

Parallel Processing Interfaces to Television

by

Frank Kao

M.S. Systems Engineering
Boston University, Boston, MA
December 1991

B.S. Computer Science
Michigan Technological University, Houghton, MI
June 1984

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1995

© Massachusetts Institute of Technology, 1995.
All Rights Reserved

Author
Program in Media Arts and Sciences
May 12, 1995

Certified by
Associate Director, MIT Media Laboratory
Andrew B. Lippman

Accepted by
Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 06 1995

LIBRARIES

Parallel Processing Interfaces to Television

by

Frank Kao

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on May 12, 1995 in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis deals with the problem of presenting and managing multiple live audiovisual data streams for television viewing. Prior work has dealt with content presentation and navigation in a large information space and typically has not concerned itself with the time critical aspect of the data. This thesis extends and modifies these concepts and applies them to entertainment.

A general purpose super computer is programmed to be the digital television prototype. The massive computing capability and the connectivity of the prototype provides the backbone for testing new user interface designs. We took advantage of the computer's internal connectivity to provide multiple perspectives for the most complete and up to date pictorial presentation.

User interface experiments are conducted to explore screen layouts for television viewing. We eliminated image motion and used peripheral vision to watch multiple television channels. An array of one full motion main and three relatively static side channels is a reasonable configuration for watching television. Command input using an externally attached proximity sensor has shown promise as means of controlling the television.

Thesis Supervisor: Andrew B. Lippman
Associate Director, MIT Media Laboratory

The work reported herein is supported by the Television of Tomorrow consortium

Parallel Processing Interfaces to Television

by

Frank Kao

Reader:

Edward H. Adelson
Associate Professor of Vision Science
MIT Media Laboratory

Reader:

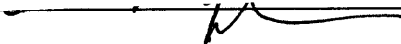
 David Epstein
Manager of Entertainment Applications
T. J. Watson Research Center
International Business Machines Corporation

Table of Contents

Chapter 1

Introduction

1.1 Motivation	9
1.1.1 Problem	11
1.1.2 Approach	12

Chapter 2

Presentation Techniques

2.1 Translucency	15
2.2 Fish-eye Views	16
2.3 Video Streaming	18
2.4 Radar-Sweeping	20

Chapter 3

User Interface Experiments

3.1 Display Spaces	22
3.2 Experiments	24
3.2.1 Motion Elimination	24
3.2.1.1 Cross Fade	25
3.2.1.2 Image Line Scanning	27
3.2.1.3 Video Streaming	28
3.2.2 Content grouping	29
3.2.3 Inputs	30

Chapter 4

Prototype Design

4.1 System Design Issues	33
4.1.1 Multi-Processor Software Design	33
4.1.2 Interface Control Device	34
4.1.3 Multimedia data types synchronization	34
4.2 System Architecture	35
4.2.1 Schematic	35
4.3 Data Paths & Types	37
4.3.1 Live Television	37
4.3.2 Movie Channel	37
4.3.3 Network	38
4.4 User Interface Input - FISH	38
4.5 Software Architecture	40
4.5.1 Distributed Processing	42
4.5.2 Process Shared Memory	43

4.5.3 Process Communications	44
4.6 User Commands	45
4.7 System Data Structures	45
4.7.1 Mailboxes	45
4.7.2 Real-time Queues	46
4.8 Software Processes	48
4.8.1 PVS	49
4.8.2 RS6000	49
4.8.3 Remote UNIX processes	49

Chapter 5

Results

5.1 User Interface	51
5.1.1 Motion Elimination	52
5.1.1.1 Cross Fade	52
5.1.1.2 Line Scans	52
5.1.2 Screen Organization	60
5.1.3 Proximity Sensor	60
5.2 Prototype Design	62

Chapter 6

Conclusions

6.1 User Interface	65
6.2 Parallel Computing	65

Appendix A User Commands	66
--------------------------------	----

Appendix B Inter-Process Mail Messages	67
--	----

Appendix C System Batch Run	69
-----------------------------------	----

Glossary	75
----------------	----

References	76
------------------	----

List of Figures

Figure 1: Computerized Television Interface	10
Figure 2: Thesis Problem Context	12
Figure 3: Translucency Example	15
Figure 4: Fish-eye Example	16
Figure 5: Video Streamer Example	19
Figure 6: Radar Sweeping Example	20
Figure 7: User Interface Space sample setup	22
Figure 8: Cross Fade Weighting Factor Calculation	25
Figure 9: Cross Fade Example	26
Figure 10: Video Streaming Algorithms	28
Figure 11: Content Grouping Example	29
Figure 12: Keyboard Menu	30
Figure 13: FISH sensor organization	31
Figure 14: Context schematic diagram	35
Figure 15: NTSC signal path	36
Figure 16: Fish Communication Setup	38
Figure 17: System Software Architecture	40
Figure 18: PVS Process Example	42
Figure 19: PVS Process Execution Sequence	43
Figure 20: Mailboxes for process communication	44
Figure 21: Mailbox Data Structure	45
Figure 22: Real time queue access strategy	46
Figure 23: Image Line Scanning	54
Figure 24: 16:9 Screen Setup	56
Figure 25: Pyramid Setup	57
Figure 26: Video Streamer	58
Figure 27: Content Grouping	59

List of Tables

Table 1: System Display Space Summary	23
Table 2: Image Scanning Algorithm	27
Table 3: Real Time Queues summary	47
Table 4: Summary of software processes	48
Table 5: Motion Elimination Results Summary	55

Acknowledgments

I would like to thank all the people for whom without them this work would not be possible.

First, thanks to Andy Lippman, my advisor, for the good ideas and direction, and an unique perspective in research and life; to the thesis readers, Ted Adelson, and Dave Epstein, for the patience to read it and the comments they provided. To the garden crew, Klee, Wad, and Henry for their advise and technical support for setting up the system and troubleshooting problems with the PVS and RS6000s.

Special thanks to the engineers of the Power Visualization System. Dave Epstein, the fearless leader for making the system and technical support available. Curt McDowell, from whom I learned so much about color conversion and parallel programming. Jeff Hamilton, and Nick Dono for their patience and many long hours of phone consultations.

Special thanks to my fiancee Grace Hu, for her support over this last 2 years. Your support and patience have made the transitions to student life much more bearable and is the main reason that have made this work possible. Grace, I love you.

Chapter 1

Introduction

1.1 Motivation

The inevitable replacement of televisions with powerful computers connected to high speed networks has made watching television more interesting and a lot more confusing. (see Figure 1) The viewer is suddenly faced with an unlimited choice of contents available from all over the network. The simple one-way television to viewer interface is replaced by the highly interactive and sometimes tedious Q&A session of a computer interface. Channel surfing over hundreds of channels is simply too confusing and time consuming. The goal is to use the computer to assist in searching the contents of all the channels and present the results in a simple and pleasing manner suitable for television viewing.



#6



#5



Network Channel



#1



#7

TV Channel



#8



Television of Tomorrow



#2



#3

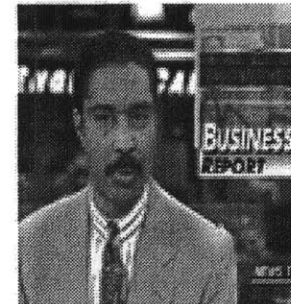
News Space



#4



#9



#10



#11

Movie Channels

Kamei told a news conference. "It is the

MIT Media Laboratory

Figure 1: Computerized Television Interface

Traditional user interface research has experimented with many strategies [Ref 1,2,3,4,5,6,7,8,9,10] for navigating and searching in a large information space. These strategies are usually performed on one large still image, however they can be converted for television viewing applications which have many smaller and fast moving images.

This thesis extends and modifies these strategies in adapting them for the digital television interface. The objective is to find the best television interface that provides the latest and most complete story with minimum effort and viewing distraction.

1.1.1 Problem

There are two challenges: a) to design a digital television prototype to be used as a test bed, and b) to build the best content browser for watching television.

Existing systems and networks have barely sufficient bandwidth to handle live video. The engineering challenge is to build the digital television from existing equipment to process multiple live audiovisual data streams. The prototype must have sufficient bandwidth left over from image processing so that different interface designs can be experimented with.

Some prior works on content navigation and browsing tools have relied heavily on using exotic 3-D computer graphics. In contrast to television, these applications are typically not concerned with processing live audiovisual data. The computing cost of using these techniques for just watching television are enormous and simply cannot be justified. The discontinuity of contents between

channels and the availability of hundreds of channels further complicates the problem. The traditional “window” computer interface is too complicated, requires too much interaction and does not work across the room for television applications. The challenge is to find an intuitive interface that is easy to use, and to provide a concise and complete description of its contents on the screen.

1.1.2 Approach

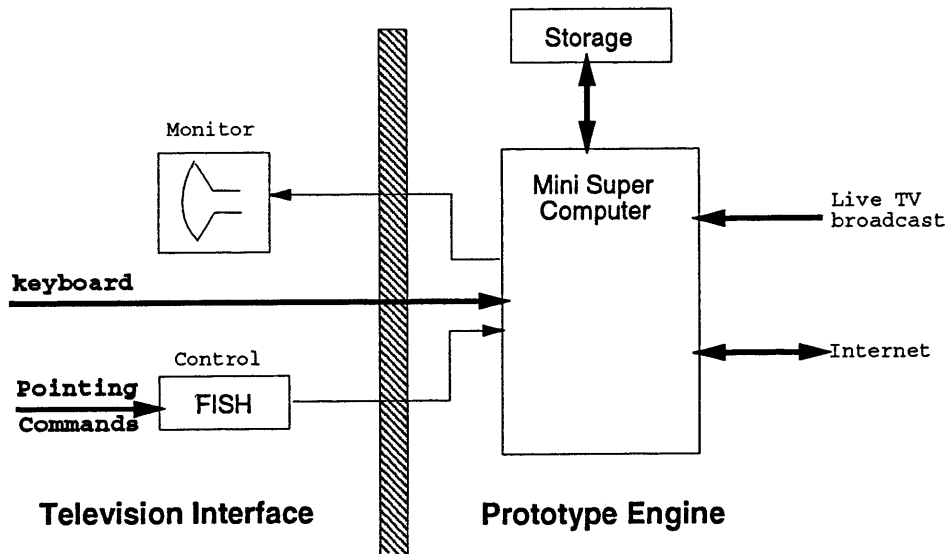


Figure 2: Thesis Problem Context

The digital television prototype is realized by programming a general purpose super computer (See Figure 2). The tremendous processing capabilities of the computer are harnessed to process the live audiovisual data streams. The prototype design strategy uses distributed processing with shared memory techniques within the system, and message passing techniques across different system platforms. The design decision balances acceptable viewing resolution and the technique’s computing requirements.

The goals of the user interface are: 1) to allow viewer to watch as many channels as possible without stress and distraction, 2) simple, and “non-window” type interface, 3) easy channel switching and navigating, 4) preservation of content time line, and smooth transition between frames, 5) to provide multiple story perspectives to enhance content understanding and 6) a simple television control.

Peripheral vision is used to increase the number of channels watched simultaneously. Peripheral vision is attracted to motion, and many moving images prove to be very distracting. Slow image line scanning technique analogous to the “radar-sweeping metaphor” [Ref 5] is developed and used to reduce the motion of the images between frame updates. Intermediate image frames are discarded to preserve the content’s time line. Screen experiments include varying channel sizes, border coloring, and spatial and content organization to explore viewer criteria for watching television.

The results of this thesis present: 1) a simple and usable digital television interface and 2) an experimental prototype design.

The rest of the thesis is organized as follows:

Chapter 2 - survey of research done on multimedia user interface designs.

Chapter 3 - experiments conducted for the television interface.

Chapter 4 - digital television prototype design.

Chapter 5 - experiments conducted and prototype design.

Chapter 6 - conclusions and future works.

A glossary of the technical terms used is appended at the end of this thesis.

Chapter 2

Presentation Techniques

This chapter contains a survey of the major trends and techniques used in current user interface research. The main objective of these techniques is to search for an effective presentation style to aid content understanding and to navigate in the vast information landscape. A brief description and overview of each technique is included. Interested readers are encouraged to go to the references for details on each technique. Familiar and well known techniques such as zooming, and tree structure knowledge presentation are readily available [ref 10] and are not discussed.

2.1 Translucency

Translucent techniques and see-through tools preserve the overall context of the image space while providing a more detailed sub-space for viewing. [ref 1,2, 3,4,6] Translucency is achieved by applying varying weights to selected areas of the display. The shape, structure, color and the weight of the translucent patches can be used to represent additional information.

Analogous to the white boards, translucent techniques allow multiple bodies of information to be displayed without interference. The preservation of the overall image space makes these techniques favorable for landscape navigation. Translucent techniques also provide smooth transitions from zooming in and out of selected areas. Figure 3 presents an example of translucency.



