad z_A = z_B \quad (5-1)$$

$$\begin{cases} P_C = \frac{\alpha_B P_B + (1 - \alpha_B) \alpha_A P_A}{\alpha_B + (1 - \alpha_B) \alpha_A} \\ \alpha_C = \alpha_B + (1 - \alpha_B) \alpha_A \\ z_C = z_B \end{cases} \quad \begin{matrix} z_A < z_B \\ \\ \end{matrix}$$

This equation can be easily extended to multiple objects. First we composite objects with equal z values as follows.

$$P_{z_i} = \frac{\sum_{j \in z_i} \alpha_j \tau_j P_j}{\sum_{j \in z_i} \alpha_j \tau_j} \quad (5-2)$$

$$\alpha_{z_i} = 1 - \prod_{j \in z_i} (1 - \alpha_j) \quad (5-3)$$

Then we composite these results.

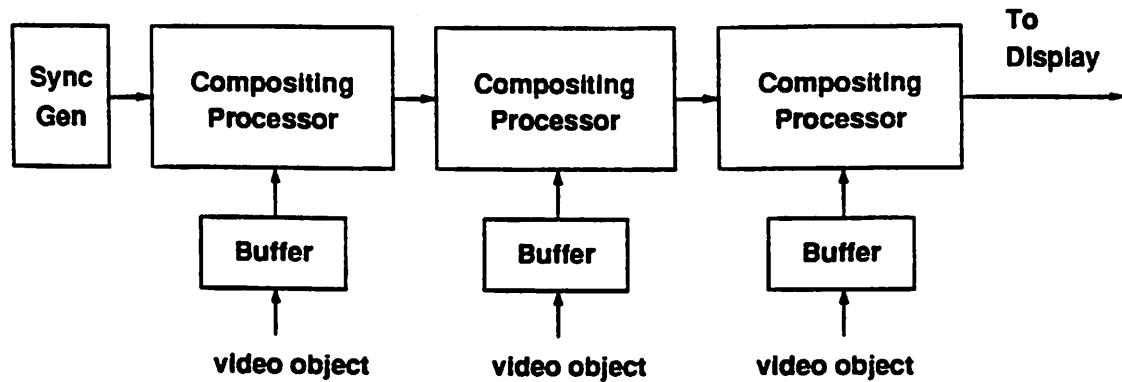


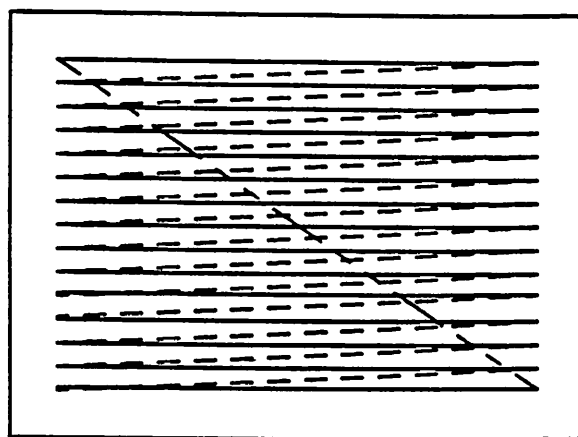
Fig. 5-3. A pipeline architecture for structured video compositing.

$$P_{composited} = \alpha_{z_1} P_{z_1} + (1 - \alpha_{z_1}) [\alpha_{z_2} P_{z_2} + (1 - \alpha_{z_2}) [\dots]] \quad (5-4)$$

5.4.2 Pipeline architecture

VideoStation uses a pipeline architecture based on the linear array. The compositing processor is decomposed into multiple processing units organized in a pipeline, as shown in Fig. 5-3. In this architecture, each pipeline stage consists of two parts — a compositing processor and a buffer memory. Each buffer stores a video object from its stage's input port, which is connected either to external networks or local video devices such as cameras, video disks, or graphics generators.

This architecture gets rid of the frame buffer which has fixed mapping to the pixels on the display. Instead of keeping the pixel structure in a physical frame buffer, it maintains the display pixel structure implicitly in the *raster scan* signals passing from stage to stage in the pipeline architecture. The *raster scan* is a pattern that the electron beam in a cathode ray tube (CRT) sweeps out to generate frames of 2-dimensional images through a series of horizontal lines moving from the top to the bottom of the image and from left to right on each line, as shown in Fig. 5-4. With raster scan, the pixels of any 2-dimensional images are arranged into 1-dimensional pixel streams with some additional synchronization signals specifying the horizontal and vertical retracing. To maintain the pixel structure implicitly, the pipeline architecture uses a synchronization generation unit (SYNC GEN in



——— SCAN LINE
 - - - - HORIZONTAL RETRACE
 - - - - VERTICAL RETRACE

Fig. 5-4. The typical raster scan pattern used in cathode ray tube (CRT).

Fig. 5-3) at the first stage of the pipeline to continuously generate the raster scan synchronization signals —vertical and horizontal retrace signals. This raster scan signal is then passed down the pipeline from stage to stage, so that every module is able to maintain the pixel structure correctly without having to access some frame buffer. When this raster scan signal goes down the pipeline stages, each compositing module performs the compositing by inserting video objects into the raster scan data stream at some correct timing. In this process, to place a video object at a certain location on the display is a matter of inserting the video object at the corresponding timing in the raster scan stream. In order to insert the video objects at any location (therefore at any arbitrary timing), some buffer is required at each stage to keep the video objects. A *double buffering* approach is required so that when one buffer is used for video object input, the other buffer can be used for simultaneous output. The detailed operation of the compositing processor module is shown in Fig. 5-5.

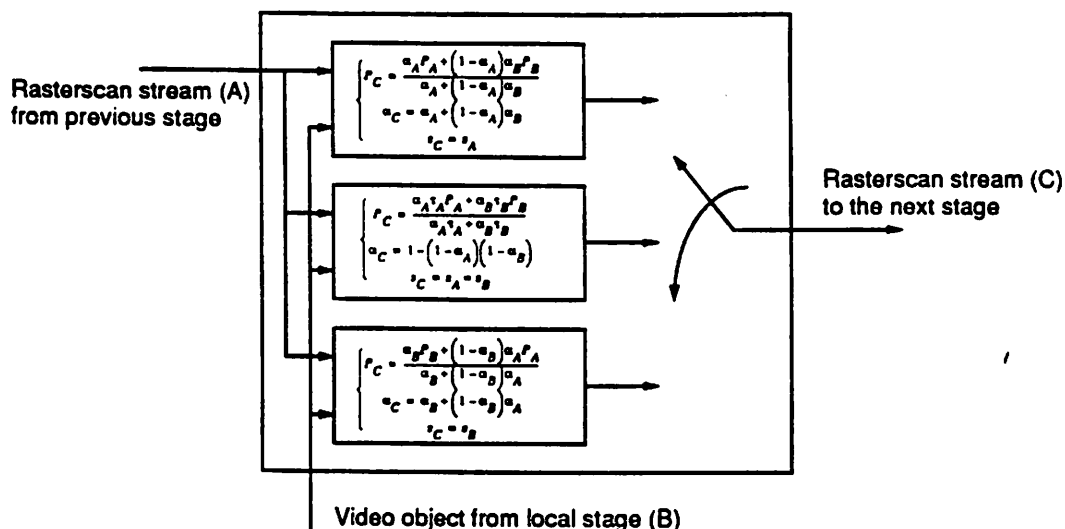


Fig. 5-5. Compositing processor module in a pipeline stage

This architecture shows a feasible way of distributing processing into multiple modules and also of avoiding the single big frame buffer, thus breaking display processing and memory bandwidth bottlenecks. Using the raster scan signal as the implicit pixel structure has several advantages. First, it avoids the single big frame buffer which is the source of the memory bandwidth bottleneck. With a traditional frame buffer approach, video objects are put into the frame buffer simply to place it at some correct location in the display pixel structure. For this simple purpose, all the video objects must go in and out of the same frame buffer, and therefore require high bandwidth on the frame buffer. With our approach, the pixel structure is not associated with a memory device. It is embedded implicitly in the raster scan signal. The memory bandwidth requirement is therefore much less. In pipeline architecture, we don't have a frame buffer. But we still need some buffer at every stage to regulate the input/output traffic of video objects. The memory bandwidth needs to support only the I/O of the video objects. This bandwidth requirement is quite different from the frame buffer case, in which the data need to be read out for display continuously.

Secondly, some compositing operations can be performed on the fly without cost. For example, to perform a translation operation in a traditional frame buffer approach, we need first to remove the object from the frame buffer, then write the object into a new address. With our pipeline approach, we don't need to remove the object from the frame buffer because there are no precomposed data stored in any memory device. All the compositing operations are performed dynamically. To change the location of a video object, we simply change the parameter in the compositing processor, where the change will be performed by inserting the video object at the new timing corresponding to the new position.

Third, the buffer memory usage is more efficient than the single frame buffer approach. In our pipeline architecture, each buffer memory need not be large enough to store an entire screen of video data as would a frame buffer. Instead, each must hold only the area where its video object exists. The total sum of all the buffer memory can be less than the size of entire screen when the video objects occupy only a portion of the display screen; the total buffer size can be smaller than the screen size. Also, since the buffer is not directly mapped to the physical size or location of the display, the buffer can be arranged flexibly to accommodate various sizes of objects. For example, a 10 K pixels buffer can be used to hold objects of various sizes, such as 100x100 object, 200x50, 500x20, etc. There are no presumptions about the shape of video objects, as long as the size of the video object can be fit into the buffer. If some video object is larger than one single buffer can hold, the video object can be split up and stored in different stages. Also, if a system needs to composite many small video objects, it can process several nonoverlapping objects in a single stage.

5.4.3 A modified approach to avoid presorting

The compositing algorithm described in the previous section has some limitations. Apparently, the multiple objects in equation (5-2), (5-3), and (5-4) must be sorted according to the priority value (z) in advance in order to be performed correctly. When we

modularize the compositing function into the pipelined architecture, we also assume that all the video objects are sorted in advance. The reason why compositing order must conform to the object priorities is that the composited pixels are the sum of pixels of two input video objects. Once two objects are composited, it is hard to insert the third object between them because the pixel data from each single object are lost, and only the composited pixel data are available. Sorting the video objects in advance prohibits the possibility of inserting the third video object into any two composited objects.

Presorting video objects, however, is not preferable. First of all, it is expensive. To sort video objects by priority is possible (with a self-routing network for example) but would require more processing than to composite them, and to sort the objects would limit the ability of this architecture to change objects' priorities or dynamically add new objects. Secondly, when the compositing is performed distributed over the network, the presorting problem becomes even more serious. Every time the object priority changes, the network connections must be closed, the compositing function mapping on the network must be redone, and then the new connections must be re-established according to the mapping.

One brute force method to avoid sorting video objects by priority is to have each pipeline stage pass both its composited result and also its local video signal to the next stage. The local object information can be used in later stages to undo the compositing if a video object at a later stage is found to have a priority value between that of the composited pixel and the local pixel. However, this solution requires that the number of video objects passed down the pipeline becomes larger further down the pipeline. The complexity of systems based on this method would increase rapidly with the number of video objects to be composited. This solution increases the compositing processing load of the downstream pipeline stages, which might result in the need to undo several compositings and then do several compositings again to derive a correct result. The extreme case of this solution is that all the video objects are passed to the last stage which does all of the compositing itself.

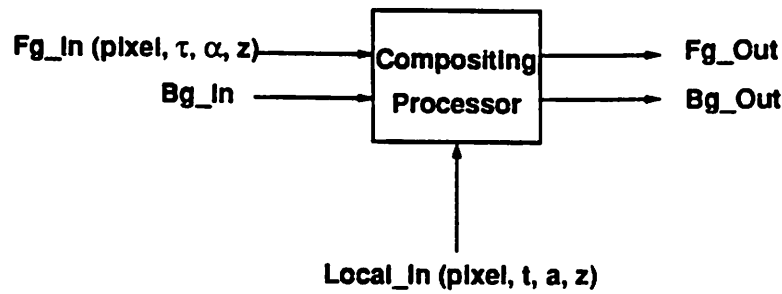


Fig. 5-6. A three-object compositing stage.

The solution we propose is a *three-object* compositing algorithm — in each pipeline stage we composite a foreground, background, and local object. As shown in Fig. 5-6, we pass two raster scan streams down the pipeline instead of one. The two streams passed down consist of video objects with the two highest priorities. The one with largest z is called the foreground stream and the one with second largest z is called the background stream; these two are not composited together until the last pipeline stage. At the pipeline stage, the compositing processor compares the priority of the local object with the priorities of the foreground and background streams. If a pipeline stage's local object has a lower z value than both the foreground and background streams, then the local object is discarded. If a local object's z value is between those of the foreground and background streams, then the background object is replaced with the local object. If a local object's z value is higher than those of the foreground and background objects, then the background object is replaced with the foreground object and the foreground is replaced with the local object. At the last stage, the final display result is derived by combining the background stream into the foreground.

This solution changes the algorithm described in equation (5-2) to (5-4) somewhat. Previously we assumed that any combination of K overlapping video objects was composited with the same linear equation. With the three-object compositing method we assume that for any number K overlapping objects, the linear function for compositing the objects only gives non-zero weight to the two objects with highest priorities. We have simulated

both the original algorithm and three-object compositing algorithms and have found that images produced with the modified algorithm are perceptually indistinguishable from images composited with the original algorithm. In fact, only the few pixels that lie on the crossing points of the boundaries of three or more video objects give different results from the two algorithms.

To simplify the design, the parameters passing between pipeline stages are not the final pixel and α values. Instead, we define several parameters as:

$$P' = \sum_{j \in z_i} \alpha_j \tau_j P_j \quad (5-5)$$

$$\tau' = \sum_{j \in z_i} \alpha_j \tau_j \quad (5-6)$$

$$\alpha' = \prod_{j \in z_i} (1 - \alpha) \quad (5-7)$$

With these definitions, the equations for each pipeline stage are:

$$P'_{fg, out} = \begin{cases} P'_{fg, in} & z_{fg, in} > z_{local} \\ P'_{fg, in} + \alpha_{local} \tau_{local} P_{local} & \text{if } z_{fg, in} = z_{local} \\ \alpha_{local} \tau_{local} P_{local} & z_{fg, in} < z_{local} \end{cases} \quad (5-8)$$

$$\tau'_{fg, out} = \begin{cases} \tau'_{fg, in} & z_{fg, in} > z_{local} \\ \tau'_{fg, in} + \alpha_{local} \tau_{local} & \text{if } z_{fg, in} = z_{local} \\ \alpha_{local} \tau_{local} & z_{fg, in} < z_{local} \end{cases} \quad (5-9)$$

$$\alpha'_{fg, out} = \begin{cases} \alpha'_{fg, in} & z_{fg, in} > z_{local} \\ \alpha'_{fg, in} (1 - \alpha_{local}) & \text{if } z_{fg, in} = z_{local} \\ 1 - \alpha_{local} & z_{fg, in} < z_{local} \end{cases} \quad (5-10)$$

$$z_{fg, out} = \begin{cases} z_{fg, in} & \text{if } z_{fg, in} \geq z_{local} \\ z_{local} & \text{if } z_{fg, in} < z_{local} \end{cases} \quad (5-11)$$

$$P'_{bg, out} = \begin{cases} P'_{fg, in} & z_{fg, in} < z_{local} \\ P'_{bg, in} & z_{fg, in} = z_{local} \\ \alpha_{local} \tau_{local} P_{local} & \text{if } z_{fg, in} > z_{local} > z_{bg, in} \\ P'_{bg, in} + \alpha_{local} \tau_{local} P_{local} & z_{bg, in} = z_{local} \\ P'_{bg, in} & z_{bg, in} > z_{local} \end{cases} \quad (5-12)$$

$$\tau'_{bg, out} = \begin{cases} \tau'_{fg, in} & z_{fg, in} < z_{local} \\ \tau'_{bg, in} & z_{fg, in} = z_{local} \\ \alpha_{local} \tau_{local} & \text{if } z_{fg, in} > z_{local} > z_{bg, in} \\ \tau'_{bg, in} + \alpha_{local} \tau_{local} & z_{bg, in} = z_{local} \\ \tau'_{bg, in} & z_{bg, in} > z_{local} \end{cases} \quad (5-13)$$

$$\alpha'_{bg, out} = \begin{cases} \alpha'_{fg, in} & z_{fg, in} < z_{local} \\ \alpha'_{bg, in} & z_{fg, in} = z_{local} \\ 1 - \alpha_{local} & \text{if } z_{fg, in} > z_{local} > z_{bg, in} \\ \alpha'_{bg, in} (1 - \alpha_{local}) & z_{bg, in} = z_{local} \\ \alpha'_{bg, in} & z_{bg, in} > z_{local} \end{cases} \quad (5-14)$$

$$z_{bg, out} = \begin{cases} z_{fg, in} & z_{fg, in} < z_{local} \\ z_{bg, in} & \text{if } z_{fg, in} = z_{local} \\ z_{local} & z_{fg, in} > z_{local} \geq z_{bg, in} \\ \alpha'_{bg, in} & z_{bg, in} > z_{local} \end{cases} \quad (5-15)$$

Using the foreground and background objects, the final display result can be derived at the last stage of the pipeline using the following equation:

$$P_{composed} = (1 - \alpha'_{fg, out}) \frac{P'_{fg, out}}{\tau'_{fg, out}} + \alpha'_{fg, out} \frac{P'_{bg, out}}{\tau'_{bg, out}} \quad (5-16)$$

The implementation of equation (5-8) to (5-15) is much more efficient than using direct implementation as shown in the 2-object compositing case in the last section. Using the direct implementation in the last section, even the 2-object compositing requires at least 5 multiplications, 5 additions, and 1 division in every stage. With the optimization in the parameters passed between stages, a stage can be implemented in 3 multiplications, 5 additions, and no division. The only division operation is in the last stage where the foreground and background objects are combined. This reduces the implementation complexity considerably in the pipeline stages. Also, equation (5-14) is actually not necessary because only the two highest priority objects are kept. This is reflected by the fact that $\alpha'_{bg, out}$ does not show up in equation (5-16). One more thing to note is that even though the priority value τ is a per-object based parameter, it becomes a per-pixel based parameter and must be passed from stage to stage because both the foreground and background result includes pixels from objects of different priorities. The cost is the extra channel required between the stages. The size of the memory buffer of the pipeline stage is not affected because the priority value of the local video object is still per-object based.

