ABSTRACT

Title of Thesis: A Mouse-Driven Interface for Androx ICS-400

Name: Chin-Tien Chen
Master of Science in Electrical Engineering, 1991
Department of Electrical and Computer Engineering

Thesis directed by: Dr. Edwin S. H. Hou

A mouse-driven interface is extremely useful for software performing image processing tasks. This thesis describes an implementation of a mouse-driven interface for the Androx ICS-400 image processing board. The Androx board is housed in a Sun 4/330 running a UNIX-based operating system. The main goal of this interface is to provide a consistent and friendly user interface for image processing tasks and integrating the Sun window libraries with the Androx libraries. Various image processing tasks, such as gray scale and binary morphological operations, histogram, thresholding, filtering, spot, logical operations, arithmetic operations, zooming, scrolling, rank and image acquisition are implemented in this interface. An on-line help is also available.

A Mouse-Driven Interface for Androx ICS-400

by

Chin-Tien Chen

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

1991

APPROVAL SHEET

Title of Thesis: A Mouse-Driven Interface for Androx ICS-400

Candidate: Chin-Tien Chen
Master of Science in Electrical Engineering, 1991

Thesis and Abstract Approved by the Examining Committee:

Dr. Edwin S. H. Hou, Advisor Assistant Professor Department of Electrical and Computer Engineerig	Date
Dr. Nirwan Ansari Assistant Professor Department of Electrical and Computer Engineerig	Date
Dr. Anthony D. Robbi Associate Professor Department of Electrical and Computer Engineerig	Date

New Jersey Institute of Technology, Newark, New Jersey.

VITA

Chin-Tien Chen

Date of Birth:

Place of Birth:

Education:

1990-1991	**New Jersey Institute of Technology**	MSEE
1978-1981	**National Taipei Institute of Technology**	BSEE

Position Held:

Taiwan International Standard Electronics Ltd.

Electronic Engineer

1983-1985

Digital Equipment Corporation Taiwan Ltd.

Project Engineer

1986-1987

GTE-TAICOM Systems Limited

Hardware, R & D

Project Engineer

1988-1989

ACKNOWLEDGMENTS

The author would like to express appreciation to Dr. Edwin S. H. Hou, his major professor, for his inspirations, guidances and support during the entire course of this thesis. The author also wishes to express his indebtedness to Dr. Nirwan Ansari for his support in the image processing software. Appreciation is also extended to the committee member, Dr. Anthony D. Robbi, for serving on the examination committee and for evaluating this research. Finally, special thanks go to Miss Heui-Chu Lin, his fiancee, for her help and encouragement.