Figure 3: Translucency Example

2.2 Fish-eye Views

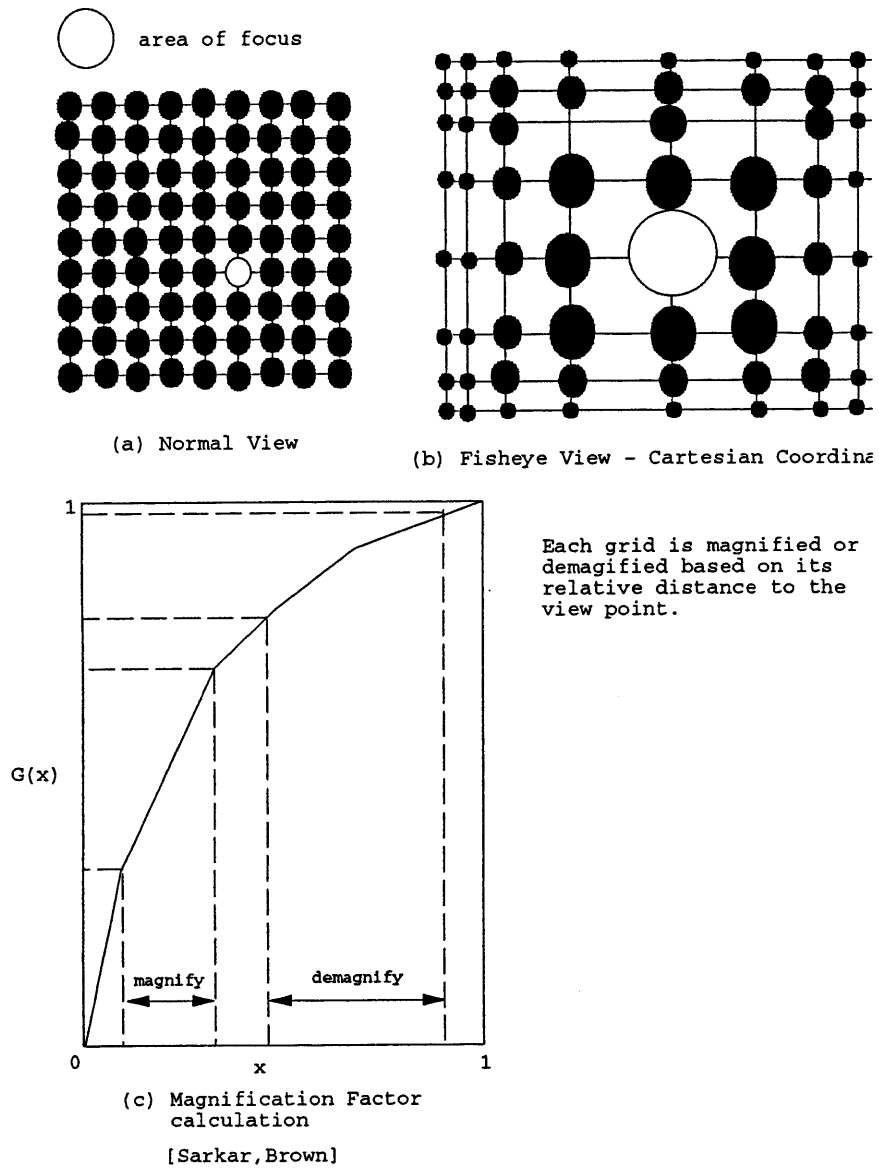


Figure 4: Fish-eye Example

The fish-eye lens is a very wide angle lens that shows objects nearby in detail while showing remote regions in successively less detail (see Figure 4) [Ref 8]

The fish-eye approach to content browsing has the advantage of preserving the context while showing large details for areas of interest.

A 2-step process is used to generate the fish-eye views. First, a geometric transformation to the normal view is applied to reposition vertices and magnify and de-magnify areas close to and far away from the focus, respectively. Each vertex has a position specified by its normal coordinates, and a size which is the length of a side of the bounding box of the vertex. Each vertex is assigned a number to represent its relative importance in the global structure. This number is the *a priori importance* or the API of the index.

A position of a vertex depends on its position in the normal view and its distance from the focus. The size of the vertex depends on its distance from the focus, its size in the normal view and its API. The amount of detail displayed in the vertex depends on its size in the fish-eye view.

$$P_{\text{feye}(v,f)} = F_1(P_{\text{norm}(v)}, P_{\text{norm}(f)})$$

The position vertex v is a function of its position in the normal coordinates and the position of the focus. F_1 is the geometric transformation function.

Secondly, the API of the vertices is used to obtain their final size detail and visual work.

$$S_{\text{feye}(v,f)} = F_2(S_{\text{norm}(v)}, P_{\text{norm}(v)}, P_{\text{norm}(f)}, \text{API}(v))$$

The size of vertex v is a function of its size and position in the normal coordinates, the position of the focus and its API.

$$DTL_{feye(v,f)} - F_3(S_{feye(v,f)}, DTL_{max(v)})$$

The amount of detail depends on the size of v and the maximum detail that can be displayed.

$$VW_{(v,f)} = F_4(D_{normal(v,f)}, API(v))$$

The visual worth of vertex v , depends on the distance between v and the focus in the normal coordinates and the vertex v 's API.

The geometric transformation functions F_1 , F_2 , F_3 and F_4 are application dependent and are not fixed. A sample set of formulas is shown in Reference 8.

2.3 Video Streaming

In a "video streamer", each new video frame is displayed slightly offset from the previous frame creating the visual effect of a three dimensional video cube. The top and left edges of each frame are preserved as the subsequent frame is displayed. (see Figure 5) The edges show events that have occurred in each frame and serve as a time marker.

In addition to being a timer marker, the video streamer also provides an overview of all the events that have occurred during the last several seconds. [Ref 7, 9]

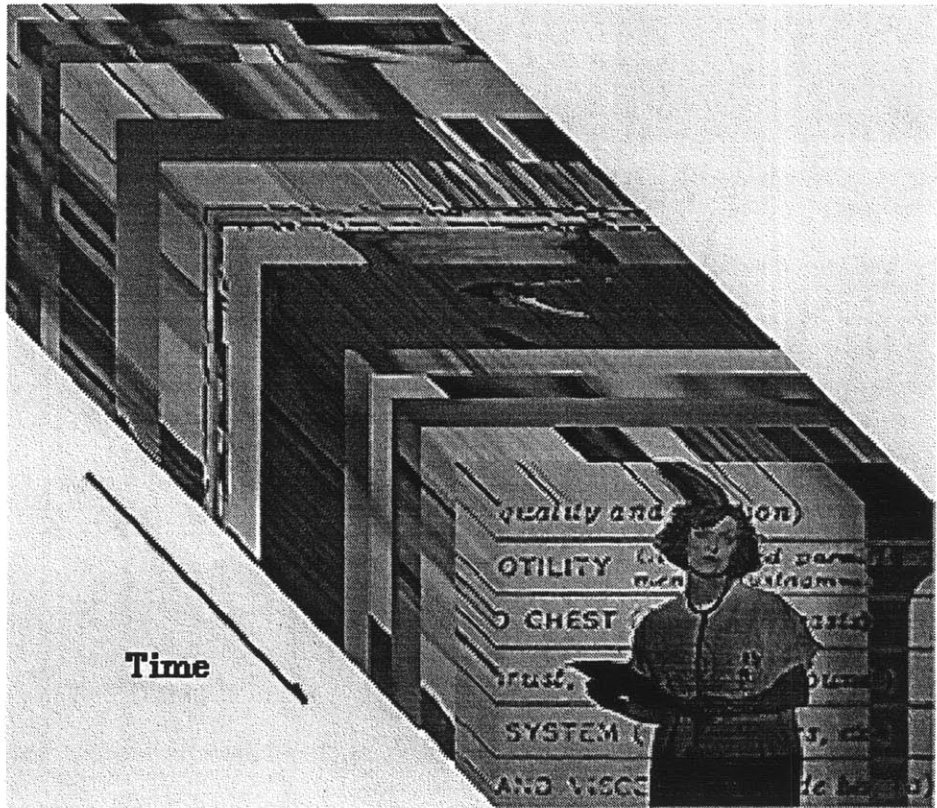


Figure 5: Video Streamer Example

2.4 Radar-Sweeping

This technique has been used for visualizing the dynamics of the program behavior such as memory allocation. [Ref 5] A series of rays emanates from the center at successive intervals. Each ray contains the state of the system behavior at that time interval. Each successive ray is plotted at an angular increment (constant) from the previous one. Additional system information can be incorporated by applying different color, line width, and ray types.

This technique does not require massive computing bandwidth, and is relatively simple to implement.

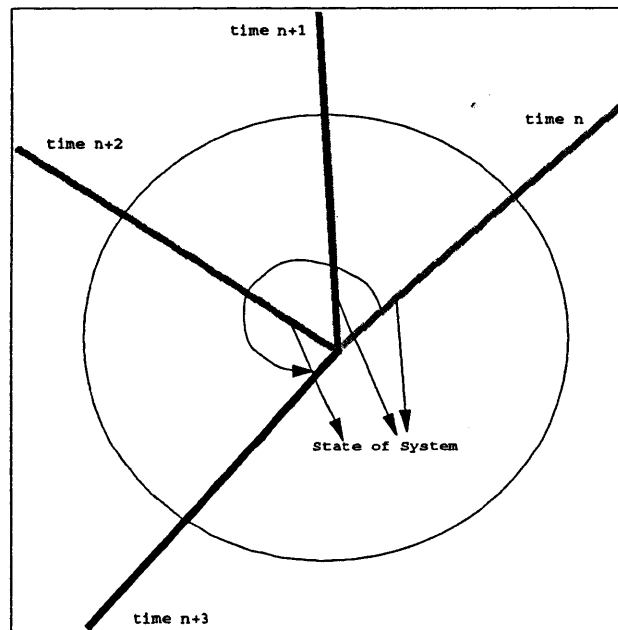


Figure 6: Radar Sweeping Example

Figure 6 presents an example of this technique. Each radar sweep reveals the state of the system at that time interval. A complete sweep reveals the different states of the system at each time interval.

Chapter 3

User Interface Experiments

This chapter presents the user interface experiments conducted. The goal is to create a “video cocktail party” so that a viewer can direct his/hers focus of attention to one of the many choices. Multiple perspectives are provided to enhance content understanding.

The user interfaces are concerned with three issues: 1) eliminate motion and content distraction, 2) avoid information overload, and 3) reduce computing requirements.

This chapter is organized as follows:

Section 3.1 discusses the display spaces for the user interface.

Section 3.2.1 talks about the motion elimination experiments conducted

Section 3.2.2 discusses the screen content grouping experiment

Section 3.2.3 discusses system input experiments.

3.1 Display Spaces

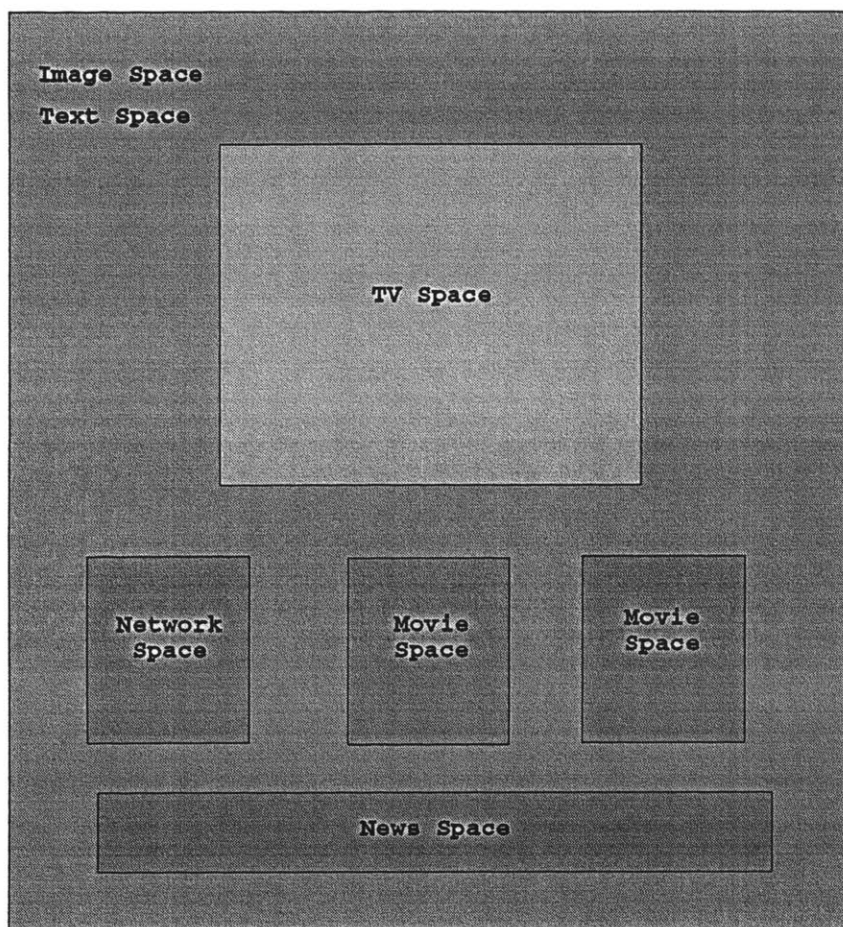


Figure 7: User Interface Space sample setup

The entire television screen is divided into several different and distinct display spaces. (see Figure 7) Each data stream is allocated a distinct display space in the system. These spaces can be overlapped, subsampled, or mathematically transformed. The organization and placement of these spaces depends entirely on the user selection. A brief description of each of the display spaces is summarized in Table 1.