5.5 Conclusion

In this chapter, we describe the design detail of a video compositing platform - Video-Station that supports real time compositing for structured video. We first reviewed the bottlenecks for the implementation with today's technology. There are mainly two of these bottlenecks — the memory bandwidth and the processing capability bottlenecks. Today's memory devices advance every year mainly in capacity rather than in speed or bandwidth. This does not help to provide the high memory bandwidth that is required by most video applications. On the contrary, it actually makes the problem more serious because a larger size of memory is put into a package that has the same limited bandwidth. It is very common in traditional graphics display to use multiple memory chips in parallel to provide sufficient bandwidth. This approach is less efficient with the larger size memories. With full motion video, the memory bandwidth is much more critical than the graphics

bandwidth because the full motion video requires balanced read/write access to the frame buffer. A traditional frame buffer using vertical/horizontal retrace for write update is not enough for full motion video update. With the increase in the number of objects involved in compositing, the write update bandwidth needed may even be higher than the read operation. The display processor used by traditional graphic stations is also not well suited for full motion video display. Most display processors today optimize some specific operations, such as bitblt operations. More advanced processors enhance the performance by providing a pipeline of various functions, such as transformation, clipping, etc. This approach can only enhance the processing capability to some extent. The architecture is not expansible to accommodate more objects when the number of objects increases.

We propose a pipeline architecture for the VideoStation. The basic idea is to get rid of the single frame buffer so that there is no more fix mapping between the memory address and the display physical location. The advantage of this is to provide more flexible mapping between the memory address and the display, thus making the frame buffer usage more efficient. The translations such as bitblt can be performed on the fly without any difficulties. It also allows a much easier method of expansion to accommodate more video objects. The buffer memory required at each stage need not to be as large as the whole frame buffer. Instead, the buffer memory needs only to be large enough to hold the local video object. To avoid the presorting requirement in the compositing, we also proposed a 3-object compositing mechanism which does not perform the actual compositing until the last stage. The function of each pipeline stage is to select the two highest priority objects and pass down to the next stage. With this 3-object compositing, any dynamic changing of priority values can be done without difficulties. By choosing proper parameters passing between the stages, we also make the operations in each pipeline stage as simple as possible. As compared with direct implementation, which requires division operations in every stage, the optimized approach pushes the division to the last stage. All the rest of the stages need only addition/multiplication, and some simple logic operations.

CHAPTER 6

VEOSTATION PROTOTYPE

We have designed a prototype of the VideoStation architecture, to understand better its hardware complexity and to demonstrate the implementation of the structured video algorithms. In this chapter, we discuss the prototypes that we have implemented. We also study various implementation alternatives available today for real-time video applications. There are two basic ways for full-motion real-time video implementations. One is to use programmable video signal processors with software coding. The other is to use fully custom design hardware. We will discuss both approaches in this chapter.

Programmable video signal processors have advantages of more flexibility in design, fast prototyping, easy debugging, easy modification, etc. There are currently some video signal processors available for real-time video processing[72][73][75]. Most of them are designed for some specific applications, especially video compression algorithms. For example, Integrated Information Technology (IIT) developed a VCP chip[72] for discrete cosine transformation (DCT) based video compression/decompression. NEC also developed a VSP chip[73] for motion compensation. Among these, the programmable video signal processor(VSP)[75][76] produced by Philip is aimed for general purpose video signal processing. It uses an architecture that allows parallel processors to expand the pro-

cessing capability of the chips. We have designed a prototype of VideoStation using the Philip VSP processor. The prototype uses 8 VSP chips to implement one pipeline stage of VideoStation. In this chapter, we will describe the Philip VSP processor in detail, then we describe the implementation of VideoStation using the Philip VSP. Through our design, we also discuss the strength and weakness of the Philip VSP chip and possible improvement of the Philip VSP as a general purpose video signal processor.

The fully custom hardware design is possible with the aid of today's VLSI computer aided design (CAD) tools. We have designed an application specific integrated circuit (ASIC) for the core of the VideoStation. The chip is designed with Berkeley LagerIV CAD tools using 1.6 μm CMOS technology. In this approach, the function of 8 VSPs and many control and glue logics used in the previous approach is put into one single ASIC chip. This reduces the system complexity a lot.

The organization of this chapter is as follows. We describe the system perspective of the prototype system in the first section. In the second section, the programmable approach of our implementation is described. The third section shows our implementation using the ASIC approaches. In the last section, we give a brief conclusion.

6.1 System Perspective

In this section, we describe the system aspects of the VideoStation prototype. The prototype uses the pipeline architecture shown in Fig. 5-3 with each pipeline stage performing the 3-object video compositing operations described in Fig. 5-6 and equation (5-5) through (5-15). A schematic of the VideoStation hardware is shown in figure 6-1. The VideoStation consists of two different pipeline stages. One is the graphic stage for an-isochronous video objects, the other is the real-time video stage for isochronous video objects. Figure 6-1 shows a VideoStation with one graphic stage and two video stages. The number of graphics and realtime video stages is determined by the number of graphic and realtime video objects to be composited. More pinepline stages can be easily inserted

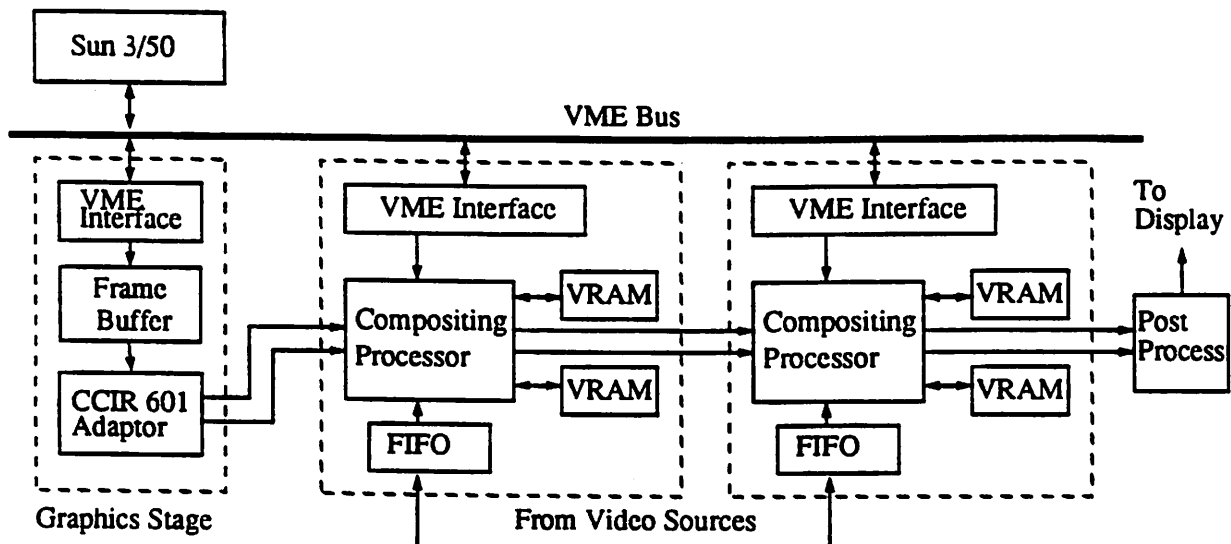


Fig. 6-1. System Diagram of the VideoStation system.

in the pipeline architecture to support more video objects. All the pipeline stages are connected to a master computer through a VME bus for control purpose. In the following, we will describe the graphic stage, the realtime video stages and the control, separately.

6.1.1 Graphic Stages

The first stage in Fig. 6-1 is a graphic stage that holds all an-isochronous video objects such as text and graphics video objects. The graphic stage performs two functions. As the first stage in the pipeline, it generates the video synchronization clock signals and provide it for use of the subsequent stages. Secondly, it performs the compositing operations of all an-isochronous graphic and text objects according to the 3-object compositing algorithm to generate the foreground and background objects to be passed down the pipeline stages.

An-isochronous video objects, by definition, have no hard time limit on when they must be shown on the display. Also they usually change only very infrequently. Therefore, the memory bandwidth and the compositing processing requirements are very low. One frame buffer with a single CPU is usually sufficient to handle all the graphics and texts shown on the display in one stage. The video compositing operations can all be done through software by the CPU. This is exactly the way implemented in today's workstation

to handle graphics and texts. In case the number of graphic objects become over the capability of the single frame buffer and the single CPU, it is still possible to add more graphic stages with frame buffer and CPU into the pipeline architecture to increase the bandwidth and the processing capability.

The implementation of the graphic stage is quite similar to the traditional frame buffer display. The difference is that the output of the frame buffer is not a completely composited video stream. Instead, it generates the foreground and background video object components as required by the 3-object compositing algorithm. To do this, the single frame buffer is modified into two frame buffers to accommodate both foreground and background video objects. Each frame buffer stores four components of video objects: the pixel value p , the α value, the priority value z , and the transparent value τ . The frame buffer is updated using the 3-object compositing algorithm described in the last chapter by the CPU in the SUN workstation through the VME bus. The frame buffer update can be done asynchronously. The output of the frame buffer, however, is read out synchronously at the 27 MHz clock rate. The video synchronization signals such as vertical and horizontal blanks and headers are embedded in the foreground video stream following the CCIR 601 video format by a display processor. The detailed diagram of a graphic stage is shown in Fig. 6-2.

6.1.2 Isochronous object stage

The second type of pipeline stage is for isochronous video objects. For isochronous video objects, the compositing operations are performed in real-time at 30 frames per second. A functional diagram of the isochronous object stage is shown in Fig. 6-3. In isochronous stages, the local object is composited with the foreground and background video objects in a synchronous sense that objects can be updated only between the frame period of 1/30 second. In order to keep this synchronization, frame sync control signals are generated from the foreground pixel stream in the compositing processor block. Also, a FIFO is used to buffer the local object from the input such that it can be synchronized with the

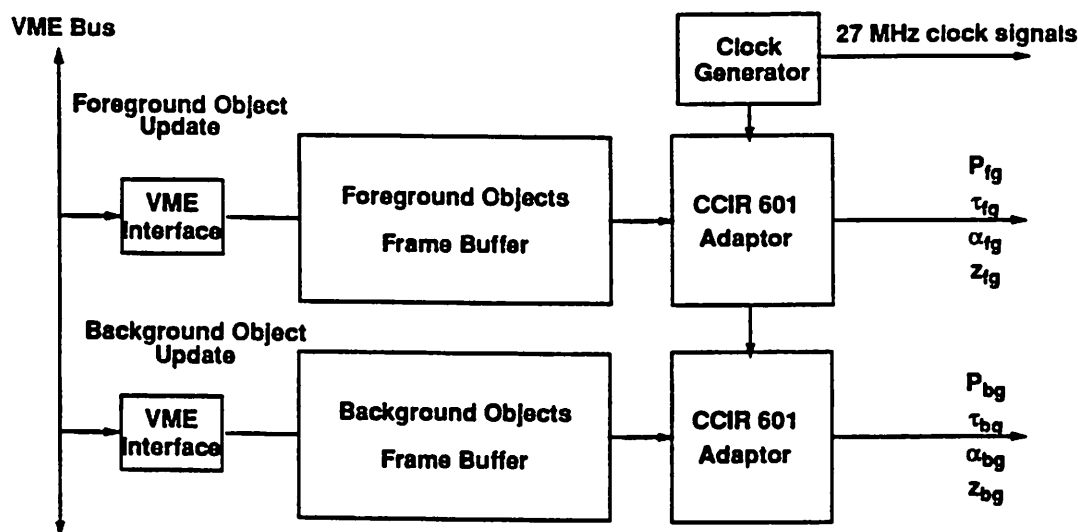


Fig. 6-2. The graphic stage implementation

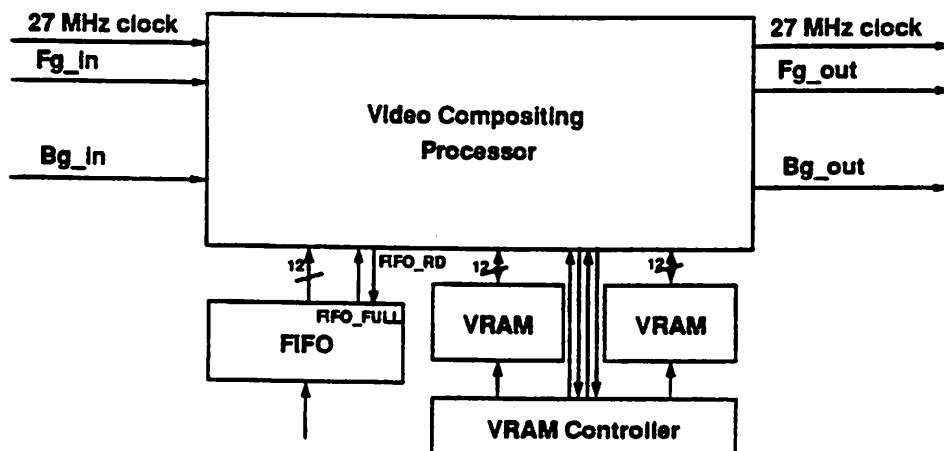


Fig. 6-3. The isochronous object stage functional block.

recovered frame sync control signals. A double buffer (VRAMs) scheme is used in this design such that when one VRAM is storing the data from the FIFO, the other one can output the data into the compositing processor. In this design, , the isochronous object stage can only support one video object from the input to be composited with the foreground and background video stream. To allow multiple objects per stage, a more complicated control and memory management is needed.