Contents

1 Introduction **1**

1.1 SunView Interface 1

1.2 Androx Interface 2

1.3 Interface Implementation 2

1.4 Operational Instructions 2

2 SunView Interface **4**

2.1 Windows 4

2.1.1 Frames and Subwindows 5

2.1.2 Panels 5

2.1.3 Windows Generating and Destroying Procedure 6

2.1.4 Manipulating Frames via Menus 7

2.1.5 Window Pop-ups 7

2.2 Menus 7

2.2.1 Menu Usage 7

2.2.2 Components of Menus Items 8

2.2.3 Menus Generate Procedure 8

2.3 Events 8

2.3.1 Notify Procedure 9

2.3.2 Events Generation 9

2.3.3 Event Handling 9

2.3.4 Classes of Events 10

2.3.5 Enabling and Disabling Events 10

2.4 SunView Libraries 12

3 Androx Interface **13**

3.1 System Structure 13

3.2 Memory Configurations 15

3.3 Event Scheduling 16

3.3.1 PEV(event) 16

3.3.2 Wait Event 17

3.3.3 Create Event 18

3.4 Androx Libraries 18

3.4.1 Applications Library 18

3.4.2 Graphics Library 19

3.4.3 Image Processing Library 19

3.4.4 Video Library 20

4 Interface Implementation **21**

4.1 SunView 21

4.1.1 Menus and Selection Structure 21

4.1.2 Event Structure 24

4.1.3 Subwindow Popup Structure 27

4.2 Androx 28

4.2.1 Accessing and Releasing System 28

4.2.2 Image Processing Functions 29

4.3 Passing Parameter 39

4.3.1 Variables 40

4.3.2 Subroutine Call 41

4.4 Software Compile and Link 42

5 Operational Instructions **44**

5.1 Load RGB File 48

5.2 Save RGB File 48

5.3 Histogram 48

5.4 Linear Operations 49

5.4.1 Convolution 49

5.4.2 Laplacian 49

5.4.3 Median 49

5.4.4 Sobel 49

5.4.5 Gaussian 50

5.5 Nonlinear Operations 50

5.5.1 Morphological 50

5.5.2 Rank 52

5.5.3 Spot 53

5.5.4 Reset 53

5.6 Logical Operations 53

5.6.1 AND Image 53

5.6.2 OR Image 53

5.6.3 XOR Image 54

5.7 Arithmetic Operations 54

5.7.1 Add Image 54

5.7.2 Sub Image 54

5.7.3 Sum Image 54

5.8 Acquisition 55

5.8.1 Digitize Mono 55

5.8.2 Digitize RGB 55

5.8.3 Data Move 55

5.9 Display Bank Image 55

5.10 Others 56

5.10.1 Thresh 56

5.10.2 Lookup Table 56

5.10.3 Pseudo Color 56

5.10.4 Scroll 56

5.10.5 Zoom 57

5.10.6 Blank 57

5.11 Help 57

5.12 Quit Help 57

5.13 Quit Program 57

6 Conclusions **58**

Bibliography **59**

List of Figures

2.1 The events process diagram 11

3.1 The Androx system structure 14

3.2 The memory mapping of ICS-400XM9 16

4.1 Hierarchical menu structure 23

5.1 The menu items of mouse-driven interface 45

5.2 The menu items of mouse-driven interface (continued) 46

5.3 An example of menu item operational instruction 47

List of Tables

4.1 Laplacian 3 x 3 mask 31

4.2 Sobel 3 x 3 image region 32

4.3 Sobel G_x 3 x 3 mask region 32

4.4 Sobel G_y 3 x 3 mask region 33

Chapter 1

Introduction

A mouse-driven interface is a very useful tool for performing image processing tasks. This thesis presents a mouse-driven interface for image processing tasks. The interface links two systems: SunView interface system and Androx interface system. The SunView interface is a tool to support interactive, graphics-based applications running within windows on the Sun workstation. The Androx interface is an image processing board which is housed in the Sun workstation running a unix-based operating system. All the image processing tasks are based on this Androx interface system.

1.1 SunView Interface

SunView is an object-oriented system. The most important class of SunView objects are windows and menus. An object can be defined as a software entity presenting a functional interface. There are two classes of windows in SunView: overlapping frames and nonoverlapping subwindows. A menu contains menu items, some of which may have an arrow pointing to the right. The menu item is selected by pressing the right mouse button and this item will be executed when the right mouse button is released. Events are the input of SunView. The event handling and notify procedure are very important concept in the functional interface of a software entity.

1.2 Androx Interface

The Androx system consists of four parallel programmable image array processors which provides four types of image libraries. The user can program these libraries according to his or her algorithm. The image processing algorithm of this mouse-driven interface contains morphological operations, filtering, image acquisition, histogram, logical operations, arithmetic operations, zooming, scrolling, rank, spot, and thresholding.

1.3 Interface Implementation

The mouse-driven interface implementation algorithm integrates the SunView and the Androx system. The SunView software and Androx software are linked together. The parameter-passing problem between SunView software and Androx software is also considered.

1.4 Operational Instructions

All the image processing operational procedures and instructions are described in this chapter. The purpose of this mouse-driven interface is to provide a user friendly interface for performing image processing tasks with the SunView and Androx system.

Chapter 2

SunView Interface

SunView(Sun Visual/Integrated Environment for Workstations) is a user-interface tool to support interactive, graphics-based applications running within windows. SunView is an object-oriented system. We can think of SunView objects as visual building blocks which the user can use to assemble the interface to his applications. The most important class of SunView objects are windows and menus. Technically, we can say that an object is a software entity presenting a functional interface. In SunView system, there are two types of windows: frames and subwindows. Frames can be viewed as a tree of subwindows. The top frame is called base frame. Menus can chain individual menu together into a collection known as a walking menu. The inputs of SunView are events which are generated in the kernel process and application process. The function of the notifier is to control the process and format it into high-level events. In this chapter, we will discuss the window objects, menu objects and events. Finally, we will overview the SunView libraries. See [1] for the SunView in details.