Table 1: System Display Space Summary

Name	Output Type	Input Type	Description
TV Space	video	Live D1	Live television broadcast
Movie Space	video	D1	Display movie selections
News Space	text	Test file	News brief captured from live wires
Image Space	still image	pbm raw image files	Still images, and can be displayed at any part of the display screen
Network Space	video	MJPEG compressed image file	Channel reserved for display network video
Text Space	text	user keyboard entry text	User messages, can be displayed anywhere on the screen

3.2 Experiments

3.2.1 Motion Elimination

The objective of these experiments is to watch multiple television channels without being distracted by the activities within each channel. The motion of the side channels are reduced so that the side images are up to date but don't distract by their motion. In addition, the time line of the channel's content are preserved for an accurate and up to date story reporting.

This section discusses the motion eliminating algorithms developed for the system. These algorithms are then compared for their ability to:

- 1) eliminate motion,
- 2) provide smooth frame transition, and
- 3) have low computing requirements.

3.2.1.1 Cross Fade

The two end frames slowly “dissolves” from one into the other. A smooth transition is done by applying a weighting factor for displaying each frame. This weighting factor is based on the position on the time line for the displayed frame. The weighting factor calculation is shown in Figure 8.

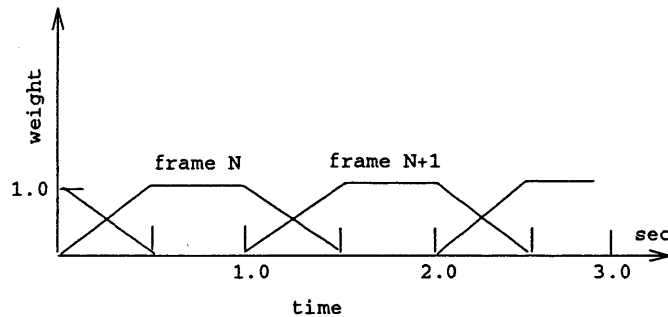


Figure 8: Cross Fade Weighting Factor Calculation

Figure 9 is an example of how the weighting factor is applied to achieve the cross fade. The display of each image frame can be divided into 3 segments. During the first and the last third time segment, the previous and the current frame are dissolved together. The weighting factor for each frame is chosen based on the cross slope. The image frame is displayed weighted at its entirety during the second third of the segment.



Image 1 * 100%
Image 2 * 0%



Image 1 * 70%
Image 2 * 30%

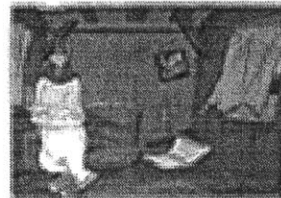


Image 1 * 30%
Image 2 * 70%

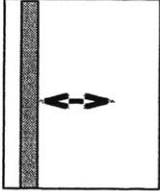
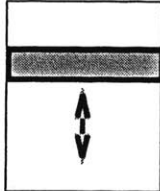
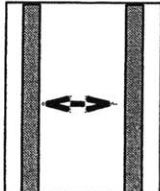


Image 1 * 0%
Image 2 * 100%

Figure 9: Cross Fade Example

3.2.1.2 Image Line Scanning

Table 2: Image Scanning Algorithm

<p>Left Scan</p> <p>Right Scan</p> <p>Left-Right Scan</p>	 <p>Vertical image line is displayed during each scan pass. Each vertical scan line have width of 1 pixel</p>
<p>Up Scan</p> <p>Down Scan</p> <p>Up - Down Scan</p>	 <p>An entire image line is displayed for each pass. The image line have height of 1 pixel.</p>
<p>Window Open</p> <p>Window Close</p> <p>Window Open & Close</p>	 <p>2 vertical scan lines are displayed. The scan lines are 1 pixel in width.</p>

These techniques are similar to the “radar-sweeping metaphor” [Ref 5] in that they display only 1 or 2 scan lines of the next image during each display pass. The next image is “swept” onto the old image. Table 2 presents a summary of the description for each of the algorithm developed.

3.2.1.3 Video Streaming

This technique is used to preserve the temporal contents of the displayed frames. A historic record of the images displayed is kept by saving the edges of each frame.[Ref 7,9] Figure 10 presents an example of the video streaming technique.

Two experiments are conducted 1) preserve only the left most and the top-most pixel for each image frame, and 2) preserve the position of the pixel specified, and eliminate all other pixels. The objective of these experiments is to find the areas of each frame that contains the most relevant content information.

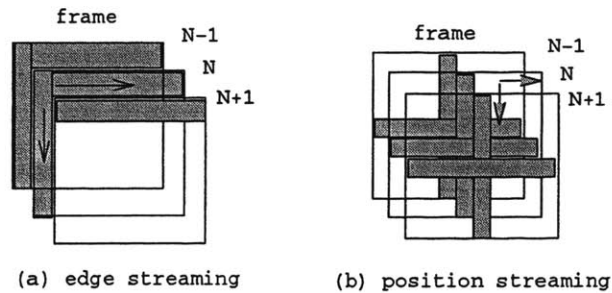


Figure 10: Video Streaming Algorithms

3.2.2 Content grouping

Similar contents are grouped together for easy search and identification. Figure 11 presents an example of content grouping. Multiple perspectives of the same content are provided for additional information support.

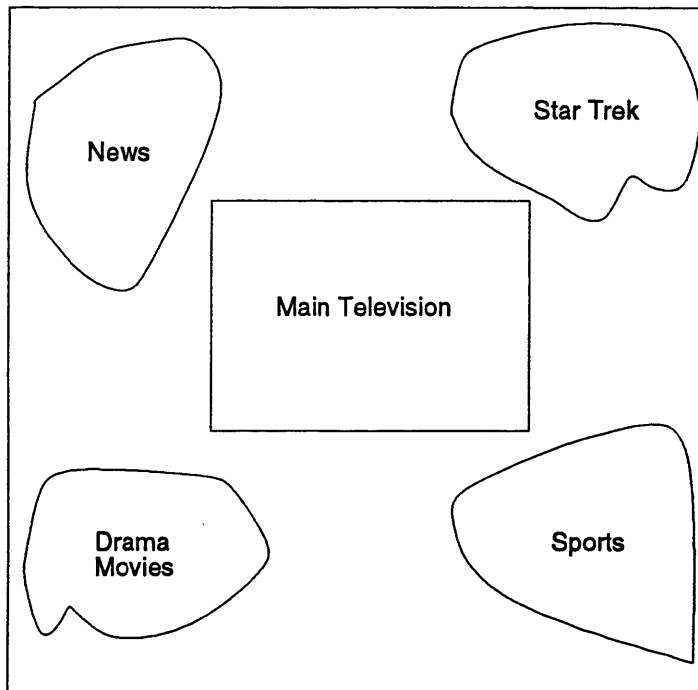


Figure 11: Content Grouping Example

3.2.3 Inputs

Two style of system inputs are implemented. One is the traditional menu keyboard input, and the second using a proximity sensor.

The traditional keyboard menu interface is shown in Figure 12. User enters the precise commands by selecting the desired menu choices.

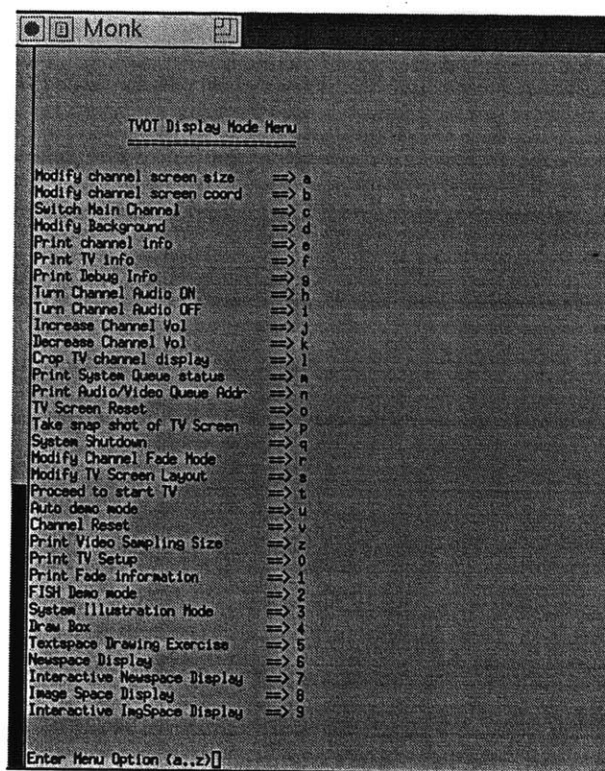
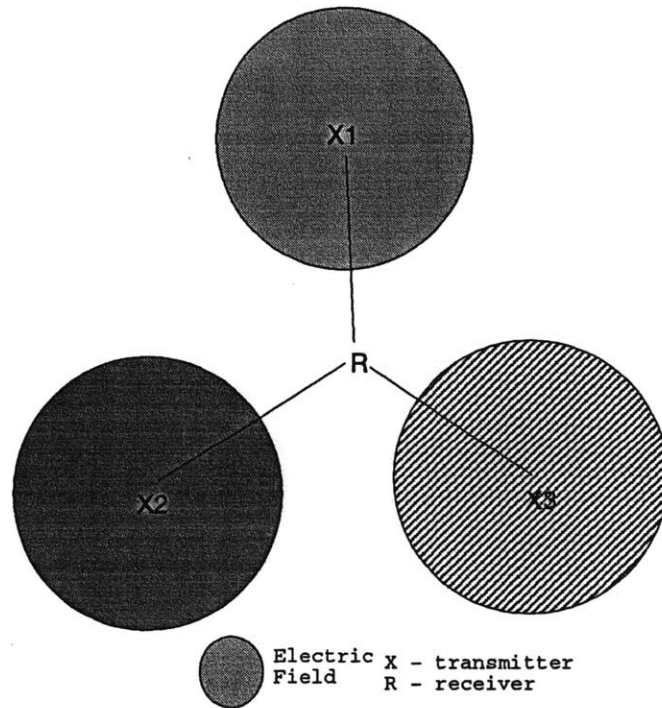


Figure 12: Keyboard Menu

The second interface uses the FISH [Ref 12], a proximity sensor as the television controller. The FISH is attached to the RS232 port of the user workstation. The FISH is a proximity sensor that consists of three transmitters and one receiver and it is spatially configured as shown in Figure 13 (a). Proximity readings consists of four bytes(Figure 13 b) representing the proximity of the view-

er's finger with respect to each of the transmitters. The system then uses these readings to determine the exact position of the viewer's finger in order to interpret the viewer's command. Details of the FISH setup is discussed in section 4.4.

The television screen is divided into three different electric field zones.(see Figure 13(a)) The viewer points to the monitor to control the television. The current state of the system determines the commands for each of the proximity zones. For example, when the user points at the slowed version of a movie channel performing fading between images, the movie then speeds up to its full rate. Further pointing at the channel switches that channel to become the new main display channel.



(a) FISH Setup & Proximity Zones

Xmter 1	Xmter 2	Xmter 3	Xmter 4
---------	---------	---------	---------

(b) FISH Sensor Data

Figure 13: FISH sensor organization

Chapter 4

Prototype Design

This chapter presents the digital television prototype design. The prototype processes multiple audiovisual data streams and is the test bed for experimenting with new user interface designs. The digital television prototype is constructed from IBM's Power Visualization System (PVS). The details of the system architectures, programming and system internal are discussed in reference 11.

Section 4.1 discusses parallel process designs.

Section 4.2 contains the system's hardware connection setup.

Section 4.3 discusses the system's data paths.

Section 4.4 contains the hardware setup connecting the FISH. [Ref 12]

Section 4.5 is the software architecture of the system.

The rest of the chapter then discusses the software data structures and processes designed to build the prototype.

4.1 System Design Issues

4.1.1 Multi-Processor Software Design

The parallel system architecture (PVS) allowed two approaches for software design: a) fine-grained, where each processor executes an independent component from a large single program, and b) coarse-grained, where each processor executes independent programs and shares the results with the other processors.

In the fine-grained computing scenario, the individual processing elements are easier to control because they are all executing the same program on different independent data elements. [Ref 15] The individual outputs are then combined for the final result. The problem with this scenario is that the entire system resource is dedicated to a single program task. The system is committed to using all the system resources assigned to it, even though the extra resources only marginally improve the system performance. Tailoring of system resources using a single program is too complex and cannot scale easily. [Ref 11, 15]

The coarse-grained scenario tailors the system resources by assigning “process groups” for each program. [Ref 11] Each group is assigned a fixed amount of system resources for the entire duration of program execution. Fine-grained program execution then takes place within each of the process groups. The problems are that the software designed is not robust and scalable, and process synchronization between different programs is hard to achieve. [Ref 15]

The goal is to find the optimum system design by combining the two approaches in order to satisfy the system performance requirements and to come up with a software design that is both robust and scalable.