All the compositing operations are performed in the *video compositing processor* (VCP) block. The detailed block diagram of the VCP is shown in Fig. 6-4. It consists of

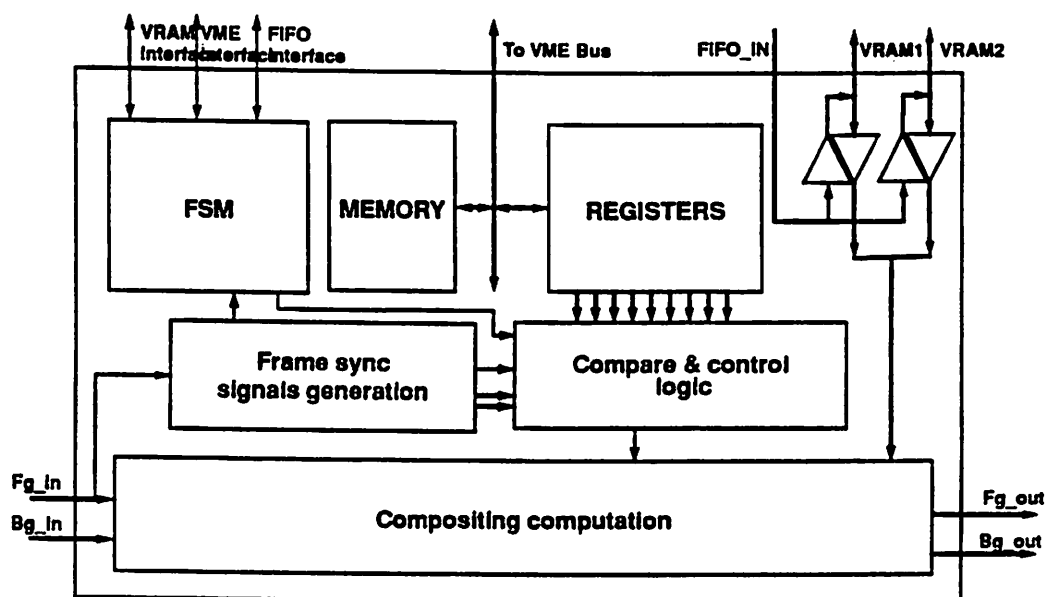


Fig. 6-4. The video compositing processor block diagram

several blocks: a compositing computation block, a frame sync signals generation block, a compare and control logic block, a finite finite state machine (FSM), a memory module, and some registers. The compositing computation block performs all the compositing calculations over the foreground and background input video streams according to the equations shown in (5-5) through (5-15). The frame sync signals generation block generates the video frame synchronization signals such as start of frame signal, start of vertical blank signal and start of horizontal signal from the foreground video stream. It also generates the current raster scan location address for use in the compare & control logic block. The compare & control logic block uses the current raster scan location and other compositing parameters, together with the parameters of the local video object to generate the control signals to instruct the compositing computation block to perform proper calculation. The finite state machine coordinates the operations among all the blocks in the VCP. In addition, it also coordinates the read/write operations between the FIFO, the two VRAMs, and VME bus interface.

Our portotype design effort emphasizes on the isochronous object stage. In implementation, the VME bus arbitration and VRAM interface is handled with one Xilinx 6400-

gate field-programmable logic device (FPLD) per stage. The VCP is a more complicated part. We have two alternatives for the implementation of the compositing processors in pipeline stage. One is to use the programmable video signal processors, the other is to design a application specific integrated circuit (ASIC). Both methods are able to provide real time video compositing capability. A detailed description of both implementations will be described in later sections.

6.1.3 Control

All the pipeline stages are connected and controlled by a Sun workstation through a VME bus. The main function of the Sun workstation is to keep track and update the compositing parameters of every video objects, such as the object size, location, priority, and transparency. The Sun workstation also provide users with an interface to interactively change the way the compositing is performed, such as to move a video object, to modify video objects' compositing parameters z and τ . Once a request is sent to the Sun workstation, it will update the parameters through the VME bus accordingly.

Note that the Sun workstation is performing real-time control operations over all the pipeline stages. All read/modify operations of the compositing parameters should be performed in every 1/30 second. Except these real-time control operation, the Sun workstation does not perform any real-time compositing operation. The VME bus does not send any of the real-time video objects pixels either. Under this condition, a single CPU with a VME bus is sufficient for all the operations. In the case of graphic stages that use the Sun workstation as the compositing engine, the Sun workstation loads all the graphics and text pixels according to the compositing algorithm through the VME bus into the graphic stage. Since these are none real-time operations, these operations can be given a lower priority than the real-time control such that all the real-time control operations are done in time.

6.2 Compositing processor implementation with programmable VSP

Use of programmable video signal processors (VSP) have several advantages. From the design perspective, it provides an easier way to implement an experimental system. It is easy to debug and modify the programmed algorithms. The design time is also much shorter. The disadvantage is less flexibility in the design, more glue logic, less compact, and higher cost as compared with a full customized designed hardware or application specific integrated circuit (ASIC) chip. From the processing capability perspective, a programmable processor is more efficient in the implementation of complicated, control intensive algorithms. A straightforward discrete transformation algorithm, for example, may be implemented efficiently with an ASIC as well as a programmable chip. A very complicated control algorithm, however, can be programmed in a programmable processor much easier than in ASIC design.

6.2.1 Programmable VSP overview

Today, there are basically two classes of programmable processors available for video signal processing. One class is designed for some pre-specified applications. The other is for general purpose video signal processing. For example, IIT's VCP chip[72] is designed for discrete cosine based video compression algorithms. The NEC's VSP chip[73] is for motion compensation purpose. These processors fall in the first class. With pre-specified applications, the processors are optimized to the applications by providing modules specific for the applications. For example, the IIT VCP chip has modules for Huffman decoding/encoding. The processor architecture is also designed such that some functions are most easily mapped into the processor.

The processors in the second class support general purpose video signal processing. They can only be optimized according to the general characteristics of video signals. For example, all video streams consume a lot of bandwidth, and need a lot of processing power. They can usually be parallelized very easily because of the large arrays of pixels in each frame. Therefore the processors in this class can be designed using heavily parallelized

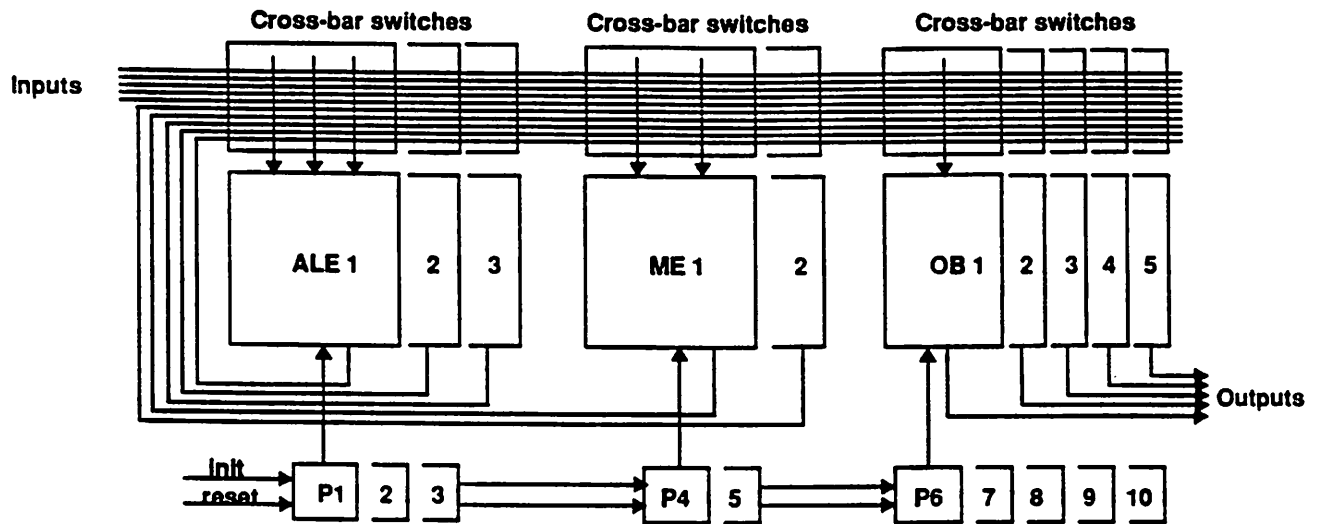


Fig. 6-5. Philip's VSP architecture

architecture to provide the required bandwidth and processing capability. Some generic functions are needed for any video signal processing, such as the frame data collection, frame sync recovery, etc. These functions can also be put into the video signal processor to ease the use of the processor. The Philip's VSP chip[75][76] falls into this category. Whether a particular processor is good or not depends a lot on the application to be implemented on the processor. In our implementation we choose Philip's VSP, which we will describe in the next section.

6.2.2 Philips architecture.

The Philip VSP chip[75][76] is an all digital, general purpose programmable processor chip designed for real-time video signal processing. A VSP chip contains a number of processing elements that operate in parallel. A complete VSP system can contain many of such chips. The architecture of the VSP architecture is shown in Fig. 6-5. The Philips VSP chip consists of several basic components: 3 arithmetic and logic elements (ALEs) and 2 memory elements (MEs), 5 output buffers(OBs), 5 input ports (12 bit wide each), and 5 output ports. The ALEs performs generic arithmetic and logic operations such as addition, subtraction, multiplication, and, or, and compare operations. The MEs are memory modules of size 512 words by 12 bits. They are mainly used to provide the table

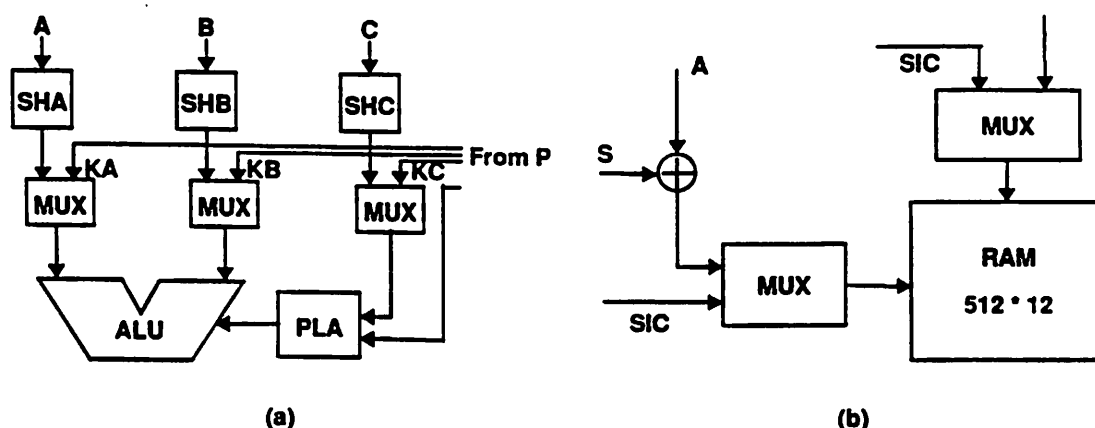


Fig. 6-6. Block diagram of (a) arithmetic and logic element (ALE) and (b) memory element (ME).

look up capability and to implement large number of delays that OBs cannot support. The OBs maintain a variable delay to synchronize the data at the output ports. A detailed diagram of the ALEs and MEs is shown in Fig. 6-6.

The whole chip is fully pipelined at a clock rate of 27 MHz. The ALEs and the OBs can read in data and generate new data at the full clock rate. The MEs can also read or write data at the clock rate. Each of the ALEs, MEs, and OBs has a program memory module associate it such that it knows which operation to perform. The inputs ports and the outputs of the ALEs and MEs are fully connected through a programmable cross bar switch such that the output of any of ALEs and MEs can be routed to any input of themselves, or to the 5 OBs. The overall architecture provides a very high processing and I/O rate— up to 135MIPS, 135 M samples input and 135 M sample outputs. This cross-bar switch architecture also make it very easy to integrate multiple VSP chips parallely on the same task.

Programming the VSPs requires first mapping the video algorithm into a synchronous signal flow graphs (called *soft draw*) with blocks of basic operations supported by the ALEs and MEs. Each block in the flow graph runs at a constant clock period which is a

multiple of 27 MHz clock period. That is, each block runs at 27 MHz, 13.5 MHz, 6.75 MHz, etc. Once the signal flow graph is available, the graphic can be partitioned and allocated to each of the processing components. The signal flow graph generation, partition, and resource allocation are done manually. The programmer must assure the task assigned to one processing component does not exceed the processing capability of that component. When multiple VSP chips are used, the programmer also specifies the connections between the VSPs in a hardware connection graph (called *hard draw*), and assigns tasks to each VSP chip manually. An example soft draw and hard draw graph is shown in Fig. 6-7.

Once the partition and allocation is done, an automatic scheduler is available to schedule the partitioned result into the instruction programs of the processing components. The scheduler works out the correct timing of the input/output data streams among all processing components. The automatic scheduler is a necessary tool to help the programmer design a system within a reasonable time.

6.2.3 Implementation result and discussion

Following the design procedure described in the last section, we are able to implement the VCP using one VSP-8 board with 8 VSP chips on the board. The hard draw in Fig. 6-7(b) shows how the 8 VSP chips are physically connected. The soft draw in Fig. 6-7(a) is part of the signal flow graph design for the VCP. A complete graph of the design of VCP is shown in Appendix A.1 and A.2. In this implementation, only part of the VCP functions are implemented with VSPs — mainly the compositing computation block and part of the compare and control block. The function of the frame sync generation block, the finite state machine, and the binary logic operations of the compare & control block are put outside of the VSP chips for efficient use of the VSP resource. Using the VSPs to implement the functions such as FSM and frame sync generation block will cost many VSP chips and meanwhile waste a lot of resource on the VSP chip. Instead, most of these functions can be implemented efficiently by using field-programmable logic devices (FPLD), such as the Xilinx 6400 that we use for VRAM and VME bus interface control.

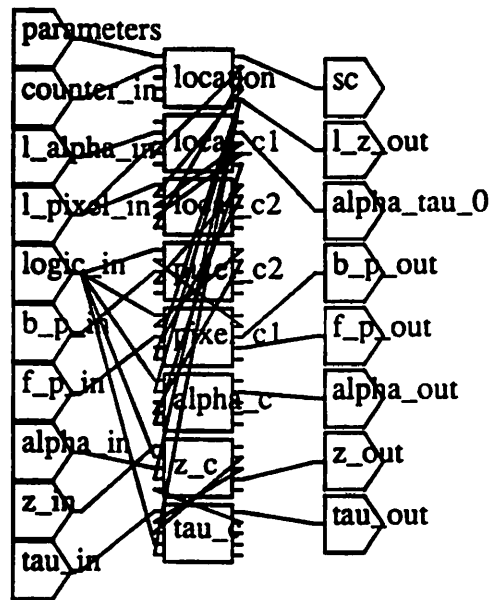
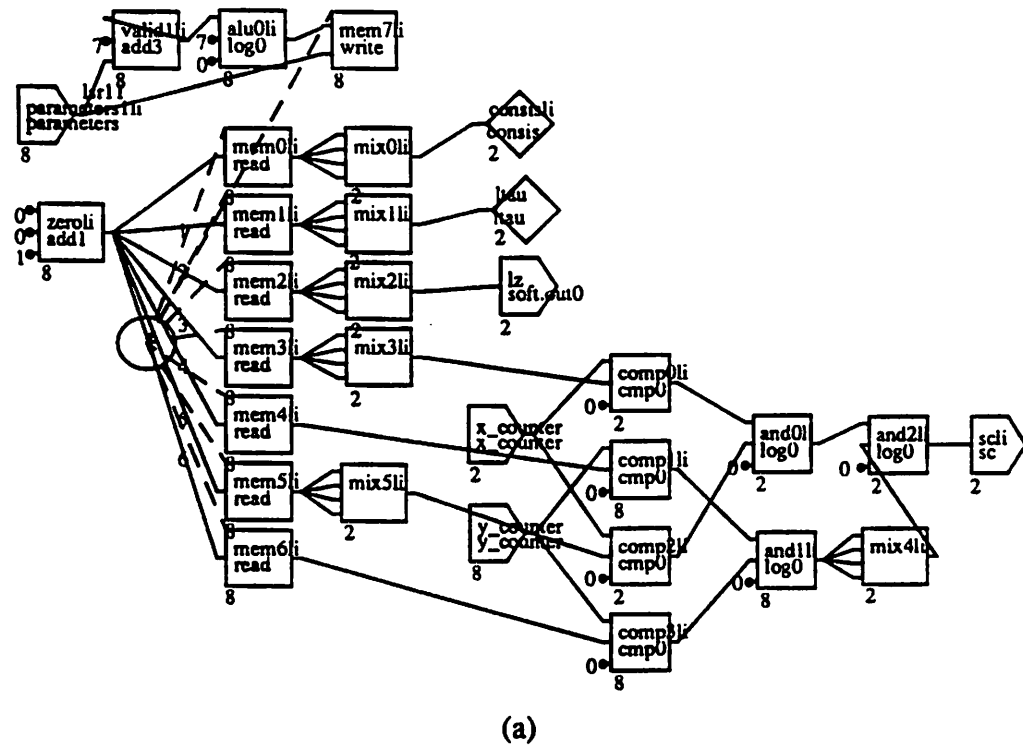


Fig. 6-7. An example of (a) soft draw and (b) hard draw used for the VSP chip programming.