2.1 Windows

There are two classes of windows in SunView: overlapping frames and non-overlapping subwindows. Every window or frame has its attributes. SunView also provides window functions and macros for creating a window, initiating event processing and destroying windows. In this section, we will discuss how to pop up a window, how to manipulate frame via menus and the panel of a window.

2.1.1 Frames and Subwindows

The purpose of a frame is to bring subwindow of different types together into a common framework so that they can be operated on as a unit. A frame is said to own the subwindows it contains. In the basic hierarchy of windows of SunView structure, frames may also own other frames. Frames also can be viewed as a tree of windows in which the non-leaf nodes are frames and the leaf nodes are subwindows. The frame at the top of the hierarchy is called base frame, other frames are called subframes.

2.1.2 Panels

Panels contain items through which the user interacts with a program. There are six basic types of panel items:

Message Item The only visible component of a message item is a label, which may be a title, comments, descriptions or pictures. Message items are selectable, and one may specify. A notify procedure will be called when the item is selected.

Button Item Button items allow user program to initiate commands. Button items, like message items, have a label, are selectable, and have a notify procedure.

Choice Item Choice items allow the user to select one choice from a list.

Toggle Item Toggle items are identical with choice items. Each choice may be either on or off.

Text Item Text items are basically typein fields with optional labels. The user can specify that his notify procedure be called on each character typed in.

Slider Item Slider items allow the graphical representation and selection of a value within a range.

2.1.3 Windows Generating and Destroying Procedure

For window object, there is a set of standard functions to create, destroy the object, and to get and set the object attributes. The window functions are: *window_create*(), *window_get*(), *window_set*(), and *window_destroy*().
The following is an example of creating a window, which also includes a panel:

```
/ * create a window * /

Window frame;

Panel panel; frame = window_create(NULL, FRAME, FRAME_LABEL, Mouse_DrivenInterface", 0);

panel = window_create(frame, PANEL, 0);

panel_create(panel,PANEL_TEXT, 0);

window_fit_height(panel);

window_main_loop(frame);
```

The call to *window_create*() does not display the frame on the screen until the *window_main_loop*() is called. The *window_fit*() macro causes a window to exactly fit its contents. The user can destroy windows with the following call:

```
window_destroy(frame);
```

2.1.4 Manipulating Frames via Menus

A menu is associated with each frame, which allows the user to manipulate the frame directly. The menu is invoked by pressing the right mouse button on the exposed parts of the frame.

2.1.5 Window Pop-ups

In SunView, pop-ups are implemented as subframes containing subwindows. The subframe, along with its subwindows, is displayed in the base frame. In this mouse-driven interface, all the Androx image processing items are invoked by the right mouse button and subwindow will be displayed by the window pop-up.

2.2 Menus

Menus allow the user to chain individual menu together into a collection known as a walking menu. A menu contains menu items, some of which may have an arrow pointing to the right. This indicates to the user that if he or she slides the mouse to the right of that item, a pull-right menu will appear. Menus can be strung together hierarchically in this fashion, so that the user can "walk" down the chain of menus to make a selection.

2.2.1 Menu Usage

Menus are created with the function *menu_create*(), which specifies the appearance of the menu items. It takes a null-terminated attribute list. The user can use the routine *menu_set*() and *menu_get*() to modify and retrieve the value of attributes for both menus and menu items.

2.2.2 Components of Menus Items

A menu item can be displayed as a string. If the item has a submenu associated with it using the *MENU_PULLRIGHT* attribute, then it is a pull-right item. Each menu item is associated with a value, by default the value is the initial ordinal position of the item if it was created with *MENU_STRING*; otherwise the default value is zero. The user also can explicitly set an item's value with *MENU_VALUE*, but if an item is a pull-right, then its *MENU_VALUE* is the value of its pull-right menu. This means that only "leaf" menu items without submenus have a true value. Each menu item has a client data field, accessible through the *MENU_CLIENT_DATA*, which can be used to associate a pointer to a menu structure with each menu item.

2.2.3 Menus Generate Procedure

We programmed the routine *initialize_menu*() to create the menu items and *build_menu*() routine to set up all menus including the pull-right menus. We used the *menu_return_value*() routine as the notify or action procedure to read the return value. If the required selection item is reached, then the action procedure is called.