4.1.2 Interface Control Device

FISH [Ref 12], the proximity sensor is explored as an alternative television control device. The technical problem is how to integrate this device into the system so that the exact position of the viewer's hand can be located. The sensor reading needs to be relayed back into the system for proper command interpretation. Section 4.4 contains the detail discussion of setting up this device.

The second problem deals with the interpretation of the viewer's intentions. Currently the proximity sensor acts only as an I/O device relaying the position of the viewer's finger or pointing direction and does not attempt to interpret the command. The viewer command depends entirely on the current state of the system. The issue is how to setup the television display such that the viewer can control the television by simply pointing at the screen without being limited by the interface. The television screen needs to be setup such that most desired control commands can be done via a simple pointing gesture.

4.1.3 Multimedia data types synchronization

Different data types have different display constraints. For example, live video needs to be displayed at 30 frames per second, while text needs to be displayed for at least several seconds in order to read. Audio plays back at 44.1 KHz, when the video frames needs to be display at 13.5 MBytes/sec. [Ref 16] The system needs to be able to handle data types from many different sources and synchronize and display them on a common screen. The data types also includes the MJPEG objects from the Media Bank. [Ref 13, 14]

4.2 System Architecture

4.2.1 Schematic

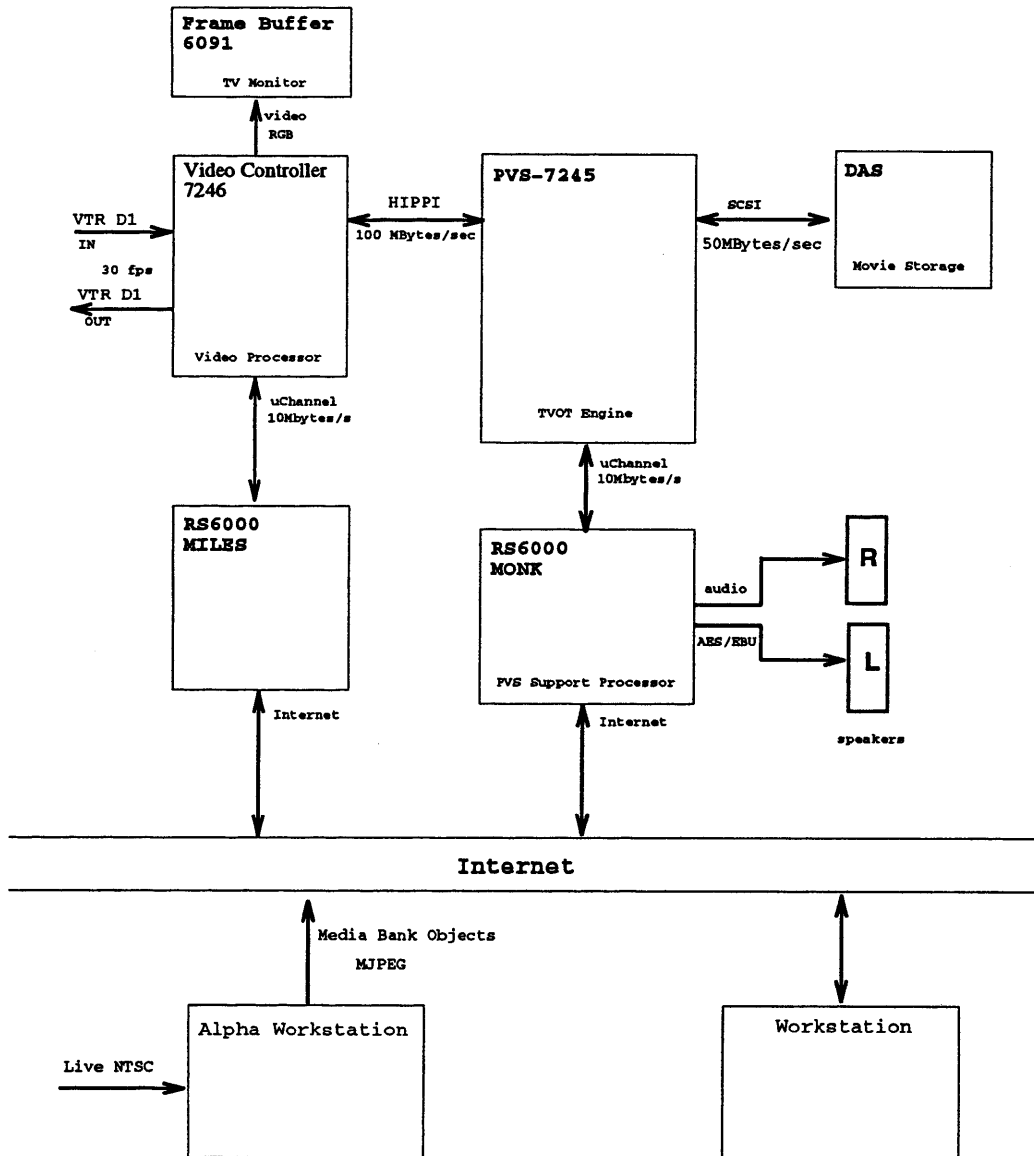


Figure 14: Context schematic diagram

Figure 14 presents the hardware schematic of the digital television prototype. The core system included two RISC workstations, the video controller and the PVS. The two RISC workstations are named MILES and MONK (see Figure 14). MONK is the front end for the PVS, and MILES controls the Video Control-

ler. MILES also serves as the user workstation. [Ref 11] The Video Controller controls the monitor display of the frame buffer. Programs are developed on MILES, and then loaded onto MONK which setups the PVS. Video is processed by the PVS and send to the Video Controller for display to the monitor. Audio data is parsed from the ancillary portion of the D1 [Ref 16] stream and then send to MONK for playback.

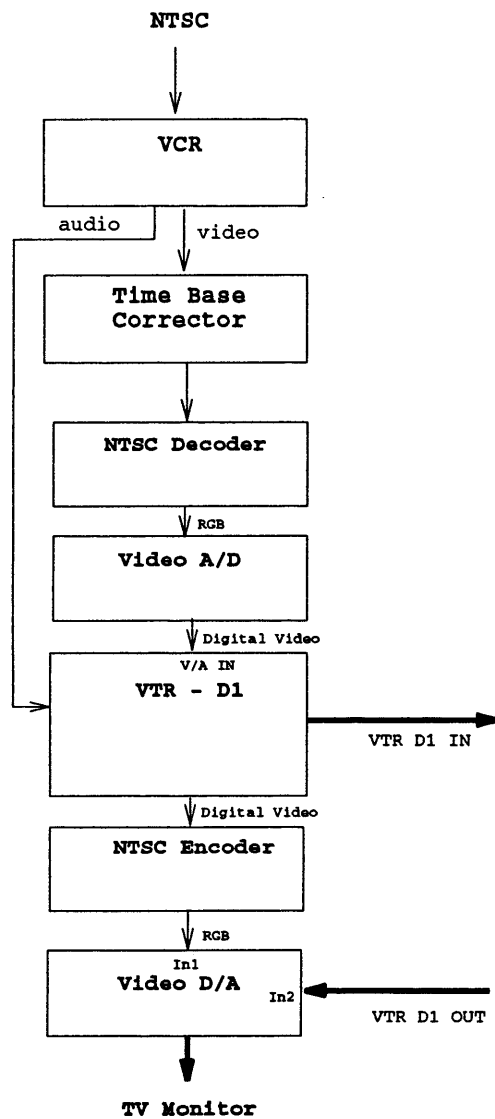


Figure 15:NTSC signal path

4.3 Data Paths & Types

4.3.1 Live Television

Live NTSC signal is transformed into its D1 (digitized and serial data stream) [Ref 16] equivalent in the system setup. (See Figure 15) The live D1 then enters the system via the serial input port of the video controller and traverse the 100 MBytes/sec HiPPI link into the PVS.

D1 video is then color space converted to ARGB format,, subsampled and cropped. The resulting image is then sent to the video controller for display at the television monitor.

Live audio is parsed from the D1 stream's ancillary data [Ref 16] at 48 kHz and placed in the shared memory with MONK. A software process executing on MONK then retrieves the audio data and play it back at the workstation's audio device (ACPA). [Ref 11, 17]

4.3.2 Movie Channel

Movie files are stored on the PVS's high speed disk array (DASD) in D1 format, and are streamed into the system via the 50 MBytes/sec SCSI link. The same process of transforming the audio and video then takes place as in the case for the live D1 signal.

Movie frames can be streamed into the system via 2 modes: 1) consecutive frames, and 2) skipped frame, only every other X movie frame is read and displayed. The skip frame count is setup the user.

4.3.3 Network

Network data bit-stream enters the system via the internet connections to MONK or MILES. Network data can be text, mjpeg and raw (rgb) video or combination of these formats.

4.4 User Interface Input - FISH

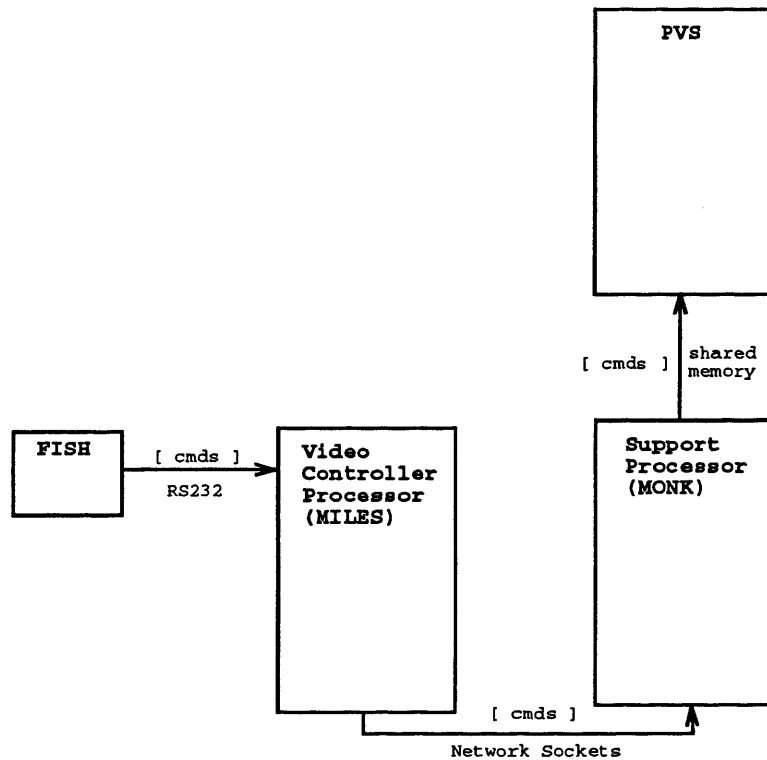


Figure 16: Fish Communication Setup

The FISH is connected to the RS232 port of the user workstation (see Figure 16). The FISH sensor readings consists of a four-byte packet. Each byte value represents the electric field reading between its transmitter and the receiver. The

electric field readings are altered as the user's hand interrupts the field. These field values are then used to determine the viewer's hand position. The readings are then sent across the internet using network socket commands to MONK. A software processing running on MILES is responsible for reading the sensor values and then sending them to MONK.

The FISH is tweaked for maximum sensitivity to sense the direction of the user's pointing direction. [Ref 12]