Here, we will discuss the strength and the weakness of the Philip VSP processor based on our implementation experience.

Strengths

- The Philip VSPs are good for realizing computations that are represented by an equation, e.g. filters. The multiple processing elements on the chip and the capability of integrating multiple VSP chips provide a very high processing power to satisfy the computation need of most realtime video signal processing algorithms.
- The crossbar switch provides a very flexible way of connecting the multiple processing elements on the chip. It enables very efficient scheduling, i.e., the resource utilization overhead due to scheduling is very low. It also eases the integration of multiple VSP chips.
- The VSP programming environment is reasonably easy to program. It provides a graphic user interface to input soft draw and hard draws. The automatic scheduling tool is also efficient. This is far ahead of the IIT's VCP chip, which relies totally on manual scheduling.

Weaknesses

- The Philip's VSP is not designed for control purpose. It does not support simple logic operations such as simple 1 bit logic operations. We can use the 12-bit wide ALEs to perform the simple logic operations, which however wastes 11/12 of the resource.
- Conditional branch operations is not supported efficiently. Unlike traditional FIR or IIR filtering algorithm, today's video algorithms uses conditional branch a lot. Examples are algorithms such as motion compensation and conditional replenishment. The current synchronous signal flow graph used by VSP cannot support conditional branch. Also, the VSP does not allow multiple sets of programs stored in the program memory and execute only one branch of the program according to previous execution results. Currently once the signal flow graph is mapped, all branches consume processing resource even though the branch should not be executed.

- Cross-bar switch does not support data dependent switching. Data dependent switching is possible at the cost of using ALEs as the switching element.
- The ALE and the ME does not have equivalent bandwidth capability. The ALEs on the VSP consume and at the same time produce data at 27 MHz clock rate. However, the memory elements can only read or write data at the 27 MHz rate, but not read and write simultaneously. This prohibits the VSP chip to become a real pipeline engine that can pump data in and out at full clock rate. It also causes a lot of programming complexity when we need to read/write at full clock rate. It is preferable to have the MEs to support 27 MHz bi-directional read/write operations, even though technically it may be hard to design a memory module with 54 MHz rate.
- It is difficult to interface with interrupt driven modules. The VSP is a fully synchronous engine which assumes constant clock rate data pumped in and out all the time. This causes some difficulty when it is to be integrated with some asynchronous modules.

All the desired capabilities described above can be done in one way or another, however, with very low resource utilization efficiency. That is, it may take many VSPs to achieve with the current VSP capability. In essence, the Philip VSP processor is very capable for computation portion (e.g., add, multiplication.) of the video compositing algorithm. However, it is less suitable to be used for control portion (e.g., conditional branching, single bit binary logic, data dependent switching, etc.) of the compositing algorithm. Since our VCP compositing algorithm combines both computation and compositing portion, the VSP is only useful for part of the algorithm.

Actually, most of today's video algorithm consists of both computation portion and the control portion, it is a must that a general purpose video signal processors support both. A good example is the IIT's VCP chip, which uses two separate modules on the chip — video processor and video controller. The video processor provides computation capability and the video controller provides the branching and control capabilities. [72]

6.3 Compositing processor implementation with ASIC

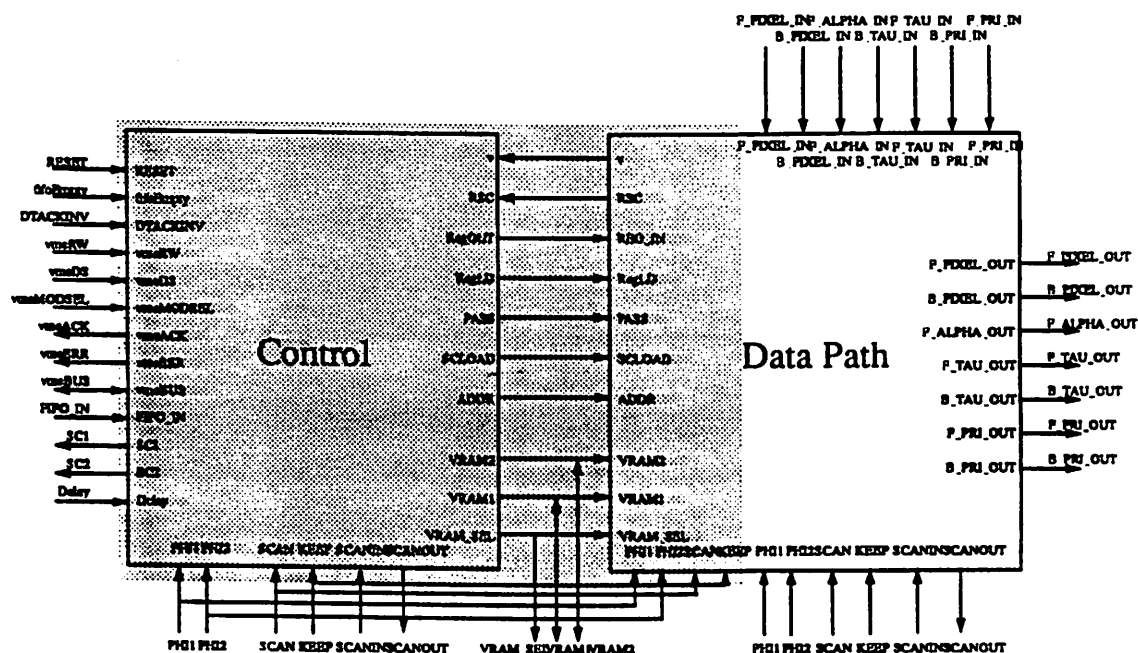
Full custom design IC chip allows most flexibility to optimize and reduce the complexity. However, it is more time consuming to design a chip from scratch. Fortunately, today's VLSI computer aided design (CAD) tools are very powerful. They make the ASIC design easier and more automatic. We have designed an application specific integrated circuit (ASIC) for the VCP of the VideoStation. In this chip, the whole VCP block shown in Fig. 6-4 was put into a very compact IC chip. This is quite a tremendous difference when compared with 8 VSPs and many glue logics in the previous approach to achieve the same functionality.

6.3.1 Design environment

We have laid out and fabricated a video compositing processor chip using the Berkeley's *LagerIV* tools[8] in 1.6 μm CMOS technology. The Berkeley's *LagerIV* is a chip design system that consists of a set of layout generation tools and a set of MOSIS SCMOS cell libraries. The tools set includes a parameterized macrocell generator called *TimLager*, a programmable logic array(PLA) generator called *Plagen*, a standard cell placement and routing tool called *Stdcell*, a bit-slice data path macrocell generator called *dpp*, a general purpose macrocell placement and routing tool called *Flint*, and other simulation tools. This environment suffices a user to design a chip and generate and simulate the layout automatically.

6.3.2 Design methodology

The design of the chip starts with defining the function of the chip, and come out with a functional diagram. A functional diagram of the VCP chip is shown in Fig. 6-8. In the functional diagram, we mainly separate the chip into two blocks — the control block and the computation data path block, as the way we analyzed our implementation of VCP using VSP in the last section. The control block includes all the control related sub-blocks, such as the finite state machine, the video frame sync signal generation, binary logics, etc.



Note: The shading part runs at the clock rate PHI1, and PHI2.

The non-shading part runs at the clock rate of PHI1F and PHI2F.

Fig. 6-8. The VCP chip functional diagram.

The computation data path block is a data path using the foreground and background video input streams to generate the composited output video streams. The detailed design of the two blocks are shown in Appendix B.1.

The reason of separating the VCP chip into two blocks is because of the different characteristics and requirements between the data path block and the control block. The data path block performs operations over the incoming video samples running synchronously at a very high speed of 27 MHz clock rate (in CCIR 601 format). The main operations are switching, addition, multiplications, etc. On the other hand, the control block is much more complicated in functionality. The speed of the control logic, however, may be much lower than the data path block. It basically controls and coordinates operations of the whole pipeline stage. It receives related informations from all the blocks in that stage, including external blocks such as VME interface module, VRAM interface, and the FIFO,

and generates signals to control and instruct all the blocks to perform operations properly. It interfaces with other modules in both synchronous and asynchronous way.

With these considerations, we are designing the chip using two different clock rate. One running at the data path clock rate of 27 MHz. The other runs at a much lower clock rate of around 9 MHz. Fig. 6-8 shows the portion of the VCP chip that runs at the lower clock rate in the black shading. The portion without the shading are running at the 27 MHz clock rate.

Once the functional diagram is available, the second step of the design is to map the functional diagram into available cells in the Lager cell library. An important consideration here is to choose an appropriate cell and an appropriate tool to generate the macrocells to satisfy the functional requirement and also minimize the macrocell size. Here, we use the data path generator (dpp) and the macrocell tiling program (TimLager) to generate most of the macrocells in the data path block. For the control block, we are using various tools. We use the PLA generator (Plagen) to generate the finite state machine and one of the combinational logic macrocells (The functional description of the finite state machine and the combinational logic cells are shown in Appendix B.2 and B.3.) We also use the standard cell library and TimLager to generate the rest of the macrocells. Once the cells are generated, they are connected together by using Flint as the placement and routing tool.

In addition to using the Lager IV tools directly, we also need to redesign some of the macrocells in the library to satisfy the speed requirement of the data path block. This is done by putting some pipeline latches into the macrocells such that the delay between latches is lower than the original macrocells, and can therefore run at a higher clock rate.

Making macrocells pipelined is easy for some macrocells and difficult for others. For example, it is easy to make the macrocells generated by the data path generator (dpp) pipelined because the dpp allows the user to specify how to insert the pipeline latch to

meet the requirement. Other macrocells, such as those generated by TimLager, may not allow users to put in the pipeline latches at will. For these macrocells, we need to rewrite the macrocell tiling program to make them pipelined and run at higher clock rate. An example is the array multiplier macrocell generated by the TimLager. In the design of the VCP, we created a new array multiplier macrocell that can be pipelined at every three bit-slice to make it run at the 27 MHz clock rate without any problem.

There are also macrocells that is extremely difficult to modify to become pipelined. The finite state machine macrocells generated by the PLA generator is an example. For these macrocells, we have no easy way to enhance the speed. Fortunately most of these macrocells are used for control purpose, and can actually be run at a lower rate than the data path clock rate through a careful design. After recognizing this fact, we choose to use 2 different clock rate for different macrocells such that most of the macrocells are running at a lower clock rate. Only those macrocells in the data path block run at the 27 MHz rate. Through this, we can avoid the requirement to modify many macrocells to be pipelined.

6.3.3 Simulation, fabrication and testing

The simulation of the chip is done by using the simulation tools provided in LagerIV. The tools we use are *Thor*, *irsim* and *spice*. *Thor* is used for simulation from gate level to behavior level. The *irsim* is used to perform switch level simulations. It can simulate the circuit extracted from the actual layout, and perform timing simulation to guarantee the correct function of the layout. The *spice* is used to perform more accurate timing simulation. The simulation time limits the use of these three different simulation tools. The higher level simulation tools, such as the *Thor*, can be run at a reasonable speed to simulate the whole chip without any problem. *Spice* simulator, however, takes a very long time for large circuits. It can only be used to simulate the timing of some critical path. *Irsim*, as a switching level simulator between *Thor* and *spice*, runs in reasonable time to simulate large macrocells. It is possible to use *irsim* to simulate the whole chip. However, it takes quite a long time to finish the job. In our design process, we did use *irsim* to simulate the

behavior of the whole chip to guarantee the correct functionality of the chip before the chip is sent out for fabrication.

For easy testing, we also incorporate some testing circuit into the chip design. We use scan register to implement the output of most critical macrocells such that the result of the macrocells can be easily shifted out serially through the scan path. Since we run the chip at two clock rates, we have two separate scanning paths for the components running at different rates. The two scanning paths are shown in Appendix B.4.

The final layout of the VCP chip has a die size of 12.1 x 12.1 mm. It contains 20698 n-channel transistors and 18529 p-channel transistors. The power consumption is about 1.3 watt according to rough calculation by using the total capacitances. The designed chip has a very large number of input/output signals, and uses a 208 pins PGA package. (See Appendix B.5 for pin assignment.) This large pin number makes the total die size so large. Without counting the I/O pads, the actual die size of the active circuit is actually only a little bit more than half of the current size.

Once we finished the design, layout and simulation, the chip was sent to the Asahi Chemical Co. in Japan for fabrication and testing. It was fabricated and passed the testing at 27 MHz clock rate using the test vectors that we provide. After the chip is fabricated, we also designed a board for the complete pipeline stage using the fabricated chip. More detail about this board can be found in [25].

6.4 Conclusion

We have implemented the VideoStation video compositing processor to demonstrate the feasibility of real-time video compositing for structured video. We use two different approaches in this chapter — the programmable video signal approach and the ASIC design approach. Basically the programmable video signal approach provides a faster solution for prototyping and an easier way of debugging and redesign. It is most suitable for lab experimental prototype implementation. In this approach, the designers are bound

with the limited availability of today's programmable processors and the built-in architecture of the video processor. When the capability or the architecture of the processor does not fully fit the requirement of the specific application, some glue logic may be needed and the implementation cost and complexity may be increased.

The ASIC design approach provides a direct way of implementation of the prototype. The designer has more flexibility to design the whole circuit from higher level architecture to the low level layout. The final design can be compact and simple and cheaper when produced massively. However, the ASIC approach is much more difficult and time consuming to design from scratch. It needs carefully design from functional design, layout, simulation to testing. Today's VLSI CAD tools provide a way to ease the whole design process a lot. Some of them even provide automatic layout generation from very high level behavior description. Even with the various VLSI CAD tools available today, this ASIC design approach is still much more complex than using the programmable signal processor. In the future, the design complexity gap between the programmable signal processor and the VLSI ASIC can hopefully be reduced when more advanced VLSI CAD tools become available.

Our implementation results show that programming with the Philip's VSP chip is not quite an efficient way for implementing the VideoStation real-time compositing processor. Using Philip's VSP chip, we need 8 VSP chips (on one VSP-8 board) to implement the data path computation portion of the VCP block. To complete the whole VCP block, we need to use more control and glue logic to be integrated with the VSP-8 board. The result is that we not only need to program the VSP processor, but also need to custom design the external control and glue logic hardware. Under this situation, the easy prototyping advantage of the programmable signal processor is reduced dramatically. Using the ASIC design approach, on the other hand, we are able to design the whole VCP block in one single chip of 12.1 x 12.1 mm die size. This shows the dramatically advantage on the final design

complexity of the ASIC design approach over the programmable signal processor approach.

In this chapter, we also discuss the general video signal processors available today. We discussed the reason why the Philip's chip is not so much suitable for the real-time compositing implementation. First, the Philip's VSP chip architecture is basically designed for implementing the data intensive computations. Its capability of supporting control functions and conditional branching is quite weak. While in most of today's video applications, both computation and control functions are equally important. Our structured video compositing algorithm shows this fact quite clearly. This makes the implementation using the Philip's VSP chip not quite efficient. Secondly, even though the Philip's VSP is claimed to be designed for realtime video signal processing purpose, its design is basically for any high speed signal processing. This "high speed signal processing" is only one of the characteristics of any video signal processing algorithms. There are other generic characteristics of video processing algorithms that can be used to optimize the VSP design. For example, most video signal processing requires frame sync restoration, frame delay implementation, block based data structure processing, etc. All these appear a lot in many different video algorithms, and should be supported efficiently. A programmable signal processor designed for video purpose should take these into consideration, and be optimized accordingly.

CHAPTER 7

CONCLUSION

7.1 Summary of research result

Advanced video services is becoming possible with all the fundamental video supporting technologies getting mature. One missing technique to support advanced video services is the real-time video compositing technique to allow real-time access and integration of video elements over the network. This thesis basically discuss various aspects of real-time video compositing — from its high level video information structure to the low level hardware implementation.

In this thesis, we first review the nature and characteristics of the advanced video services. Then we propose a *structured video model* to provide a framework to support efficient real-time video compositing for advanced video services. It essentially represents the compositing video scene in a hierarchical tree structure while at the same time keeps all the video elements logically separate over the network until the very last stage of video compositing at the users workstation. By doing this, the whole data structure is maintained

in a very clean, structural way. All the video elements can also be kept in a very simple form that can be most efficient for data compression, video material sharing and reuse. This makes the network management and the network resource utilization more efficient. The structured video model also allows instantly interactive control for real-time compositing of video information. The model also provides a way for efficient implementation of the compositing system. This is achieved through introducing the basic *compositing functions* for modular implementation and the restructuring capability of the composite object.