2.3 Events

SunView is a notification-based system. The notifier acts as the controlling entity within a user process, reading UNIX input from the kernel, and formatting it into high-level events. Events are generated from several sources. These include standard devices such as mouse and keyboard. The notifier weaves events from these sources into a single, ordered event stream.

2.3.1 Notify Procedure

The notifier is a general-purpose mechanism for distributing events to a collection of clients within a process. The notifier reads events and notifies, or calls out to, various procedures which the application has previously registered with the notifier. These procedures are called notify procedure.

2.3.2 Events Generation

In SunView system, events are generated in two processes: kernel process and application process. Figure 2.1 illustrates the event process diagram. In the kernel process, hardware events are input from the physical input devices, such as mouse and keyboard, to kernel device drivers. The kernel device drivers will interpret the hardware events by a "virtual" user input device (VUID) interface. The VUID interface packages the data it receives into firm events and sends them to the window drivers for event mask. The window drivers deliver mask events to the notifier. In the application process, the notifier receives events from window drivers then notifies events to event translation procedure. After action events are translated, the notifier event procedure will received all the events for execution.

2.3.3 Event Handling

How do subwindows handle events ? The canvas[1] and panel subwindows pass events that they receive on to an event procedure. These event procedures are supplied by the application as the value of *WIN_EVENT_PROC*. If the *WIN_EVENT_PROC* of a canvas or a panel is set to a function, then event will be process by that function. Every event has an identifying code, which is

[1]a canvas is essentially a window for drawing or displaying an image

accessed with the macro *event_action*().

2.3.4 Classes of Events

Every event has its event code, we can group each event code into logical class. Specifically, we will look at the following two events:

Locator Button Events The Sun locator is a three button mouse, whose buttons generate the event codes "*MS_LEFT* ", "*MS_MIDDLE* ", and " *MS_RIGHT* ", indicating left button pressed, middle button pressed, and right button pressed, respectively.

Locator Motion Events The locator constantly provides a (x,y) coordinate position in pixels; this position is transformed by SunView to the coordinate system of the window receiving an event. Locator motion event codes include "*LOC_MOVE* ", "*LOC_DRAG* " and others.

Since the locator tracking mechanism reports the current position at a set sampling rate, 40 times per second, fast motions will yield non-adjacent locations in consecutive events. A *LOC_MOVE* event is reported when the locator moves, despite the state of the locator buttons when the user enable *LOC_MOVE* or *LOC_DRAG*. The window system gives user the current locator position by collapsing consecutive locator motion events into one. The location of the event in the window's coordinate system is accessed with the macro *event_x*() and *event_y*().

2.3.5 Enabling and Disabling Events

Event input can be controlled by using input focus and input mask. The input focus is the window that is currently receiving input. The input mask