4.5 Software Architecture

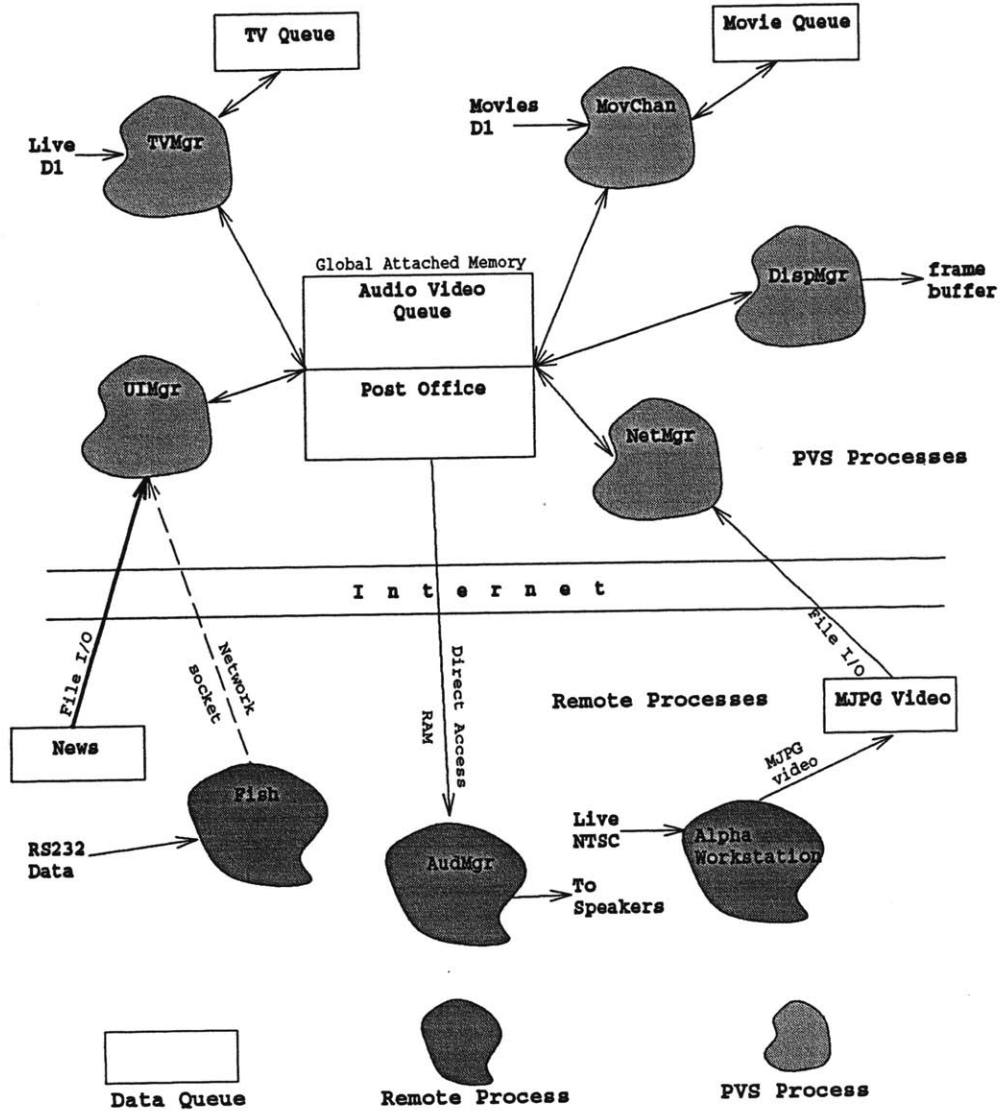


Figure 17: System Software Architecture

The software architecture of the digital television prototype is presented in Figure 17. The processes are divided into two categories: 1) PVS processes and 2) Remote processes. The remote processes execute on the workstations con-

ected via the internet. The functional description of each of processes are discussed in the section “Software Processes”, and the data queues in “Data Structures”.

The PVS software processes are:

UIMgr: sets up user interface to process user commands

TVMGR: sets up and process live television

MovChan: sets up and process the movie channels

DisMgr: synchronize and display all data streams

NetMgr: sets and process network data streams

The remote processes are:

FISH: reads the FISH and sends the sensor values to UIMgr

AudMgr: playback the audio at the workstation’s audio device

MJPEG_Video: send the compress video stream across network into system

The following sections discusses the design and the technical details of setting up the overall architecture, software processes and asynchronous data queues.

4.5.1 Distributed Processing

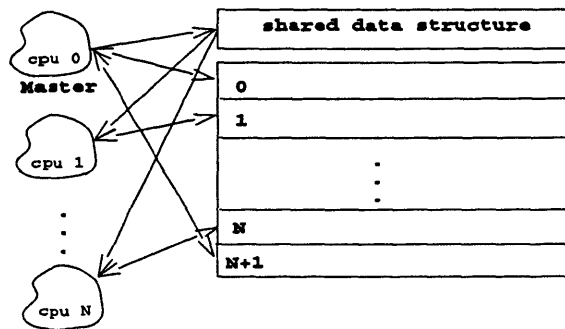


Figure 18: PVS Process Example

The Master Process, CPU #0 is defined at the OS level, sets up the data structures for the rest of the processors for fine grain execution

PVS processes are coarsely defined at the OS level, each with a group of processors allocated to it. The functionality of these processes are mutually exclusive. One processor in each process group is designated as the master and the rest slave processors. (See Figure 18) The master processor is responsible for performing communications and the initialization of each task prior to execution. Fine grained parallel execution of the task then takes place by all the processors in the group to achieve maximum throughput.

Figure 19 shows an example of how a PVS task is executed. Remote processes executes on their host workstations, and communicates with the PVS processes either through

- 1) network sockets,
- 2) network files, or
- 3) the 0.5 MBytes of direct access RAM on MONK.

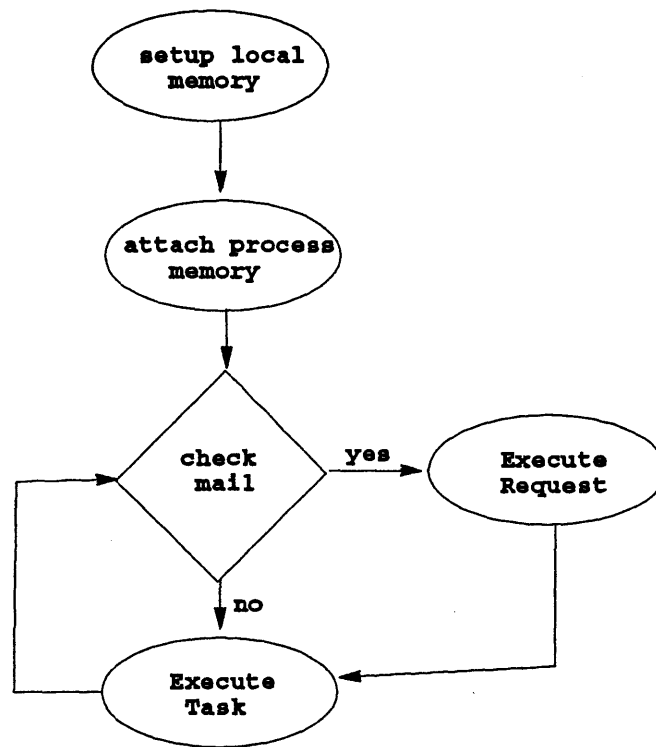


Figure 19: PVS Process Execution Sequence

4.5.2 Process Shared Memory

These memory segments are setup by the DispMgr, and all the other processes attaches to it during their initialization sequence. Process shared memory is the method used for communicating between all the PVS processes.

4.5.3 Process Communications

Process communications are categorized into 2 groups: 1) PVS - PVS processes, 2) PVS - network processes.

The PVS processors communicates using mailboxes. The mailboxes are setup in process shared memory. The post office containing all the of the mailboxes is in Figure 20.

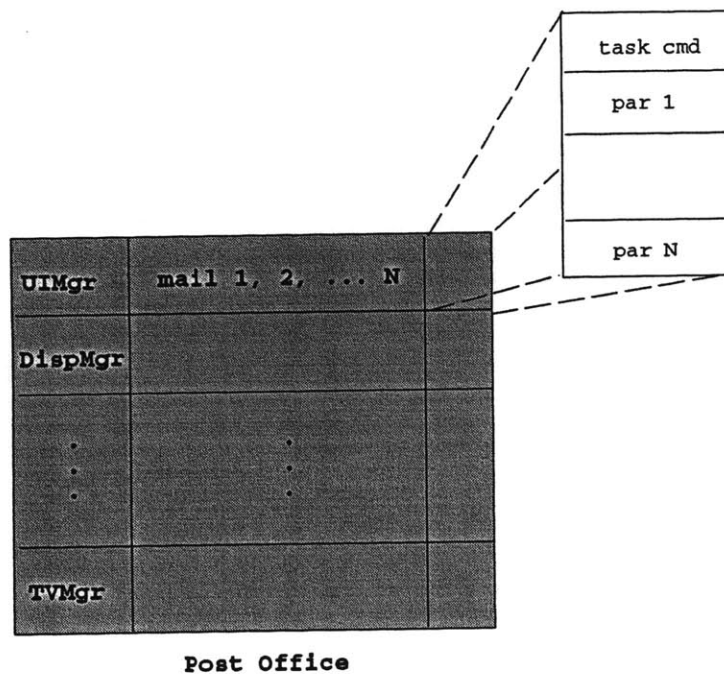


Figure 20:Mailboxes for process communication

Each process checks its own mailbox, and places the contents on its process task queue for execution. Sending mail consists of setting up the destination address and the task command for the destination process to execute.

4.6 User Commands

User commands enter the system in 2 ways: 1) keyboard input or 2) FISH sensing. The process UIMgr retrieves the keyboard inputs from the user and sends the task commands to the appropriate process for execution. FISH threshold values are read by the Video Support Controller Processor, and sent over to the UIMgr via network sockets. The UIMgr then sends the task commands to the appropriate process for execution. Appendix A contains the list of user commands implemented.

4.7 System Data Structures

4.7.1 Mailboxes

All the mailboxes are grouped together in the shared memory attached during initialization of each process (see Figure 20). Process can request another process to perform specified tasks by sending mail to the destination processes. Each process check its own mailbox once for each complete background pass. The format of the mail messages is shown in Figure 21.

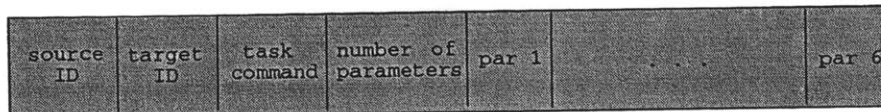


Figure 21:Mailbox Data Structure

4.7.2 Real-time Queues

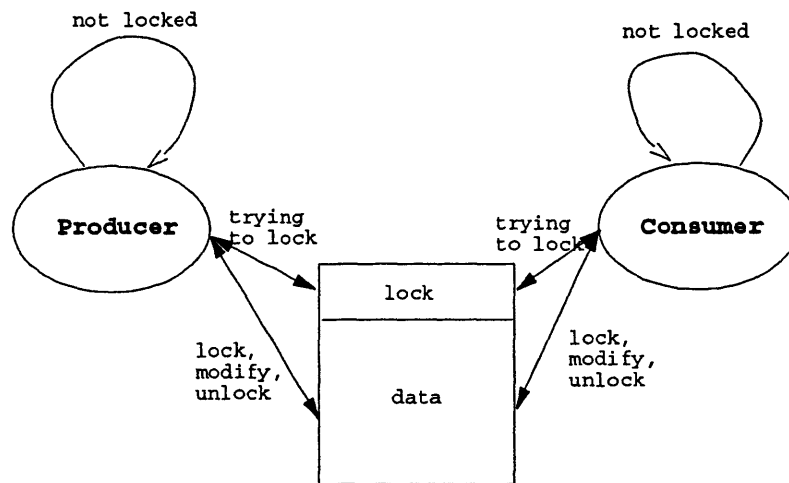


Figure 22:Real time queue access strategy

The real-time queues contains time critical data elements. The size of the data queues are kept small (four) for a) rapid system crash recovery due to overflow or underflow and b) memory conservation. These queues are accessed in a “producer and consumer” manner shown in Figure 22.

Each queue contains: 1) Number of elements currently in the queue, 2) lock structure, locking out concurrent processes 3) the get and put pointers which are used to insert or remove elements from the queue.

The process first tries to “lock” the queue to prevent other process from modifying queue contents. After the lockup, the process modifies the Number of Elements in the queue to reflect the status of the queue. The process then unlocks the queue releasing queue for other processes to access. If the process failed to lock onto the queue, the process waits until it can lock the queue before proceeding. The system’s real-time queues are summarized in Table 3.

Table 3: Real Time Queues summary

Name	Memory Type	Process	Data Type	Description
Video Queue	PVS Shared	DispMgr (PVS)	video ARGB	Screen display memory
TV Queue	PVS Shared	TVMgr (PVS)	video ARGB	Live NTSC, color space converted to D1 then to ARGB for display
Movie Queue	PVS Shared	MovChan (PVS)	video ARGB	Movies stored in D1 format, converted to ARGB for display
Network Queue	PVS Shared	NetMgr (PVS)	MJPEG	MJPEG movie file converted to ARGB for display
Text Space Queue	PVS Shared	UIMgr (PVS)	text	text are rendered then displayed anywhere on the display screen
Image Space Queue	PVS Shared	UIMgr (PVS)	pbm	image file are converted to pbm format, raw RGB and then displayed
News Space Queue	PVS Shared	UIMgr (PVS)	text	News text are rendered, and then displayed inside the news space box
RS6000 Audio Queue	RS6000 Shared	AudMgr (RS6000)	audio PCM	RS6000 process fetches the audio data and sends it to ACPA card for audio playback

4.8 Software Processes

There are 3 types of software processes: 1) PVS, 2) RS6000 and 3) remote UNIX process. Table 4 contains a summary of all the software processes for the prototype.

Table 4: Summary of software processes

Name	Type	System Resources	Description
DispMgr	PVS	shared memory: 64M (32-bit) 12 processors	Read video and audio data from all the realtime queues, and update the monitor display
TVMgr	PVS	shared memory: 64M (32-bit) 12 processors	Take the live D1, perform color space conversion image subsampling if required and write the video and audio outputs to perspective data queues
MovChan	PVS	shared memory: 64M (32-bit) 5 processors	Stream in the D1 streams from each movie, color space convert it to ARGB format, subsample image, perform desired fading alg and write to the video and audio data queues
AudMgr	RS6000	PVS Direct Access Mem: 0.5M (32-bit)	Takes the audio PCM data from the PVS's Direct Access Memory, and playback the data at the workstation's ACPA audio device
UIMgr	PVS	shared memory: 64M (32-bit) 1 processor	Setup user menu for selection, take user commands from the keyboard, and sends cmds to all the other processes to adjust system behavior, communicate with FISH processor for adjusting system behavior
NetMgr	PVS	shared memory: 64M (32-bit) 1 processor	Take a MJPEG compressed file on the network, performs MJPEG decompression, image subsampling if required and write the video out to the video queue for display
FISH	RS6000 or UNIX	any UNIX process	Reads the FISH sensor value from its RS232 port, and then use network socket and transmit the value to UIMgr for appropriate action

4.8.1 PVS

The number of processors, and global memory is allocated before each process is executed. CPU #0 is designated the Master processor. The Master processor handles the inter-process communications, perform semaphore updates, and initialize global data for the slave processors to execute. Each process is coarsely defined at the OS level. Fine-grained parallel execution then takes place after the Master processor have completed the global memory updates. All slave processing are terminated and program control returned to the Master after the fine-grained processing task is completed.