Based on this structured video model, we study the technologies that actually support the realization of the structured video model in chapter 4. In spatial compositing, we start with Porter and Duff's anti-aliasing algorithm, extend it, and derive the compositing algorithm of all the compositing functions that we defined in the structure video model. We also study the implementation flexibility of the structured video model in allocating video compositing functions to a distributed network environment. The structured video can provide a means for studying various possibilities of implementation, and provides as a tool for its performance analysis and optimization. To do this, we explored the generic structure of compositing functions and their useful properties such as *associative, commutative, and distributive*, which enable easy manipulations of the compositing functions and restructuring of the structured video representation of a video service and its associated implementation.

In temporal compositing, we discuss two important synchronization issues — the clock rate matching and the multiple object synchronization. In rate matching, we propose to use a global timing clock for all the components in a composite video object. It essentially uses separate channels for transmitting timing signals for global clock. With the complexity of tree structure, it is difficult to use traditional methods such as slip buffering or the buffer monitoring method to match the clock rates among all the components. Since the goal of structured video is to support real-time full motion video, it is best to use a global clock to allow each component to run freely without other intervention. To main-

tain the synchronization among multiple video objects, we use a simple mechanism to set the origin of the time basis of all the components, and measure the delay of the links in the composite object structure.

In chapter 5, we described the design detail of a video compositing platform - VideoStation that supports real time compositing display for structured video. We first reviewed the implementation bottleneck with today's display technology. There are mainly two bottlenecks — the memory bandwidth and the processing capability bottlenecks. To get around of these bottlenecks, we proposed a pipeline architecture for the VideoStation. It uses a multiple parallel compositing modules on a pipeline data path to provide enough processing capability and the memory bandwidth. The basic idea of this pipeline architecture is to avoid the traditional single frame buffer approach so that there is no fix mapping between the memory address and the display physical location. The advantage of this is to provide more flexible mapping between the memory address and the display, thus making the memory usage more efficient. Some of the compositing operations, such as bitblt, can be performed easily and efficiently with this pipeline architecture. The pipeline architecture also allows a much easier method of expansion to accommodate more video objects.

To avoid the presorting requirement in the compositing, we proposed a 3-object compositing mechanism which does not perform the actual compositing until the last stage. The function of each pipeline stage is to select the two highest priority objects and pass down to the next stage. With this 3-object compositing, any dynamic change of priority values can be done without difficulties. By choosing proper parameters passing between the stages, we also make the operations in each pipeline stage as simple as possible. Compared with direct implementation which requires division operations in every stage, the optimized approach needs the division only in the last stage. All the rest of the stages need only addition/multiplication, and some simple logic operations.

In the last chapter, we present our prototype design of VideoStation in two different approaches — the programmable video signal processing approach and the VLSI ASIC

design approach. This basically compares the two most common implementation methods used today for real-time video processing. Our results shows that the implementation using the Philip's programmable video signal processor is not quite efficient for our real-time compositing algorithm of VideoStation. Using Philip's VSP chip, we need 8 VSP chips (on one VSP-8 board) to implement the data path computation portion of the VCP block. To complete the whole VCP block, we need to use more control and glue logic to be integrated with the VSP-8 board. Using the ASIC design approach, on the other hand, we are able to design the whole VCP block in one single chip of 12.1 x 12.1 mm die size. This shows the dramatic advantage on the final design complexity of the ASIC design approach over the programmable signal processor approach.

There are several reasons why the Philip's chip is not quite suitable for the real-time compositing implementation. First of all, the Philip's VSP chip architecture is basically designed for implementing the data intensive computations. Its capability of supporting control functions and conditional branching is quite weak. While in most of today's video applications, both computation and control functions are equally important. Our structured video compositing algorithm shows this fact quite clearly. This makes the implementation using the Philip's VSP chip not quite efficient. Secondly, even though the Philip's VSP is claimed to be designed for real-time video signal processing purpose, it is actually designed basically for any high speed signal processing. This "high speed signal processing" is actually only one characteristic of any video signal processing algorithms. There are other generic characteristics of video processing algorithms that can be used to optimize the VSP design. For example, most video signal processing requires frame sync restoration, frame delay implementation, block based data structure processing, etc. All these appear a lot in many different video algorithms, and should be supported efficiently. A programmable signal processor designed for video purpose should take these into consideration, and be optimized accordingly.

7.2 Future Direction

We have followed a very straightforward thinking of real-time video compositing issues of advanced video services — starting from the high level information structure to the low level hardware implementation. Along this line, the extent that we covered is still quite limited. There are a lot more to pursue in the future. Here, we would like to point out some major ones.

In the discussion of the structured video model, we pointed out the possibility of using the model to optimize the network resource allocation in a distributive network environment. We studied the properties of the compositing function in the structured video model to enable flexible resource allocation. However, we did not study how to achieve the optimization. The resource allocation algorithms using the restructuring capability of the structured video model need to be further explored.

Along with the resource allocation optimization issue is the real-time scheduling issue in video compositing. The temporal representation proposed in chapter 3 can be used as a foundation for scheduling. However, the scheduling can actually be more involved because structured video includes not only limited life span video objects, but also undetermined life span objects and interactive operations. It is sure that some kind of dynamic scheduling scheme is needed. When the optimization considers both the resource allocation in the spatial domain and the scheduling in the temporal domain, it becomes a very challenging problem, and deserves further detailed study in the future.

Once the scheduling and the resource allocation are done, another issue that needs to be investigated is the distributive call model and its associative signalling and negotiation protocol for this allocated tree structure. The call model not only need to handle the initial establishment and tear down of multi-point multi-media call connection, but also needs to handle dynamic connection establishment during the run time. There are currently many activities in the ATM Forum on the signalling protocols for switched virtual connections (SVC) over ATM network. The current standard adopted is one called Q.2931. This sig-

nalling protocol, however, supports only point to point connections. It does not support any multi-party multi-media connection in a call, not to mention the complicated tree structure used in the structured video model. The call model and signalling protocol of the structured video can be built based on the current Q.2931 standard and extend it. This will basically satisfy the need of the structured video protocol issue.

Regarding VideoStation, we have designed and optimized the architecture with respect to a subset of compositing functions in the structured video model. Those compositing functions includes `over()`, `transparent`, `translation()`, `delay()`, etc. There are many other compositing functions described in chapter 3 that are of interest for general video compositing applications. A more generic compositing processor implementing all these compositing functions need to be further studied.

In the pipeline architecture proposed, each pipeline stage can only support one isochronous video objects disregarding how large the video object is. This is a waste of the compositing resource considering the processing capability of each pipeline stage is far beyond. A more flexible design to enable multiple isochronous video object compositing in one pipeline stage is definitely another issue that need to be further pursued.

REFERENCES

- [1] Francis Kretz and Francois Colaitis, "Standardizing Hypermedia Information Objects", *IEEE Communication Magazine*, May, 1992.
- [2] Francois Oguet, Christiane Schwartz, and Francis Kretz, "RAVI, A proposed Standard for the Interchange of Audio/Visual Interactive Applications", *IEEE Journal on Selected Areas in Communications*, Vol. 8, NO. 3, April 1990.
- [3] Brian D. Markey, "HyTime and MHEG," *IEEE COMPCON*, pp. 25-40, San Francisco, CA, Feb. 1992.
- [4] DTAM, Document Transfer, Access and Manipulation, protocol described in the CCITT recommendations on Open Document Architecture (ODA), T431-Introduction and General Principles, T432-Service Definitions, T433-Protocol Specifications, 1988.
- [5] Wataru Kameyama, Tsuyoshi Hanamura and Hideyoshi Tominaga, "A Proposal of Multimedia Document Architecture and Video Document Architecture," ICC 91, pp. 511-515.
- [6] Naoki Kobayashi and Toru Nakagawa, "Multimedia Document Structure for Dialog Communication Service," ICC 91, pp. 526-531.
- [7] Patrick McLean, "What's in a Picture? A Structured Approach to Video Coding," *Intl. Symp. on Signals, Systems, and Electronics*, Sept. 1992.
- [8] R.D. Gaglianella, T.B. London, B.S. Robinson, and D. Swicker, "The Liaison Network Multimedia Workstations," *Proc. Global Commun. Conf.*, Phoenix, Dec. 1991.
- [9] Martin De Prycker, *Asynchronous Transfer mode - Solution for Broadband ISDN*, Ellis Horwood, 1993.
- [10] CCITT Recommendation I.352, *Network Performance Objectives for Connection Processing Delays in an ISDN*, Vol. III, Fascicle III.8, Blue Book, 1988.
- [11] CCITT IVS Baseline Document, SG XVIII/8, Geneva, June 1992.

- [12] Raif O. Onvural, *Asynchronous Transfer Mode Networks: Performance Issues*, Artech House Inc., 1994.
- [13] D. A. Pattern, G. A. Gibson and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *ACM SIGMOD 88*, Chicago, 1988.
- [14] G. A. Gibson, "Redundant Disk Arrays: Reliable, Parallel Secondary Storage", Ph. D. Dissertation, University of California, Berkeley, 1991.
- [15] A. L. Narasimha Reddy and J. C. Wyllie, "I/O Issues in a Multimedia System", *IEEE Computer Magazine*, March, 1994, pp. 69 - 74.
- [16] K. Lantz and W. Nowicki, "Structured Graphics for Distributed Systems," *ACM Transactions on Graphics*, vol.3, #1, January, 1984, pp. 23-51.
- [17] CCITT Recommendation H.261, "Video Codec for Audiovisual Services at px64 kbits/s"
- [18] Standard Draft, JPEG-9-R7, Feb. 1991
- [19] Standard Draft, MPEG Video Committee Draft, MPEG 90/ 176 Rev. 2, Dec. 1990.
- [20] Pedro A. Szekely and Brad A. Myers, "A user interface toolkit based on graphical objects and constraints," *SIGPLAN NOTICES*, September 1988.
- [21] Ralph D. Hill, "A 2-D graphics system for multi-user interactive graphics based on objects and constraints," in *Advances in Object-Oriented Graphics I*, E. H. Blake, P. Wisskirchen (Eds), pp. 67-91.
- [22] H.G. Musmann, M. Hotter, and J. Ostermann, "Object-Oriented Analysis-Synthesis Coding of Moving Images," *Signal Processing: Image Communication*, pp. 117-138, 1989.
- [23] H.-D. Lin and D.G. Messerschmitt, "Video Compositing Methods and Their Semantics," *IEEE ICASSP '91*, Ontario, Canada, 1991.
- [24] W.-L. Chen, P. Haskell, L. Yun, and D. G. Messerschmitt, "Videostation: a hardware implementation of structured video," in preparation.
- [25] Louie Yun, "The VideoStation Board," Master Report, UC Berkeley, 1993.
- [26] S.-F. Chang and D. G. Messerschmitt, "A New Approach to Decode and Composite Motion Compensated DCT-Based Video," to appear on *ICASSP '93*, Minneapolis, Minnesota, April, 1993.

- [27] S.-F. Chang, W.-L. Chen and D. G. Messerschmitt, "Video Compositing in the DCT domain," *IEEE Workshop on Visual Signal Processing and Communications*, Raleigh, NC, Sep. 1992.
- [28] S.-F. Chang, "Compositing and manipulation of video signals for multimedia network video services", Ph.D. Dissertation, UC Berkeley, 1993.
- [29] H. Kamata, T. Katsuyama, T. Sizuki, Y. Minakuchi, K. Yano, "Communication Workstations for B-ISDN: Monster (Multimedia Oriented Super Terminal)" *IEEE Globecom*, Nov. 1989, pp. 959-964.
- [30] Peter Wisskirchen and Klaus Kansy, "The new graphics standard - Object-Oriented!", in *Advances in Object-Oriented Graphics I*, E.H. Blake, P. Wisskirchen (Eds), pp 199-215.
- [31] D. Bursky, "Programmable-Architecture Parallel Processor Handles Real-Time Video," *Electronic Design*, November 22, 1990, p. 34.
- [32] W.-L. Chen, S.-F. Chang, P. Haskell, and D. G. Messerschmitt, "Structured Video Model for Interactive Multimedia Video Services," U.C. Berkeley, Dept. of EECS, 1992.
- [33] S.-F. Chang, P. Haskell, and D. Messerschmitt, "Allocation of Video Compositing Hardware in Multimedia Networks," U. C. Berkeley Dept. of EECS, 1992.
- [34] T. Duff, "Compositing 3-D Rendered Images," *Siggraph*, November 1985, vol. 19, pp. 41-44.
- [35] T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics*, July 1984, Vol. 18, No. 3, pp.253-259.
- [36] H. Fuchs, et. al., "Coarse-Grain and Fine-Grain Parallelism in the Next Generation Pixel-Planes Graphics System," in *Parallel Processing for Computer Vision and Display*, P. M. Dew, R. A. Earnshaw, and T. R. Heywood, editors, New York, Addison-Wesley, 1989, pp. 241-253.
- [37] D. Hearn and P. Baker, *Computer Graphics*, New York, Prentice Hall, 1986.
- [38] H. Kamata, T. Katsuyama, T. Sizuki, Y. Minakuchi, K. Yano, "Communication Workstations for B-ISDN: Monster (Multimedia Oriented Super Terminal)," *IEEE Globecom*, Nov. 1989, pp. 959-964.
- [39] *LagerIV Distribution 1.0: Silicon Assembly System Manual*, Report of the Electronics Research Laboratory, University of California, Berkeley, June 1988.

- [40] "VideoWindows" and "VideoWindows HR," New Media Graphics Corp., 780 Boston Road, Billerica, MA, 01821-5925.
- [41] T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics*, 1984, vol. 18, #3, pp. 253-259.
- [42] Murat Kunt, Michel Benard, and Riccardo Leonardi, "Recent Results in High-Compression Image Coding," *IEEE Transactions on Circuits and Systems*, Vol. CAS-34, No. 11, November, 1987.
- [43] "Digital Television", edited by C. P. Sandbank, John Wiley & Sons, 1990.
- [44] Arun N. Netravali, and Barry G. Haskell, "Digital Pictures: Representation and Compression", Plenum Press, 1988.
- [45] "TMS44C251 256Kx4-bit Multiport Video RAM," *Texas Instruments MOS Memory Databook*, Commercial and Military Specifications. publication #SMYD008, Houston, TX.
- [46] "TARGA+," TrueVision, 7340 Shadeland Station, Indianapolis, IN 46256.
- [47] P. Vasilopoulos, *Interactive Video Services Simulated on the VideoStation System*, M. S. Report, University of California, Berkeley Department of Electrical Engineering and Computer Sciences, October 1990.
- [48] "DVA-4000/MCA," VideoLogic, Inc., 245 First Street, Cambridge, MA 02142.
- [49] "Broadband ISDN and Asynchronous Transfer Mode (ATM)", Steven E. Minzer, *IEEE Communication Magazine*, September 1989, pp. 17 - 24.
- [50] "Progress in Standardization of SONET", Rodney J. Boehm, *IEEE LCS Magazine*, May 1990.
- [51] Special Issue of Congestion Control in ATM Network, *IEEE Network*, September 1992, Vol. 6, No. 5.
- [52] "Congestion Control for Multimedia Services", Ljiljana Trajkovic, and S. Jamaloddin Golestani, *IEEE Network*, September 1992, Vol. 6, No. 5, pp. 20 - 26.
- [53] Thomas D. C. Little and Arif Ghafoor, "Network Considerations for Distributed Multimedia Object Composition and Communication". *IEEE Network*, November 1990, pp. 32 - 49.