4.8.2 RS6000

This process (AudMgr) executes on MONK. This process currently only process the audio data for playback at the workstation's ACPA audio device.

4.8.3 Remote UNIX processes

This process (FISH) currently executes on the PVS's Video Controller Processor, (MILES) however this can be an UNIX process executing on any workstation connected to the internet. This process reads the threshold reading from the FISH sensor and sends it across the network using socket commands. The PVS process (UIMgr) on the other end of the network retrieves the threshold readings and adjusts the behavior of the system.

Chapter 5

Results

This chapter presents the completed prototype and television interface experiment results. Three types of data sources are used: 1) live television, 2) stored movies, and 3) network data.

The prototype provided the massive computing resources needed for the user interface to manage these data streams in real time. The prototype can handle a maximum of 12 movie streams, one live television channel, one mjpeg formatted movie file, and one text file. The individual data streams are in their native formats, and are manipulated as they enter the system. All the user interface experiments are conducted with live data streams. The movie files are currently stored on the high speed disk array connected to the system. Network mjpeg compressed movie files are used as the alternative movie source for the system. The text files are read and their corresponding fonts are rendered for the television screen to display.

User interface results are captured via screen dumps. The results are then compared for user acceptance, motion elimination and computing requirements.

Section 5.1 presents the experimental results.

Section 5.2 discusses parallel process designs.

5.1 User Interface

The following sections present the experimental results.

The different motion elimination techniques, screen layout, and television control device are evaluated for the television application. The evaluation criteria are:

- 1) the technique's ability to reduce or eliminate motion.
- 2) computing requirement.
- 3) content grasp and navigability of the screen
- 4) user acceptance to use the interface for watching television.

Appendix A lists the system commands implemented.

Appendix B lists the mail message commands implemented.

Appendix C contains the screen captured during a system batch run.

5.1.1 Motion Elimination

The objective of these experiments is to reduce the motion of the channel contents in order to minimize distraction it causes. Three types of algorithms are evaluated: 1) cross fade, 2) line scans, 3) and video streaming.

These techniques are compared against one another for their ability to eliminate motion and the results are summarized in Table 5.

5.1.1.1 Cross Fade

An example of the channel cross fade sequence is shown in Figure 9. The cross fade technique provides a smooth transition between the frames, and has proved to be successful in reducing the motion between updates. The main problem is that this technique is computing intensive. Every single frame display time needs to have an image to display in order to provide a smooth transition. Each of these images must be computed by using the weight factor based on the time position between the two end frames. This technique is an excellent selection for future systems with a greater and cheaper computing bandwidth.

5.1.1.2 Line Scans

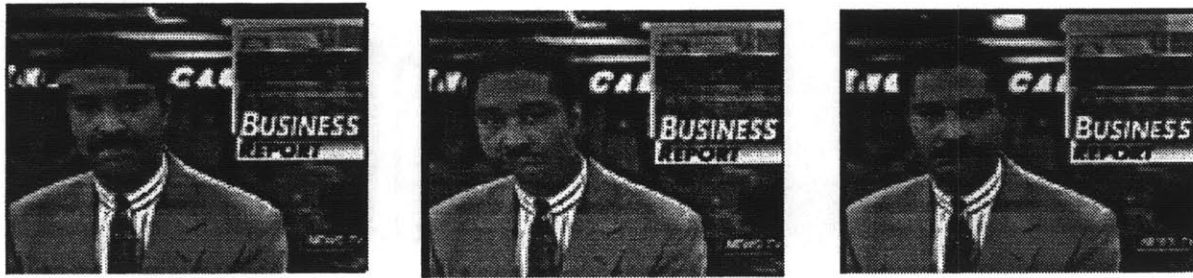
Figure 23 presents the line scanning algorithms evaluated. The surprising results suggest the area on an image where motion is most likely to occur.

Figure 23 (a) is the sequence captured for the “up-down” scan. This tech-

nique is the most efficient of the three under consideration. This is due to the way the computer addresses adjacent connected memory locations in an image. The images are updated so fast that motion between frames causes an abrupt jump. Although simple and efficient, this technique fails to reduce the motion of the side channels.

Figure 23 (b) is the sequence captured for the “window open-close” scan. The results reveal that image motion is usually at the center of the image. This is supported by the fact that when we take pictures, the center of the image is typically focused on the object of interest. The double scan is not as efficient as the “up down scan”, however it is efficient enough so that jumps can be detected between frame updates.

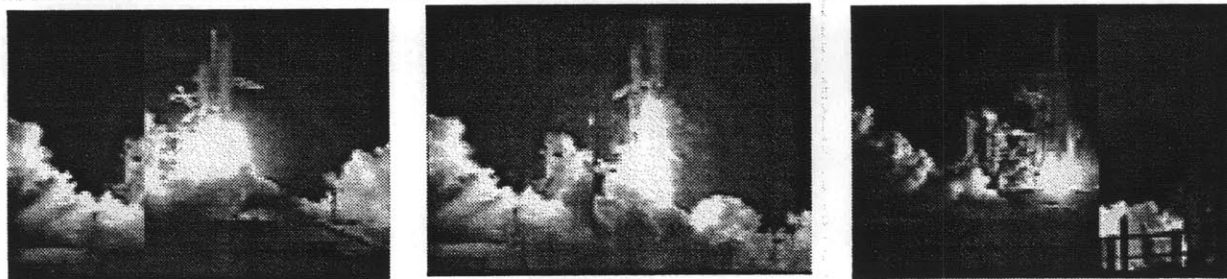
Figure 23 (c) is the sequence captured for the “left-right” scan. This technique is not as efficient as are the previous two because individual pixels in a vertical line must be addressed for each update. The inefficiencies cause the routine to be executed slowly giving the effect of a slow radar sweep between frame updates. This update is smooth and hides the motions well. The computing requirement is not unreasonable and is the recommended technique for the television application.



(a) Up - Down Horizontal Line Scan



(b) Window Open-Close Vertical Line Scan



(c) Left-Right Vertical Line Scan

Figure 23: Image Line Scan Algorithms

Table 5: Motion Elimination Results Summary

Alg Name	Rating			Comments
	Motion Hiding	computing requirement	smoothness	
Left Scan Right Scan Left-Right Scan	G	E	E	For currently available system this algorithm is the best compromise
Up Scan Down Scan Up - Down Scan	F	E	F	Need to be significantly slow down for this to be effective, too much additional work needs to be done
Window Open Window Close Window Open & Close	G	G	G	Window Closing hides motion better then widow open. Not as smooth as left-right scan.
Cross Fade	E	P	E	Recommended for future more powerful systems.
Video Streaming	F	F	F	Not recommended

Rating: (P)oor, (F)air, (G)ood, (E)xcellent

Television of Tomorrow



TVChannel



Movie Channels

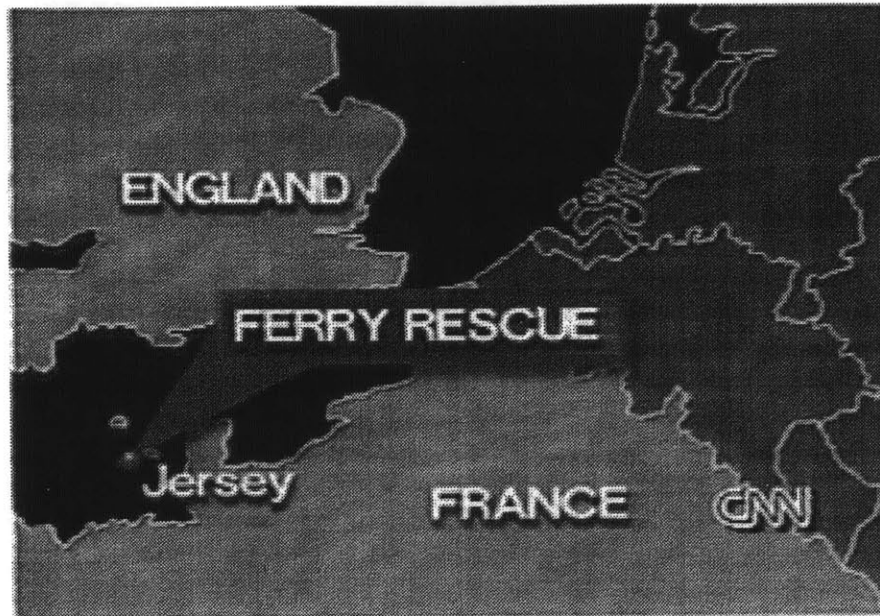
News Space

authorities that should take the blame for

MIT Media Laboratory

Figure 24: 16:9 Screen Organization

Television of Tomorrow



Network



TV Channel
News Space



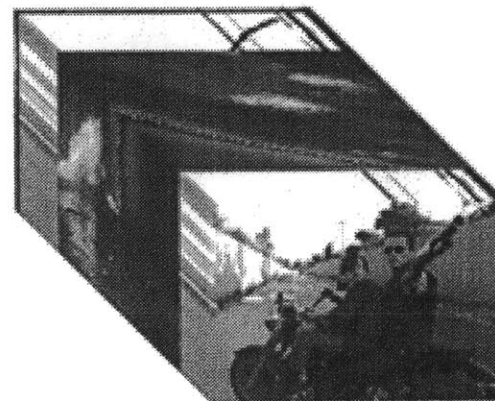
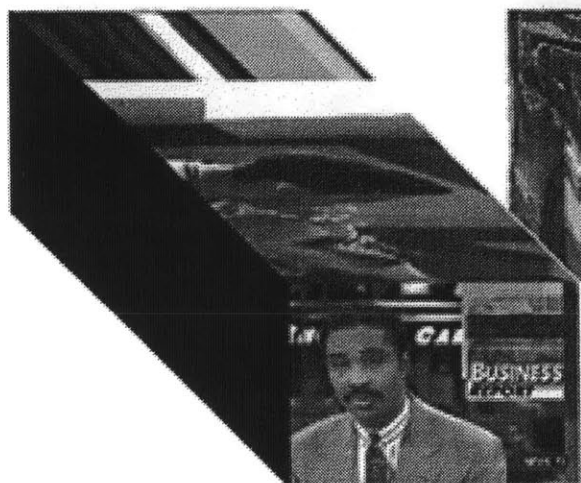
Movie Channels

(1993-1994) should have cleaned up this affair

MIT Media Laboratory

Figure 25: Pyramid Setup

Television of Tomorrow



TV Channel
News Space

Movie Channels

Television Video Streamer

Figure 26: Video Streamer

MIT Media Laboratory

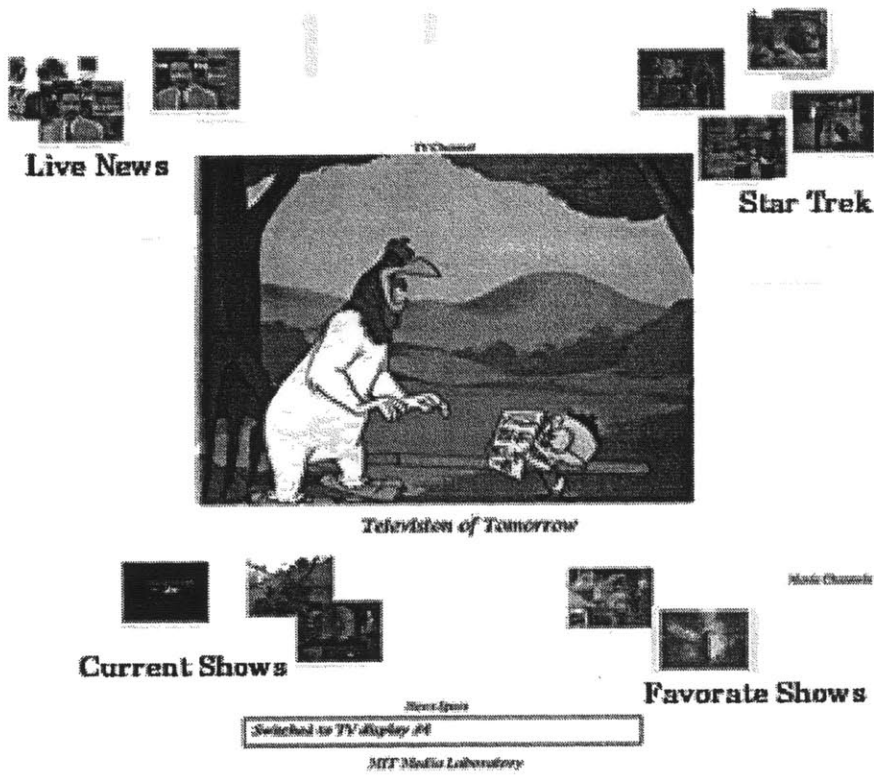


Figure 27: Content Grouping

5.1.2 Screen Organization

The screen organization experiments are presented in Figures 24 - 27. Four experiments have been conducted:

- 1) one with an almost 16:9 ratio similar to that of HDTV format.
- 2) array setup with one main channel and 3 smaller side channels.
- 3) video streamer
- 4) channel content grouping

The channel size is used to visually separate the main channel from the side channels. The side channels can display a different story perspective to support the main story or an entirely different story.