- [54] Thomas D. C. Little and Arif Ghafoor, "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks", *IEEE Computer Magazine*, October 1991, pp. 42-50.
- [55] Anil K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Inc., 1989.
- [56] Hanan Samet and Robert E. Webber, "Data Structures: Hierarchical Data Structures and Algorithms for Computer Graphics, Part I- Fundamentals," *IEEE Computer Graphics & Applications*, pp 48 - 68, May, 1988.
- [57] Whitton, M.C., "Memory Design for Raster Graphics Displays," *IEEE Computer Graphics & Applications*, Vol. 4, No. 3, March 1984, pp. 48-65.
- [58] Ralf Steinmetz, "Synchronization Properties in Multimedia Systems," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April, 1990, pp. 401 - 412
- [59] Thomas D. C. Little, and Arif Ghafoor, "Synchronization and Storage Models for Multimedia Object," *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3, April, 1990, pp. 413 - 427.
- [60] David P. Anderson and George Homsy, "A Continuous Media I/O Server and Its Synchronization Mechanism," *IEEE Computer Magazine*, October 1991, pp. 51 - 57.
- [61] Thomas D. C. Little, and Arif Ghafoor, "Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks," *IEEE Computer Magazine*, October 1991, pp. 42 - 50.
- [62] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832 - 843.
- [63] L. Lamport and P.M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM*, 32(1), pp. 54-78.
- [64] T. K. Arikanth and S. Toueg, "Optimal clock synchronization," *Journal of the ACM*, 34(3), pp. 626-645.
- [65] F. Cristian, "Probabilistic clock synchronization," *Distributed computing* 3, pp. 146-158.
- [66] Sharp memory databook 1991/1992.

- [67] "SunVideo and Digital Video", technical white paper by Sun Microsystems Corp.
- [68] S. Gupta, R. F. Sproull, and I. E. utherland, "A VLSI architecture for updating raster-scan displays," *Computer Graphics*, 15(3), pp. 71-78.
- [69] S. Demetrescu, "High speed image rasterization using scan line access memories," *Chapel Hill Conference on Very Large Scale Integration*, ed. 1985, pp. 221-243.
- [70] J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heinecke, and et. al., "Pixel planes: Building a VLSI based graphic system," *Chapel Hill Conference on Very Large Scale Integration*, ed. 1985, pp. 35-60.
- [71] N. England, "A graphic system architecture for interactive application-specific display functions," *IEEE Computer Graphics and Applications*, Vol. 6, No.1, January 1986, pp. 60-70.
- [72] Doug Bailey, Matthew Cressa, Jan Fandrianto, Doug Neubauer, Hedley Rainnie, and Chi-Shi Wang, "Programmable vision processor/controller for flexible implementation of current and future image compression standards," *IEEE Micro*, October 1992, pp. 33-39.
- [73] Masakazu Yamashina, Tadayoshi Enomoto, Takemitsu Kunio, Ichiro Tamitani, HidenobuHarasaki, Yukio Endo, Takao Nishitani, Masao Sato, and Koichi Kikuchi, "A microprogrammable real-time video signal processor (VSP) for motion compensation," *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 4, August 1988, pp. 907-915.
- [74] Kevin Harney, Mike Keith, Gary Lavelle, Lawrence D. Ryan, and Daniel J. Stark, "The i750 video signal processor: a total multimedia solution," *Communications of the ACM*, Vol. 34, No. 4, April 1991, pp. 64-78.
- [75] Dave Bursky, "Programmable-architecture parallal processor handles real-time video," *Electronic Design*, November 22, 1990.
- [76] "VSP support tools user guide," Silicon & Software System, November 30, 1992.

Appendix A: Video compositing processor implementation with Philip VSP

A.1 VSP-8 hardware connection graph(Hard Draw)

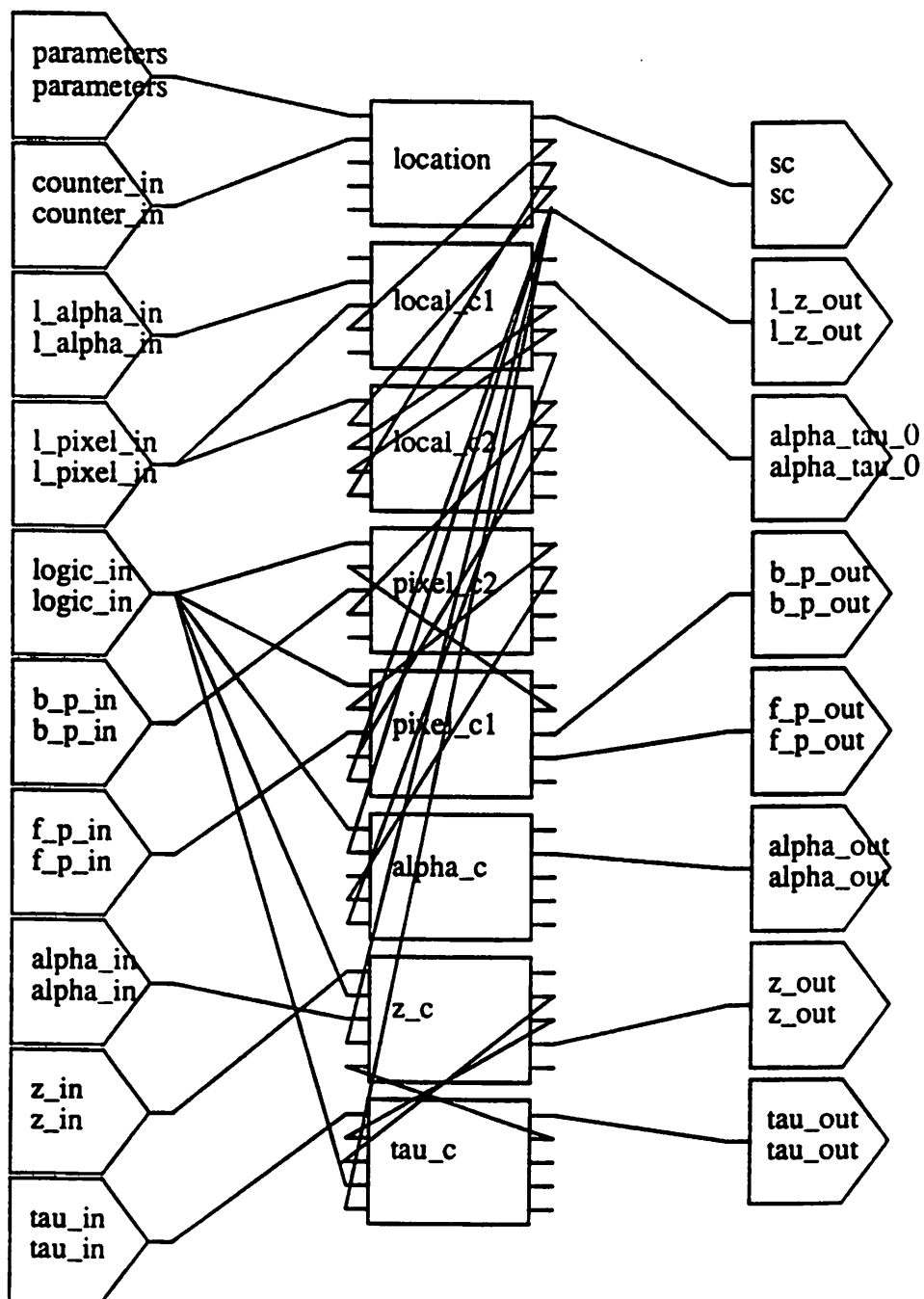


Fig. A-1. Hard draw: This graph specifies the hardware connection between the 8 VSPs and input/output ports on the VSP-8 board.

A.2 Video compositing algorithm signal flow graph (Soft Draw)

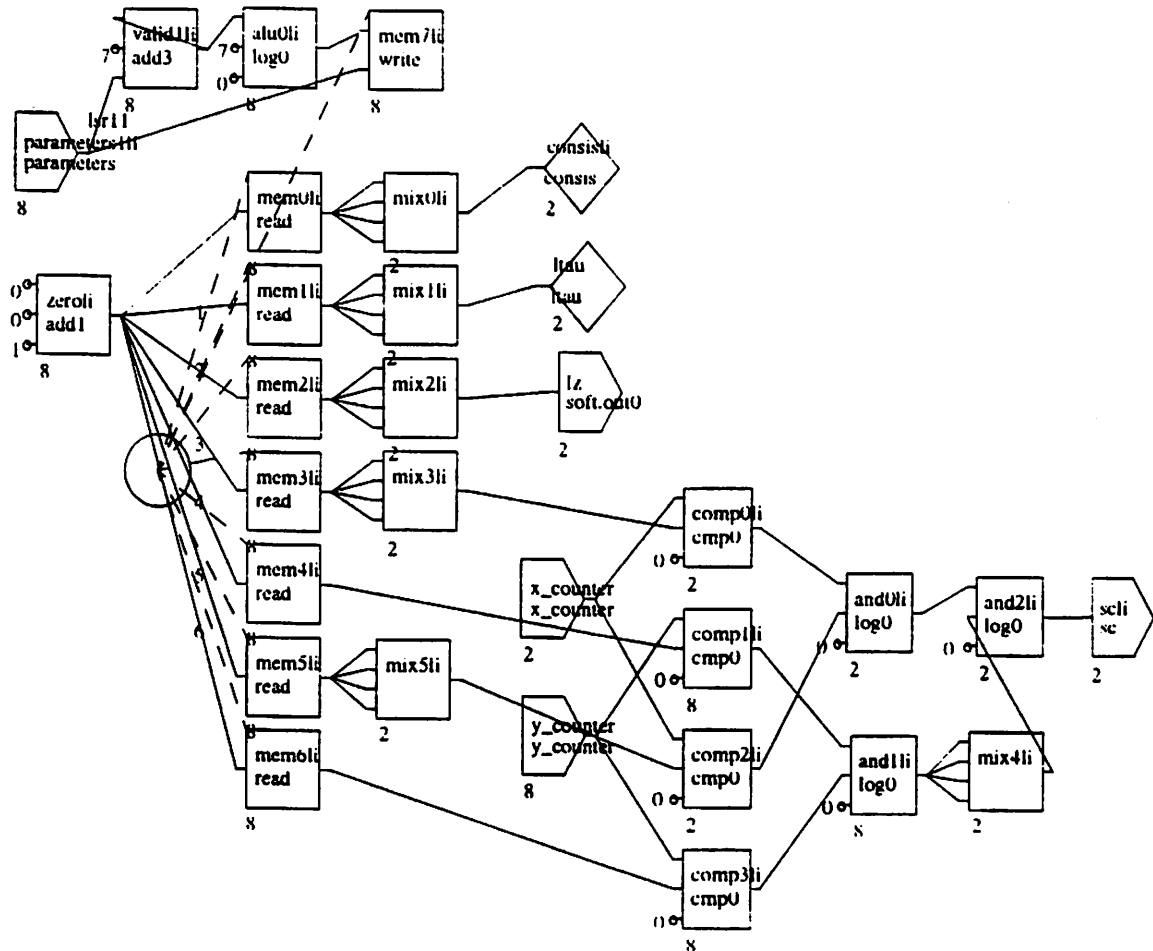


Fig. A-2. Soft Draw: part(1). This signal graph read in the compositing parameters and store in the memory, then use the parameters to compare with current scanning location and generate a control signal to read local pixel values from VRAM.

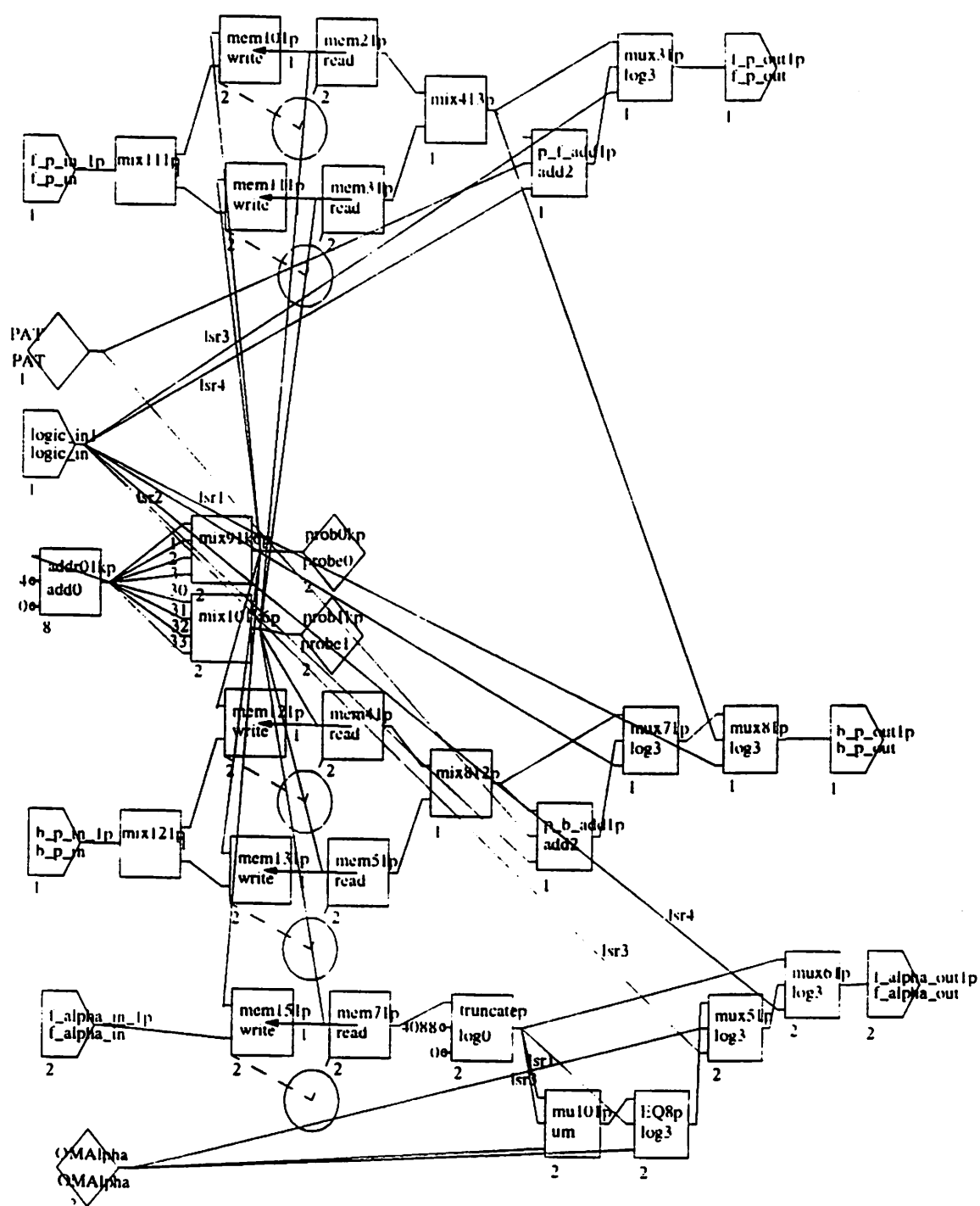


Fig. A-4. Soft Draw: part(3). This signal flow graph perform compositing calculation of foreground pixel, background pixel, and foreground α vlue.

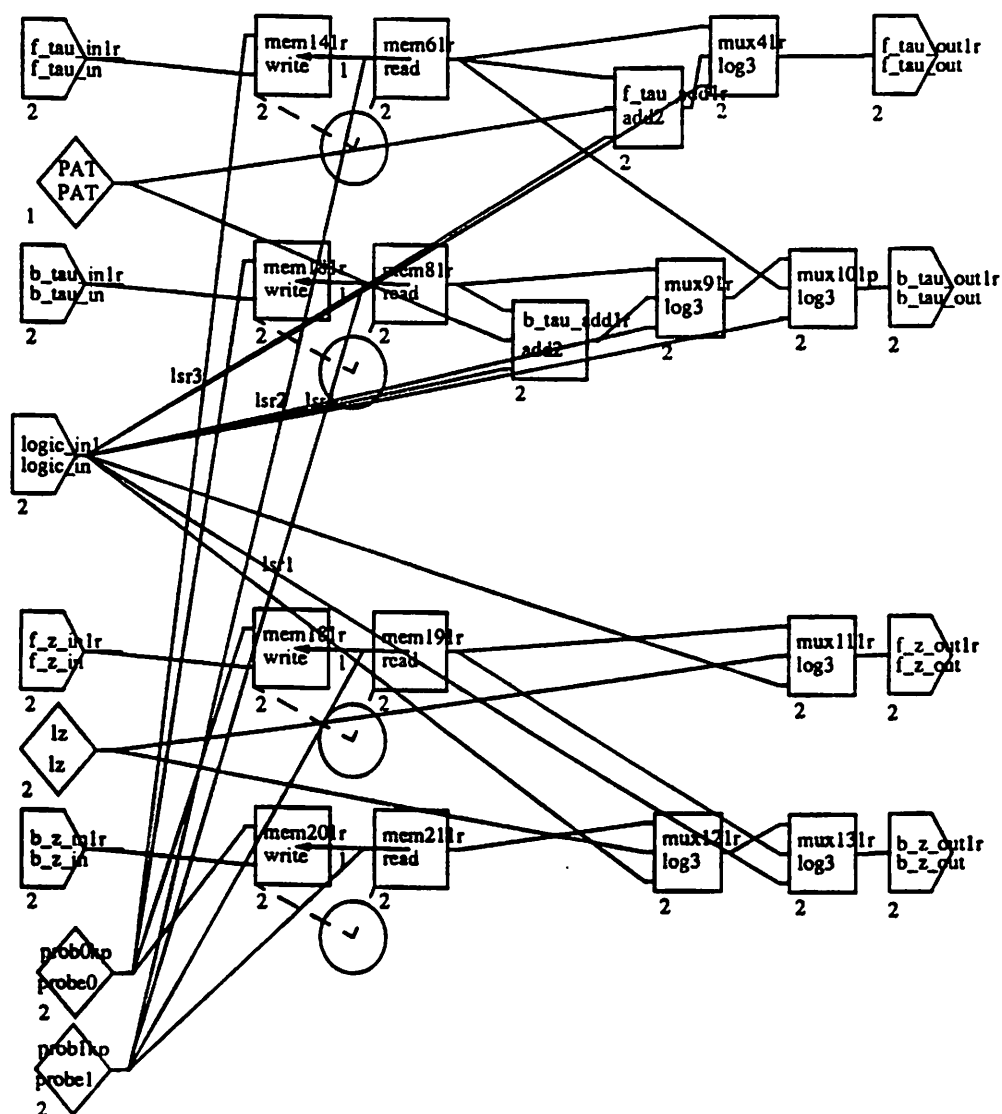
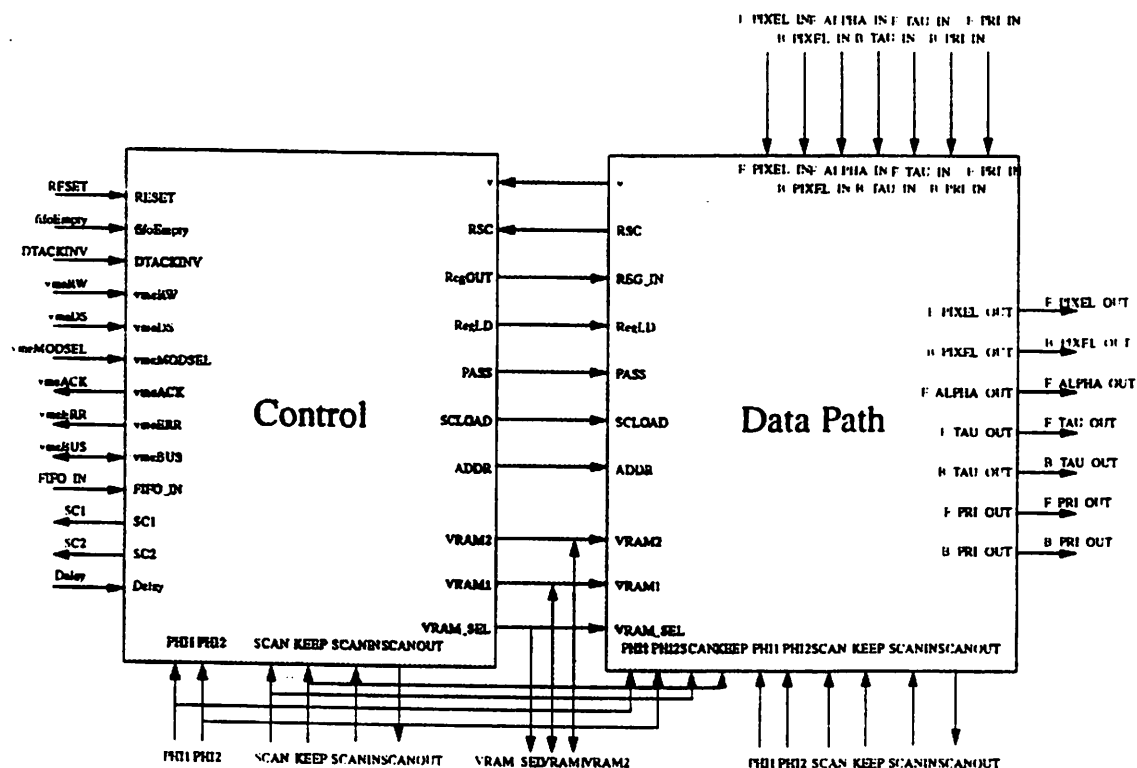


Fig. A-5. Soft Draw: part(4). This signal flow graph perform compositing calculation of foreground and background τ , z values.

Appendix B: Video compositing processor implementation with ASIC approach

B.1 Chip functional diagram



Note: The shading part runs at the clock rate PHI1 and PHI2.

The non-shading part runs at the clock rate of PHI1F and PHI2F.



Fig. B-1. VCP chip block diagram

B.2 VCP finite state machine block implementation

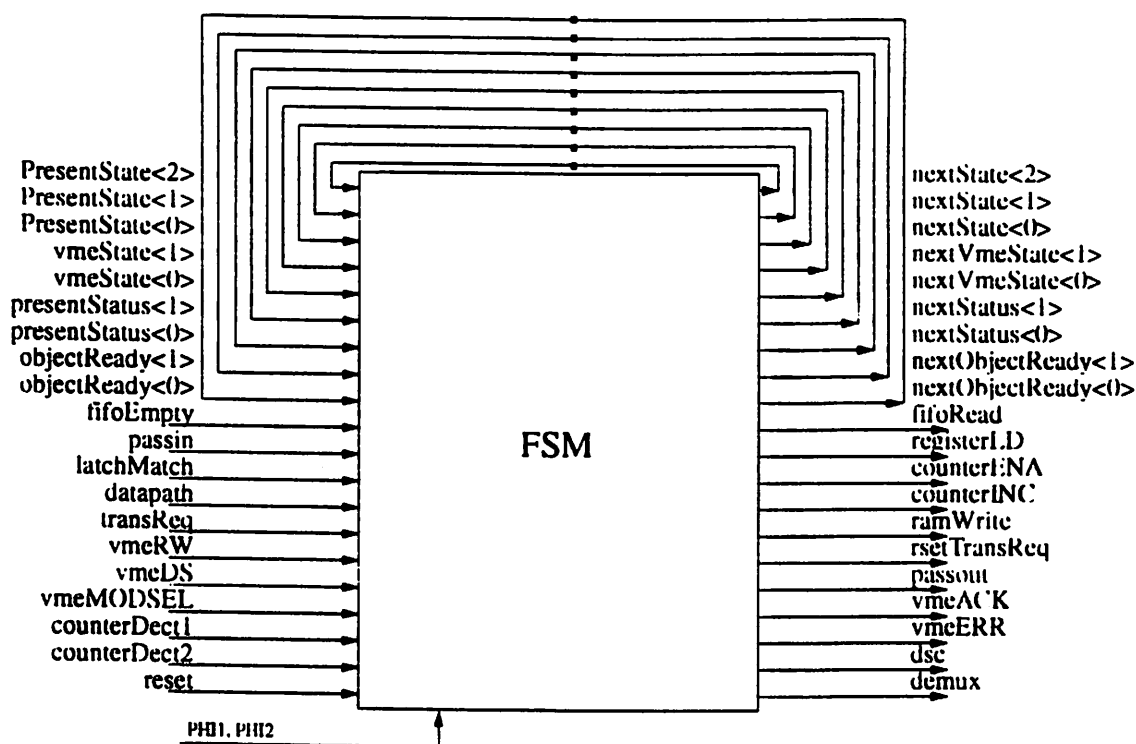


Fig. B-4. Finite state machine input/output graph.

```
MODEL fsm nextState<2:0>, nextVmeState<1:0>, counterENA, counterINC,
  ramWrite, rsetTransReq, fifoRead, registerLD, nextStatus<1:0>,
  nextObjectReady<1:0>, passout, vmeACK, vmeERR, dsc, demux
= passin, presentState<2:0>, fifoEpty, presentStatus<1:0>,
  counterDec1, counterDec2, latchMatch, datapath, transReq,
  objectReady<1:0>, vmeRW, vmeDS, vmeMODSEL, reset,
  vmeState<1:0>;
```

! transReq: vertical blank matched, request to load new parameters from
! RAM into registers.

! rsetTransReq: Used to reset TransReq flip-flop

! Status: 0: No unused parameters in RAM yet.

! 1: New parameters in RAM, not read by host yet.

! 2: Parameters read by host, not modified yet.

! 3: Parameters modified by host, not loaded into registers yet.

!

ROUTINE cntl_gen:

counterENA = 0;

nextStatus = presentStatus;

nextObjectReady = objectReady;

nextVmeState = vmeState;

registerLD = 0;

counterINC = 0;

nextState = presentState;

```

ramWrite = 0;
fifoRead = 0;
demux = 0;
rsetTransReq = 0;
vmeACK = 1;
vmeERR = 1;
dsc = 0;

```

```

! objectReady signify the # of valid fields in VRAM. If only one or
! less valid field data in VRAM, the data from previous stages are
! passed directly to the next stage.
! The passout signal is used to control a Flip Flop outside which is
! triggered by the vinv rising edge.
! In this way, the passin remains in a single state ( 1 or 0 ) during
! every field cycle.

```

```

passout = 1;
if ( objectReady GTR 1 ) THEN passout = 0;

```

```

! VME response loop. This finite state machine always response
! to the interrupt from VME bus first.

```

```

if ((NOT vmeMODSEL AND (((presentStatus EQL 1) AND vmeRW )
OR ((presentStatus EQL 2) AND NOT vmeRW ))) OR ( vmeState NEQ 0 )) THEN
  BEGIN
    select vmeState FROM
    [0]: BEGIN
      counterENA = 1;
      nextVmeState = 1;
      END;
    [1]: BEGIN
      IF vmeDS THEN nextVmeState = 1 ELSE
        BEGIN
          IF vmeRW THEN
            BEGIN
              vmeACK = 0;
              nextVmeState = 2;
            END
          ELSE
            BEGIN
              ramWrite = 1;
              vmeACK = 0;
              nextVmeState = 3;
            END;
          END;
        END;
      END;
    [2]: BEGIN
      vmeACK = 0;
      IF NOT vmeDS THEN nextVmeState = 2
      ELSE
        BEGIN
          vmeACK = 1;
          IF counterDec12 THEN
            BEGIN
              nextStatus = 2;
              nextVmeState = 0;
              counterENA = 1;
            END
          END
        END
      END;
  END

```

```

ELSE
    BEGIN
        counterINC = 1;
        nextVmeState = 1;
    END;
END;
END;
[3]: BEGIN
    vmeACK = 0;
    IF NOT vmeDS THEN nextVmeState = 3
    ELSE
        BEGIN
            vmeACK = 1;
            IF counterDect1 THEN
                BEGIN
                    nextStatus = 3;
                    nextVmeState = 0;
                    counterENA = 1;
                END
            ELSE
                BEGIN
                    counterINC = 1;
                    nextVmeState = 1;
                END;
            END;
        END;
    END;
ENDSELECT;
END
ELSE

```

! Starting to fetch input from FIFO, and put put to VRAM or RAM,
! depending on the header matched or not.

```

BEGIN
    IF NOT vmeMODSEL THEN vmeERR = 0;
    SELECT presentState FROM

```

! This state is a state when the frontend input latch is empty.
! Fetches data from FIFO if it is not empty. Wait if FIFO is empty.

```

[1]: BEGIN

```

! Loading parameters from RAM into registers, if the parameters is
! already modified by the host, and the transReq flag is set.

```

    IF transReq AND ( presentStatus EQL 3) THEN
        BEGIN
            counterENA = 1;
            nextState = 2;
        END
    ELSE
        BEGIN
            IF fifoEPTY THEN nextState = 1 ELSE
                BEGIN
                    fifoRead = 1;
                    nextState = 4;
                END
            END
        END
    END

```

```

                                END;
                                END;
                                END;

[2]: BEGIN
! Load the parameter into registers
! Here, assume that counter can be access and
! incremented at the same time.

    IF counterDect1 THEN
        BEGIN
            rsetTransReq = 1;
            nextStatus = 0;
            nextState = 1;
            counterENA = 1;
            END
    ELSE
        BEGIN
            registerLD = 1;
            counterINC = 1;
            nextState = 2;
            END;
    END;

[3]: BEGIN
    IF counterDect1 THEN
        BEGIN
            rsetTransReq = 1;
            nextStatus = 0;
            nextState = 4;
            counterENA = 1;
            END
    ELSE
        BEGIN
            registerLD = 1;
            counterINC = 1;
            nextState = 3;
            END;
    END;

[4]: BEGIN

! Send to the data path, and at the same time fetch
! from FIFO.
    IF transReq AND ( presentStatus EQL 3 ) THEN
        BEGIN
            counterENA = 1;
            nextState = 3;
            END
    ELSE
        BEGIN
            IF latchMatch THEN
                BEGIN
                    IF (( presentStatus NEQ 0 ) OR
                        (( passin EQL 1 ) AND ( objectReady
                        EQL 2 ))) THEN
                        ! Wait until the previous parameters
                        ! are modified and consumed.
                        ! Also wait until the VRAM are switched
                        ! to vacate a VRAM.
                        nextState = 7
                END
            END
        END
    END

```

```

ELSE
    BEGIN
    SELECT objectReady FROM
        [0]: nextObjectReady = 1;
        [1]: nextObjectReady = 2;
        [2]: nextObjectReady = 2;
    ENDSELECT;
    counterENA = 1;
    IF fifoEMPTY THEN nextState = 5 ELSE
        BEGIN
        fifoRead = 1;
        nextState = 6;
        END;
    END;
END
ELSE
    BEGIN
    IF datapath THEN
        BEGIN
        dsc = 1;
        IF fifoEMPTY THEN nextState = 1 ELSE
            BEGIN
            fifoRead = 1;
            nextState = 4;
            END;
        END
        ELSE nextState = 4;
        END;
    END;
END;

[5]: BEGIN
! Load an item from FIFO
    IF fifoEMPTY THEN nextState = 5 ELSE
        BEGIN
        fifoRead = 1;
        nextState = 6;
        END;
    END;

[6]: BEGIN
    IF counterDec2 THEN
        BEGIN
        nextState = 1;
        IF datapath THEN
            BEGIN
            dsc = 1;
            IF fifoEMPTY THEN nextState = 1 ELSE
                BEGIN
                fifoRead = 1;
                nextState = 4;
                END;
            END;
        ELSE
            nextState = 4;
            counterENA = 1;
        END
    ELSE
        BEGIN

```

```

        counterINC = 1;
        demux = 1;
        ramWrite = 1;
        IF fifoEPTY THEN nextState = 5 ELSE
            BEGIN
                fifoRead = 1;
                nextState = 6;
            END;
        END;
    [7]: BEGIN
        ! Wait until parameters are written into registers
        IF ((presentStatus EQL 1) OR (presentStatus EQL 2) OR
            ((presentStatus EQL 3) AND NOT transReq) OR
            ((presentStatus EQL 0) AND (passin EQL 1) AND
            (objectReady EQL 2))) THEN
            nextState = 7
        ELSE
            ! Start transfer the parameters into register.
            IF ( presentStatus EQL 3 ) THEN
                BEGIN
                    counterENA = 1;
                    nextState = 3;
                END
                ! Allow to read new parameters
            ELSE
                BEGIN
                    SELECT objectReady FROM
                        [0]: nextObjectReady = 1;
                        [1]: nextObjectReady = 2;
                        [2]: nextObjectReady = 2;
                    ENDSELECT;
                    counterENA = 1;
                    IF fifoEPTY THEN nextState = 5 ELSE
                        BEGIN
                            fifoRead = 1;
                            nextState = 6;
                        END;
                    END;
                END;
            ENDSELECT;
        END;
    END;
if RESET THEN
    BEGIN
        passout = 1;
        counterENA = 1;
        nextStatus = 0;
        nextObjectReady = 0;
        nextState = 1;
        nextVmeState = 0;
        registerLD = 0;
        counterINC = 0;
        ramWrite = 0;
        fifoRead = 0;
        demux = 0;
        rsetTransReq = 0;
        vmeACK = 1;
        vmeERR = 1;
        dsc = 0;
    
```


END;
ENDROUTINE;
ENDMODEL fsm;