The experiments show that one main channel coupled with 3 smaller side channel to be an acceptable number of the channels for viewing. With this screen organization and number of channels, and with proper motion reduction techniques on the side channels, the viewer can watch all the channels simultaneously without distraction and confusion.

5.1.3 Proximity Sensor

The FISH sensor has been integrated into the system and explored as an alternative television control interface. The goal is to allow the user to issue control commands by simply pointing to the area of interest on the television monitor. The FISH is attached to MILES, the user workstation which accepts the user pointing commands. The sensor have three transmitter and one receiver. The relative positioning of the user's hand is measured by the strength of the transmitter

signal to the receiver. Three distinct areas on the television monitor are drawn for the sphere of influence for each transmitter. The user points to an area on the monitor, the transmitter threshold readings are read and sent over to the FISH process executing on the user workstations. The process determines the position of the finger pointing area from the threshold readings and sends the result over to UIMgr via network socket commands. UIMgr then determines the appropriate menu action based on the current state of the system. The actions are typically either to speedup or slow down one of the side channels, or switch the side channel to become the main channel.

Experiments with the FISH sensor have failed due to its inconsistencies and lack of gesture understanding capabilities. The inconsistencies are due to the lack of additional flow control hardware to ensure proper serial communication with the user workstation.

The lack of gesture understanding capability proves to be the main issue. More intelligence must be incorporated into the device in order to decipher the complex meaning behind a simple user pointing gesture.

5.2 Prototype Design

The key to parallel programming is the task division at the operating system level. Real time task queues in process shared memory have successfully relayed the results of one process to another to achieve the overall system objective. Duplicating each field and each pixel during color space conversion has greatly improved system performance. The following lists the design decisions made to achieve desired system performance:

1. Color space conversion. The i860 chip local RAM lookup table, 8-bit Y, 7-bit Cr, 6-bit Cb, sacrificed image resolution and internal RAM to accomplish fast table lookup. Image resolution is acceptable due to the already low resolution of NTSC video signal.
2. Image processing. Using only field 1, duplicating field 1 for every other scan line. Each adjacent pixel is duplicate to reduce number of table lookup. This created a 2x2 blocking effect of the image, however to the low resolution of the NTSC signal, and image motion, the side effect does not create significant problems for application.
3. Image processing. Vertical and horizontal subsampling of the image is integrated into color space conversion. This saves the unnecessary processing required of the large original image, and reduces the size of data at the earliest possible time.
4. Parallel Computing. Distributed processing using process attached shared memory. The processes use an internal shared memory mailbox system to communicate and synchronize.
5. Distributed processing. Real time data queues are used to synchronize and relay data from process to process to achieve system objectives.
6. Distributed processing. Applications are farmed out to the surrounding workstations for processing, and the results are sent back and applied into main application.

Audio video synchronization is achieved by keeping the two data streams together in D1 format and keeping the two playback queues as small as possible. The small queue sizes prevent the two data streams from getting too far off sync from each other. Throwing away audio data instead of video data to synchronize the 2 data streams has proved to be more tolerable.

Chapter 6

Conclusions

The work here has shown that a digital television is realized by harnessing the massive computing power of a parallel super computer. This work have also shown that the success of digital television depends heavily on the user acceptance of its interface. Future research should proceed in the direction of using the power of the computer to make the television interface as simple and easy to use as possible. This can be done by merging the user interface and parallel computing technologies.

6.1 User Interface

This area should concentrate on using the computer to assist in television viewing instead of using it as a simple display controller. This can be done by using intelligent agent software. These agents tracks the viewing habits and tries to anticipate user behavior. Different story perspectives are automatically gathered and displayed without user intervention. This avoids the tedious and time consuming setup procedures making the system simple and easy to use.

6.2 Parallel Computing

This area should focus on making the parallel software more robust so that modification and maintenance can be done easily. Message passing between processes have shown great promise for achieving this goal.

Appendix A User Commands

This section presents the user commands implemented for the system.

%%	[string]	;comment statement, no action
AskDrawBox	[ltx, lty, rbx, rby, backgrd]	;draw box with backgrd color
AudOff		;turn off system audio
AudOn		;turn on system audio
ChnReset	[channel #]	;reset movie channel
CropTV		;crop channel image area
DecAud		;decrement system audio volume
FishDemo		;FISH demos
HighLight	[channel #, backgrd]	;highlight backgrd of channel
ImgSpace	[file, x size, y size, xloc, yloc]	;plot file to image space
IncAud		;increment system audio volume
IntNewsDisplay		;interactive text write to news area
Modsize	[channel #, x size, y size]	;modify size of channel
ModCoord	[channel #, x, y]	;modify channel coordinates
ModBgrd	[channel #, backgrd color]	;modify channel
ModFade	[channel #, alg]	;modify channel fade alg
ModScrn	[channel #, screen #]	;modify system display screen
NewsDisplay		;write text to news display area
PrAddr		;display address of the real time queues
PrDbg		;print system debug information
PrMovChn	[channel #]	;print content of movie channels
PrTVChn	[channel #]	;print info on TV channel
PrQStat		;display real time queue info
PrSize	[channel #]	;print the size of the audio queue
pr_tv_setup		;print the tv setup
pr_fade_struct		;display fade struct for all channels
ShotDown		;system shutdown
SnapShot		;screen dump and store to file
StartTV		;start the TV
SwrchChn	[channel #]	;switch main channel
SysII		;system illustration mode
TextspaceEX		;write to text space
TVReset		;TV system reset
Wait	[seconds]	;pause program from # seconds

Appendix B Inter-Process Mail Messages

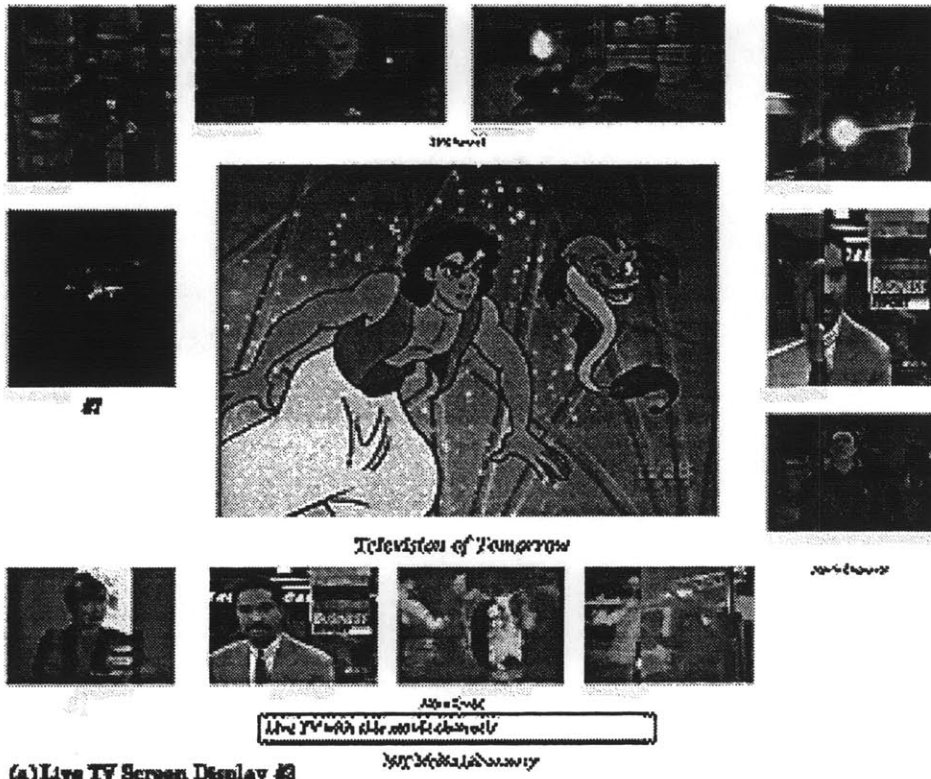
This appendix contains the inter-process communication messages implemented for the system.

<u>Message</u>	<u>Module Name</u>	<u>Description</u>
C_MOV_MEM_INIT	movie_memory_init()	;movie memory init
C_ST_MEM_INIT	stream_memory_init()	;stream memory init
C_MOV_ST_INIT	init_movie_streams()	;initialize movie streams
C_INIT_MOV_BUFF	init_movie_buff()	;initialize movie buffers
C_OPEN_MOVIES	open_movies()	;open the movie streams
C_FILL_MOV_ST	fill_movie_streams()	;fill the movie streams
C_ATTACH_SHM	attach_process_shm()	;attach process shared mem
C_INIT_AV_PTRS_SHM	init_shm_av_ptrs()	;init av ptrs in shared mem
C_INIT_PROC_MBOX	init_seg_mailbox()	;initialize process mailbox
C_PROC_MOVIE	process_movies()	;process movies
C_SWITCH_MAIN	switch_main_channel()	;switch main channel
C_CHN_DISP_NORMAL	scrn_disp_normal()	;normal display
C_CHN_DISP_FADE1	scrn_disp_fade1()	;fade 1 display
C_CHN_DISP_FADE2	scrn_disp_fade2()	;fade 2 display
C_CHN_DISP_FADE3	scrn_disp_fade3()	;fade 3 display
C_MOD_CHN_PARM	modify_chn_param()	;change channel parameters
C_MOD_CHN_BACKGRD	modify_chn_backgrd()	;change channel backgrd color
C_PR_CHN_STAT	pr_movchn_stat()	;print channel status
C_PR_TV_STAT	pr_tv_stat()	;print tv status
C_CROP_CHN_DISP	crop_tvchn_disp()	;crop channel display
C_MOD_CHN_COORD	mod_chn_coord()	;modify channel coordinates
C_TV_SCRN_RESET	reset_tv_scrn()	;tv screen reset
C_TV_SCRN_SET	set_tv_scrn()	;set the TV background scrn
C_CHN_SCRN_RESET	reset_chn_scrn()	;channel display reset
C_PR_DEBUG	pr_debug_info()	;print debug info
C_ADD_LIVE_CHN	add_live_chn()	;add live tv channel
C_DEL_LIVE_CHN	del_live_chn()	;delete live tv channel
C_ADD_MOVIE_CHN	add_movchn()	;add movie channel
C_DEL_MOVIE_CHN	del_movchn()	;delete movie channel
C_SYS_SHUTDOWN	sys_shutdown()	;system shutdown
C_SYS_RESET	sys_reset()	;system reset
C_INC_VOL	inc_vol()	;increase audio volume
C_DEC_VOL	dec_vol()	;decrease audio volume
C_CHN_AUDIO_ON	trn_audio_on()	;turn channel audio on
C_CHN_AUDIO_OFF	turn_audio_off()	;turn channel audio off
C_PR_AUDIO_STAT	pr_audio_stat()	;print audio device status