B.3 VCP combinational logic block implementation

MODEL comb_log

SC, B_SEL13, B_SEL45, B_SEL1345, B_SEL2, F_SEL13, F_SEL2, SEL_Y
 = FLAG, CONSISTENCY, ALPHA1, ALPHA0, F_ALPHA1L, F_ALPHA1M, FGE, FEQ, BGE,
 BEQ, ulx, uly, lrx, lry, bulx, buly, blrx, blry, pass, y_uv;

```

ROUTINE logic_gen;
  STATE F_1, F_2, F_3, B_1, B_2, B_3, B_4, B_5, PRESENCE, range, FGT,
    BGT, FLS, BLS, F_ALPHA1;
  F_ALPHA1 = F_ALPHA1L AND F_ALPHA1M;
  SC = ulx AND uly AND lrx AND lry;
  FGT = FGE AND (NOT FEQ);
  BGT = BGE AND (NOT BEQ);
  FLS = NOT FGE;
  BLS = NOT BGE;
  range = SC AND bulx AND buly AND blrx AND blry;
  PRESENCE = range AND (NOT pass) AND (NOT ALPHA0) AND
    ( NOT ( ( NOT y_uv ) AND (NOT FLAG) AND
      CONSISTENCY));
  F_1 = FLS AND PRESENCE;
  F_2 = FEQ AND PRESENCE;
  F_3 = (NOT PRESENCE) OR FGT;
  B_1 = ( FGT OR ( FEQ AND ALPHA1 AND (NOT F_ALPHA1))) AND BLS AND
    PRESENCE;
  B_2 = BEQ AND ( FGT OR ( FEQ AND ALPHA1 AND (NOT F_ALPHA1))) AND
    PRESENCE;
  B_3 = (NOT PRESENCE) OR BGT OR ( FEQ AND (NOT ALPHA1) AND (NOT
    F_ALPHA1));
  B_4 = FEQ AND ALPHA1 AND F_ALPHA1 AND PRESENCE;
  B_5 = ( FLS OR ( FEQ AND (NOT ALPHA1) AND F_ALPHA1 ) ) AND
    PRESENCE;
  F_SEL13 = NOT F_3;
  F_SEL2 = F_2;
  B_SEL13 = B_1;
  B_SEL45 = B_4;
  B_SEL1345 = NOT (B_4 OR B_5);
  B_SEL2 = B_2;
  SEL_Y = y_uv;
ENDROUTINE;
ENDMODEL;

```

B.4 VCP scanning register path for testing

- PHI1F, PHI2F path:

```
SCANINF -> VRAM_IN_PIXEL ( 0 : 7 ) -> L_ALPHA ( 0 : 2 ) -> VRAM_IN_ALPHA ( 0:3 ) ->
TAU ( 0:3 ) -> L_ALPHA ( 3 ) -> BLANK ( 0:1 ) -> L_TAU ( 0:6 ) -> ALPHA0 ( 0 ) -> BLANK ( 0:2 ) ->
L_PIXEL ( 0:10 ) -> L_X ( 6:10 ) -> L_X ( 0:5 ) -> L_Y ( 4:8 ) -> L_Y ( 0:3 ) -> FIN ( 0:3 ) -> BIN ( 0:3 ) ->
PRI ( 0:3 ) -> SC_0 -> B_SEL13_0 -> BLANK -> B_SEL1345_0 -> B_SEL2_0 -> F_SEL13_0 ->
F_SEL2_0 -> BLANK -> SC_1 -> B_SEL13_1 -> BLANK -> B_SEL1345_1 -> B_SEL2_1 -> F_SEL13 ->
F_SEL2_1 -> BLANK -> SCLOAD -> BLANK -> v -> BLANK -> ALPHA0 -> PASS -> datapath ->
DSC -> BLANK ( 0:3 ) -> SCANOUTF
```

- PHI1, PHI2 path:

```
SCANIN -> REG_IN ( 0 : 11 ) -> ADDR ( 0:3 ) -> RegLD -> VmeACK -> VmeERR -> state ( 0:2 ) ->
vstate ( 0:1 ) -> status ( 0:1 ) -> objReady ( 0:1 ) -> SCANOUT
```

B.5 VCP chip pin diagram

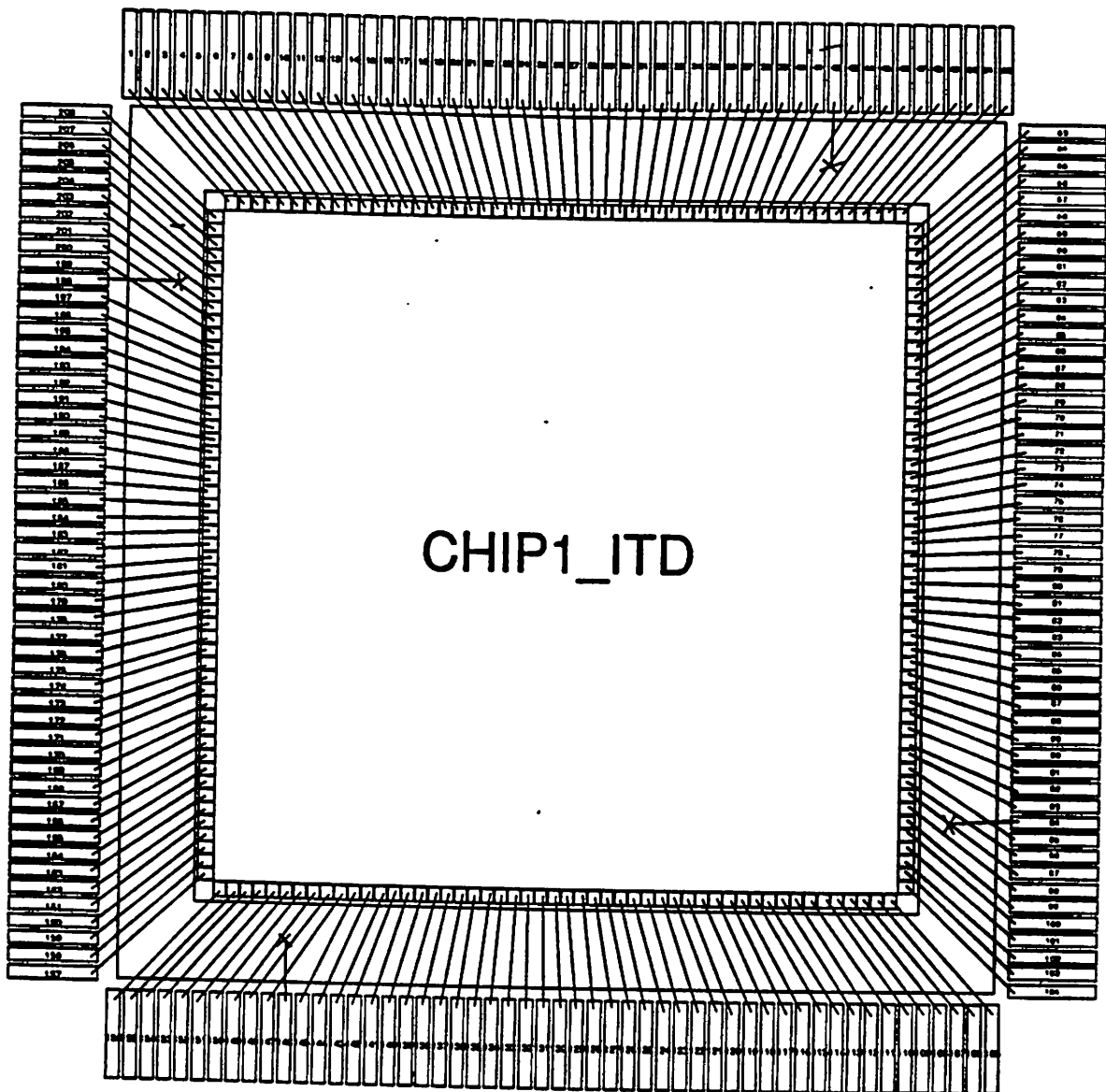


Fig. B-5. VCP chip pin diagram using 208 pin PGA package.

PIN	Name	PIN type
1	RESET	IN
2	VRAM1[0]	I/O
3	VRAM1[1]	I/O
4	VRAM1[2]	I/O
5	VRAM1[3]	I/O
6	VRAM1[4]	I/O
7	VRAM1[5]	I/O
8	VRAM1[6]	I/O
9	VRAM1[7]	I/O

9	VRAM1[7]	I/O
10	VRAM1[8]	I/O
11	F_PIXEL_IN[9]	IN
12	F_PIXEL_IN[10]	IN
13	B_PIXEL_IN[0]	IN
14	B_PIXEL_IN[1]	IN
15	B_PIXEL_IN[2]	IN
16	B_PIXEL_IN[3]	IN
17	B_PIXEL_IN[4]	IN
18	B_PIXEL_IN[5]	IN
19	B_PIXEL_IN[6]	IN
20	B_PIXEL_IN[7]	IN
21	B_PIXEL_IN[8]	IN
22	B_PIXEL_IN[9]	IN
23	B_PIXEL_IN[10]	IN
24	PHI1	IN
25	PHI2	IN
26	Vdd	Vdd
27	GND	GND
28	F_PIXEL_OUT[0]	OUT
29	F_PIXEL_OUT[1]	OUT
30	F_PIXEL_OUT[2]	OUT
31	F_PIXEL_OUT[3]	OUT
32	F_PIXEL_OUT[4]	OUT
33	F_PIXEL_OUT[5]	OUT
34	F_PIXEL_OUT[6]	OUT
35	F_PIXEL_OUT[7]	OUT
36	F_PIXEL_OUT[8]	OUT
37	F_PIXEL_OUT[9]	OUT
38	F_PIXEL_OUT[10]	OUT
39	B_PIXEL_OUT[0]	OUT
40	B_PIXEL_OUT[1]	OUT
41	B_PIXEL_OUT[2]	OUT
42	NOT CONNECTED	NC
43	B_PIXEL_OUT[3]	OUT
44	B_PIXEL_OUT[4]	OUT
45	B_PIXEL_OUT[5]	OUT
46	B_PIXEL_OUT[6]	OUT
47	B_PIXEL_OUT[7]	OUT
48	B_PIXEL_OUT[8]	OUT
49	B_PIXEL_OUT[9]	OUT
50	B_PIXEL_OUT[10]	OUT
51	Vdd	Vdd
52	GND	GND
53	KEEP	IN
54	KEEPF	IN
55	SCANF	IN
56	F_PIXEL_IN[0]	IN
57	F_PIXEL_IN[1]	IN
58	F_PIXEL_IN[2]	IN
59	F_PIXEL_IN[3]	IN
60	F_PIXEL_IN[4]	IN
61	F_PIXEL_IN[5]	IN
62	F_PIXEL_IN[6]	IN
63	F_PIXEL_IN[7]	IN
64	F_PIXEL_IN[8]	IN
65	VRAM1[9]	I/O
66	VRAM1[10]	I/O
67	VRAM1[11]	I/O

68	VRAM2[0]	I/O
69	VRAM2[1]	I/O
70	VRAM2[2]	I/O
71	VRAM2[3]	I/O
72	VRAM2[4]	I/O
73	VRAM2[5]	I/O
74	GND	GND
75	Vdd	Vdd
76	VRAM_SEL	OUT
77	DELAY[0]	IN
78	DELAY[1]	IN
79	DELAY[2]	IN
80	DELAY[3]	IN
81	DELAY[4]	IN
82	SCANINF	IN
83	SCANOUTF	OUT
84	SCANOUT	OUT
85	SCANIN	IN
86	SC1	OUT
87	SC2	OUT
88	DTACKINV	IN
89	VmeACK	OUT
90	VmeERR	OUT
91	F_ALPHA_OUT[0]	OUT
92	F_ALPHA_OUT[1]	OUT
93	F_ALPHA_OUT[2]	OUT
94	NOT CONNECTED	NC
95	F_ALPHA_OUT[3]	OUT
96	F_ALPHA_OUT[4]	OUT
97	F_ALPHA_OUT[5]	OUT
98	F_ALPHA_OUT[6]	OUT
99	F_ALPHA_OUT[7]	OUT
100	F_ALPHA_OUT[8]	OUT
101	F_ALPHA_OUT[9]	OUT
102	F_ALPHA_OUT[10]	OUT
103	Vdd	Vdd
104	GND	GND
105	VRAM2[6]	I/O
106	VRAM2[7]	I/O
107	VRAM2[8]	I/O
108	VRAM2[9]	I/O
109	VRAM2[10]	I/O
110	VRAM2[11]	I/O
111	F_ALPHA_IN[0]	IN
112	F_ALPHA_IN[1]	IN
113	F_ALPHA_IN[2]	IN
114	F_ALPHA_IN[3]	IN
115	F_ALPHA_IN[4]	IN
116	F_ALPHA_IN[5]	IN
117	F_ALPHA_IN[6]	IN
118	F_ALPHA_IN[7]	IN
119	F_ALPHA_IN[8]	IN
120	F_ALPHA_IN[9]	IN
121	F_ALPHA_IN[10]	IN
122	F_PR_OUT[0]	OUT
123	F_PR_OUT[1]	OUT
124	F_PR_OUT[2]	OUT
125	F_PR_OUT[3]	OUT
126	GND	GND

127	NOT CONNECTED	NC
128	Vdd	Vdd
129	F_TAU_IN[0]	IN
130	F_TAU_IN[1]	IN
131	F_TAU_IN[2]	IN
132	F_TAU_IN[3]	IN
133	F_TAU_IN[4]	IN
134	F_TAU_IN[5]	IN
135	F_TAU_IN[6]	IN
136	B_TAU_OUT[0]	OUT
137	B_TAU_OUT[1]	OUT
138	B_TAU_OUT[2]	OUT
139	B_TAU_OUT[3]	OUT
140	B_TAU_OUT[4]	OUT
141	B_TAU_OUT[5]	OUT
142	B_TAU_OUT[6]	OUT
143	B_PR_OUT[0]	OUT
144	B_PR_OUT[1]	OUT
145	B_PR_OUT[2]	OUT
146	NOT CONNECTED	NC
147	B_PR_OUT[3]	OUT
148	F_PR_IN[0]	IN
149	VmeBUS[0]	I/O
150	VmeBUS[1]	I/O
151	VmeBUS[2]	I/O
152	F_PR_IN[1]	IN
153	F_PR_IN[2]	IN
154	F_PR_IN[3]	IN
155	GND	GND
156	Vdd	Vdd
157	VmeBUS[3]	I/O
158	VmeBUS[4]	I/O
159	VmeBUS[5]	I/O
160	VmeBUS[6]	I/O
161	VmeBUS[7]	I/O
162	VmeBUS[8]	I/O
163	VmeBUS[9]	I/O
164	VmeBUS[10]	I/O
165	VmeBUS[11]	I/O
166	B_PR_IN[0]	IN
167	B_PR_IN[1]	IN
168	B_PR_IN[2]	IN
169	B_PR_IN[3]	IN
170	B_TAU_IN[0]	IN
171	B_TAU_IN[1]	IN
172	B_TAU_IN[2]	IN
173	B_TAU_IN[3]	IN
174	B_TAU_IN[4]	IN
175	B_TAU_IN[5]	IN
176	B_TAU_IN[6]	IN
177	FIFO_EMPTY	IN
178	Vdd	Vdd
179	GND	GND
180	VmeRW	IN
181	VmeMODSEL	IN
182	VmeDS	IN
183	FIFO_POP	OUT
184	FIFO_IN[0]	IN
185	FIFO_IN[1]	IN

186	FIFO_IN[2]	IN
187	FIFO_IN[3]	IN
188	FIFO_IN[4]	IN
189	FIFO_IN[5]	IN
190	FIFO_IN[6]	IN
191	FIFO_IN[7]	IN
192	FIFO_IN[8]	IN
193	FIFO_IN[9]	IN
194	FIFO_IN[10]	IN
195	FIFO_IN[11]	IN
196	PHI1F	IN
197	PHI2F	IN
198	NOT CONNECTED	NC
199	F_TAU_OUT[0]	OUT
200	F_TAU_OUT[1]	OUT
201	F_TAU_OUT[2]	OUT
202	F_TAU_OUT[3]	OUT
203	F_TAU_OUT[4]	OUT
204	F_TAU_OUT[5]	OUT
205	F_TAU_OUT[6]	OUT
206	SCAN	IN
207	Vdd	Vdd
208	GND	GND