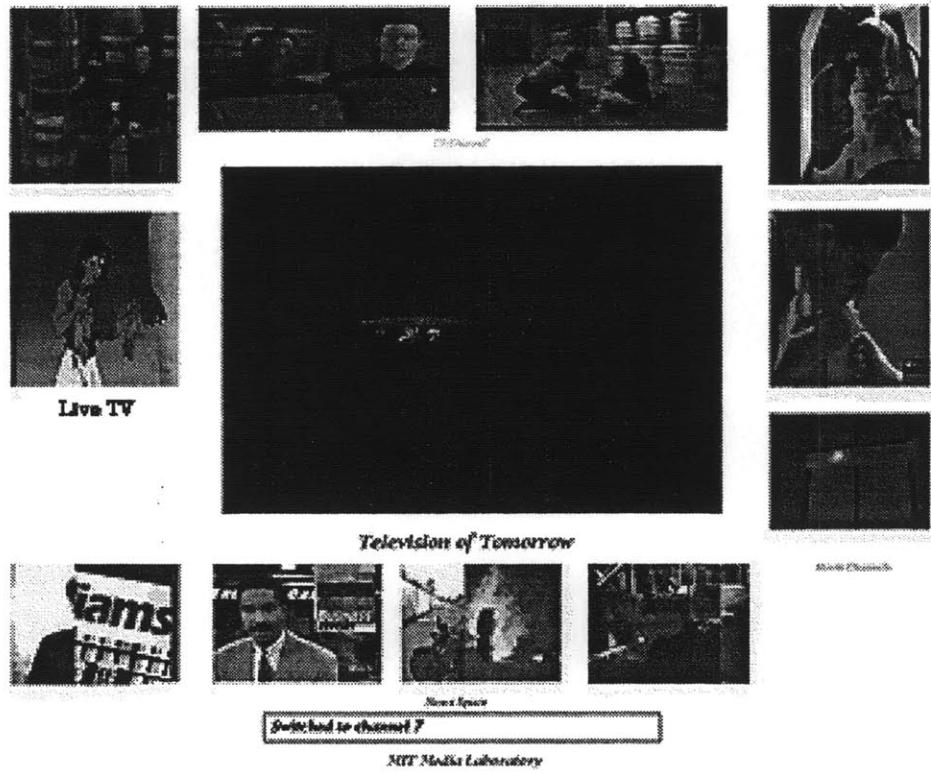
<u>Message</u>	<u>Module Name</u>	<u>Description</u>
C_EST_PROC_COMM	establish_proc_comm()	;establish comm with process
C_PR_PROC_STAT	pr_proc_stat()	;print process status
C_SAVE_SYS_STATUS	sav_sys_stat()	;save system status
C_CHECK_MAIL	check_mail()	;check incoming mail
C_SEND_MAIL	send_mail()	;send mai
C_READ_MAIL	read_mail()	;read mail
C_ETASK	etask()	;execute task
C_MESSANGER		
C_MOD_FADE_ALG	mod_chn_fade()	;modify current fade alg
C_START_TV	startup_tv()	
C_FLUSH_VQUE	flush_VQue()	;flush video queue
C_FLUSH_AQUE	flush_AQue()	
C_DRAW_BOXNP	draw_boxnp()	
C_DRAW_BOX	draw_box()	
C_ROVING_MODE	roving_mode()	

Appendix C System Batch Run

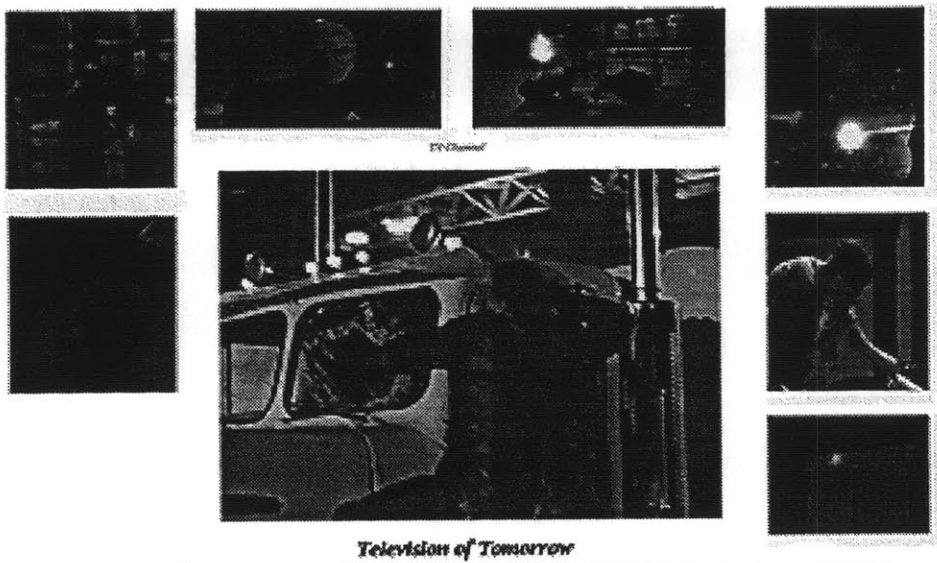
This appendix contains the screen dumped captured during a system batch run. The batch commands are shown in appendix A.



(a) Live TV Screen Display #3



(b) Switched Main to Channel #7



Television of Tomorrow



Street Space

Live TV

Switched to channel 12

(c) Switch Main to Channel #12

MIT Media Laboratory

Television of Tomorrow



Network

Street Space

Media Channel

TV Display Screen #2

MIT Media Laboratory

(d) Switch to TV Screen Display #2

Television of Tomorrow



#2



Network

TV Channel
News Spot

Movie Channel

TV Display Screen #2

(a) Screen Display #2

MIT Media Laboratory

Television of Tomorrow



Network

TV Channel
News Spot

Movie Channel

Switched to Movie Channel #2

(c) Switch Main to Channel #2

MIT Media Laboratory

Television of Tomorrow



TV Channel



Movie Channel

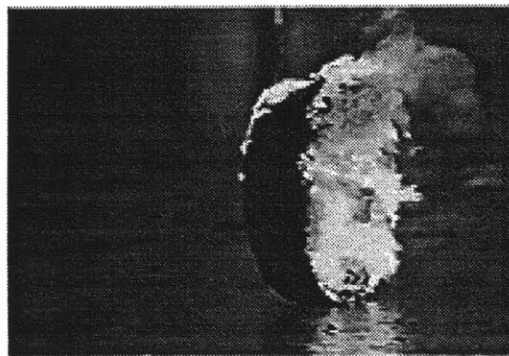
News Space

Switched to TV display #1

JIT Media Laboratory

(g) Switch to Screen Display #1

Television of Tomorrow



TV Channel



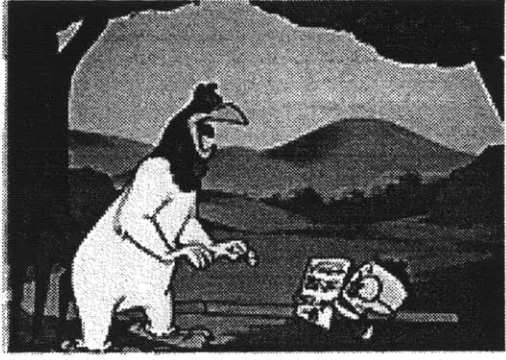
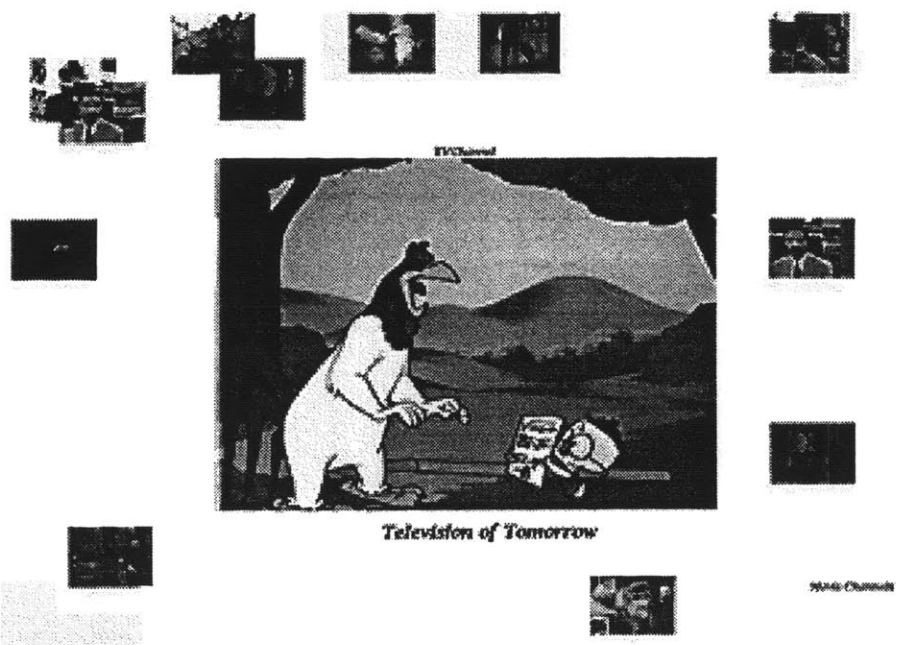
Movie Channel

News Space

Switch to Channel #4

JIT Media Laboratory

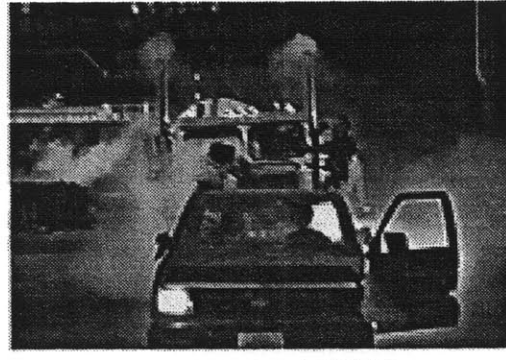
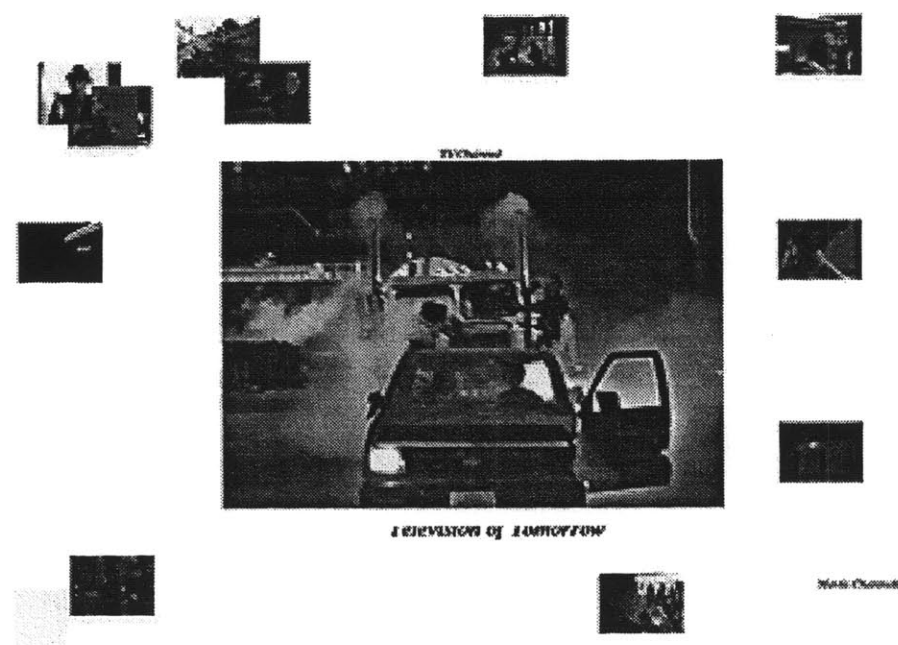
(h) Switch Main to Channel #4



Television of Tomorrow

(i) Switch to Screen Display #4

Screen Space
 Switched to TV display #4
 MIT Media Laboratory



Television of Tomorrow

(j) Switch Main to Channel #12

Screen Space
 Switched to channel #12
 MIT Media Laboratory

References

- [1] Kramer, "*Translucent Patches - Dissolving Windows*", UIST '94, pp.121-130
- [2] Bier, Stone, Fishkin, Buxton, Baudel, "*A Taxonomy of See-Through Tools*", CHI '94, pp.358-364
- [3] Elrod, Bruce, Gold, Goldberg, Halasz, Janssen, Lee, McCall, Pedersen, Pier, Tang, Welch, "*LiveBoard: A large interactive display supporting group meeting, presentations, and remote collaboration*", CHI '92, pp.599-607
- [4] Lieberman, "*Powers of Ten Thousand: Navigating in Large Information Spaces*", UIST '94, pp.15-16
- [5] Griswold, Jeffery, "*Nova: Low-Cost Data Animation Using a Radar-Sweep Metaphor*", UIST '94 pp. 131-132
- [6] Stone, Fishkin, Bier, "*The Movable Filter as a User Interface Tool*", CHI '94, pp.306-312
- [7] Davis, Marc; *Media Streams: An Iconic Visual Language for Video Annotation*, Proceedings of 1993 IEEE Symposium on Visual Languages in Bergen, Norway, IEEE Computer Society Press, 196-202, 1993
- [8] Sarkar, Brown, "*Graphical Fisheye Views of Graphs*", CHI '92, pp. 83-91
- [9] Davis, Andrew W, *Motion Image Processing: Striking Possibilities*, Advanced Imaging, August 1992, pp. 22-25
- [10] Furnas, Zacks, "*Multitrees: Enriching and Resuing Hierarchical Structure*", pp. 330-336, CHI '94
- [11] IBM Power Visualization System: *Programmer's Technical Reference*, First Edition, Doc #: SC38-0487-00, IBM Corporation, Thomas J. Watson Research Center/Hawthorne, PO Box 704, Yorktown Heights, NY 10598
- [12] Physics and Media Group; *The "FISH" Quad Hand Sensor: Users Guide*, MIT Media Laboratory, 20 Ames St, E15-022, Cambridge, MA 02139-4307
- [13] Lippman, A.B: *The Distributed ATM Media Bank*, MIT Media Laboratory, July 1994
- [14] Lippman, A.B; Holtzman: *The Distributed Media Bank Specification*, MIT Media Laboratory, May 19, 1994
- [15] Hwang, Kai, "*Advanced Computer Architecture: Parallelism, Scalability, Programmability*", McGraw-Hill Inc, 1993
- [16] D1 spec - CCIR-601 standard
- [17] *M-ACPA Programmer's Guide and Reference, AIX Version 3.2*, International Business Machines Corporation, 1993. Doc #: SC23-2584-00