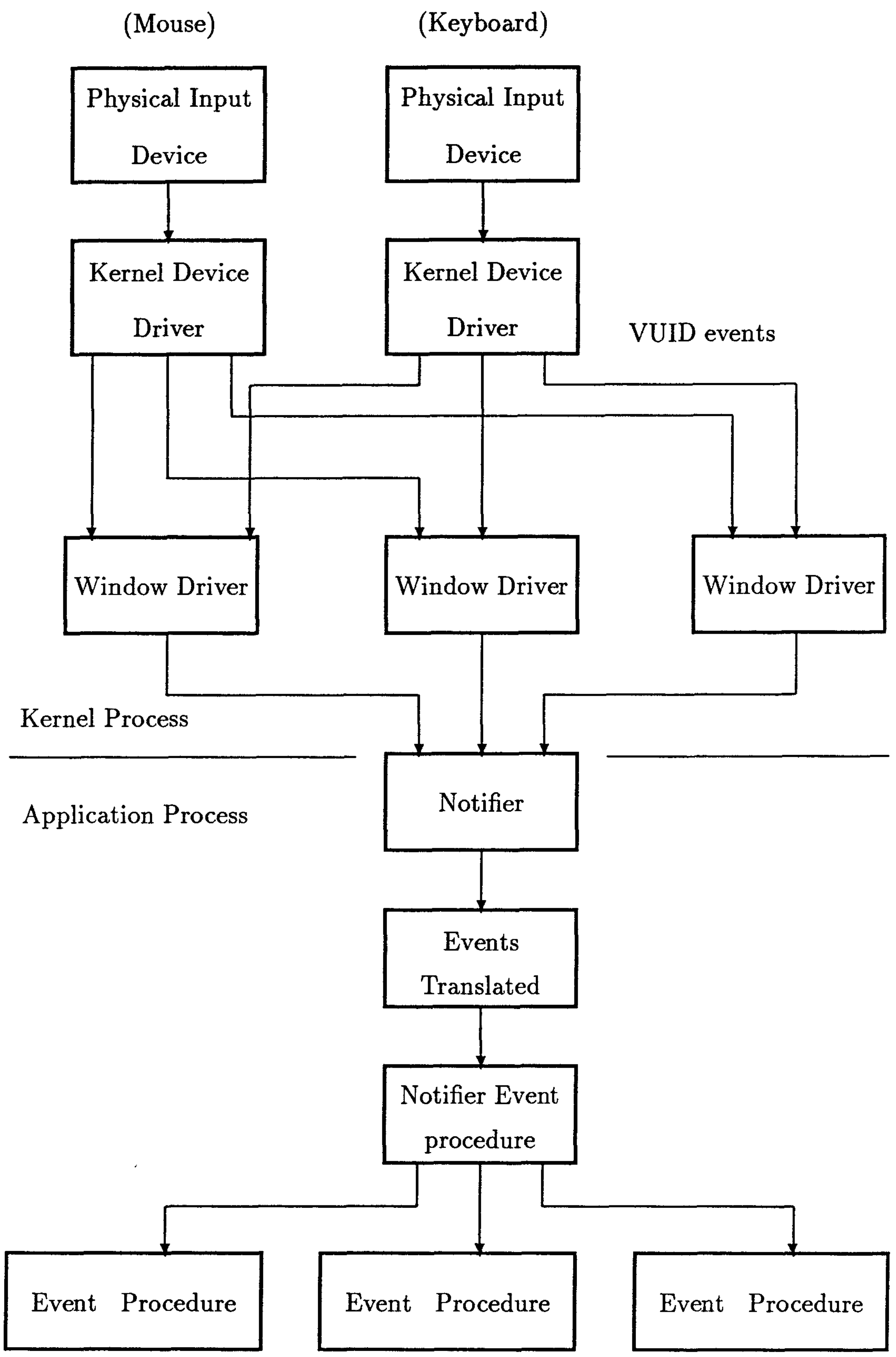


Figure 2.1: The events process diagram

specifies which events a window will receive or ignore. To enable or disable locator motion and button events, we can use the pick mask. An application needing to track mouse motion with the button down would enable *LOC_DRAG* by calling:

window_set(canvas, *WIN_CONSUME_PICK_EVENT*, *LOC_DRAW*, 0);

To disable the mask, the user can use the special *WIN_NO_EVENTS* descriptor :

window_set(*canvas*, *WIN_CONSUME_PICK_EVENT*, *WIN_NO_EVENTS*, *LOC_DRAW*,0);

If the user want to find out whether the *MS_LEFT* is down he or she would call:

window_get(canvas, *WIN_EVENT_STATE*, *MS_LEFT*);

The call will return non-zero if the key is down, and return zero if the key is up.

2.4 SunView Libraries

The SunView functions are mostly in the library file */usr/lib/ libsuntool.a* (if using the archive libraries) and */usr/lib/libsuntool*.so (if using shared libraries). The basic definitions needed by a SunView application are obtained by including the header file:

#< *suntool/sunview.h* >

#< *suntool/panel.h* >

#< *suntool/canvas.h* >

Chapter 3

Androx Interface

The Androx ICS-400 consists of four parallel programmable image array processors featuring multiple digital signal processing chips($DSP's$) with cache memories and a graphics processing chip(GRP). The programmability of the $DSP's$ provides the flexibility to address the board spectrum of image processing applications and parallel processing capabilities result in system performance. The GRP comes with an extensive library of $C - callable$ graphics processing functions, which allow the user to create attractive image processing displays. In this chapter, we will discuss the system structure, memory configurations, event scheduling, and libraries. See [2] for the Androx in details.

3.1 System Structure

Figure 3.1 shows the block diagram of the Androx system structure. In this system structure, the graphics controller is used to transform and transmit graphics data to graphics data RAM. There are four image processing nodes in the system, each node consisting of an analog device ADS2100 and associated cache memory. Image pixel can be transmitted by the way of pixel bus interface to pixel buffer. The analog-to-digital converter (ADC) can convert the video

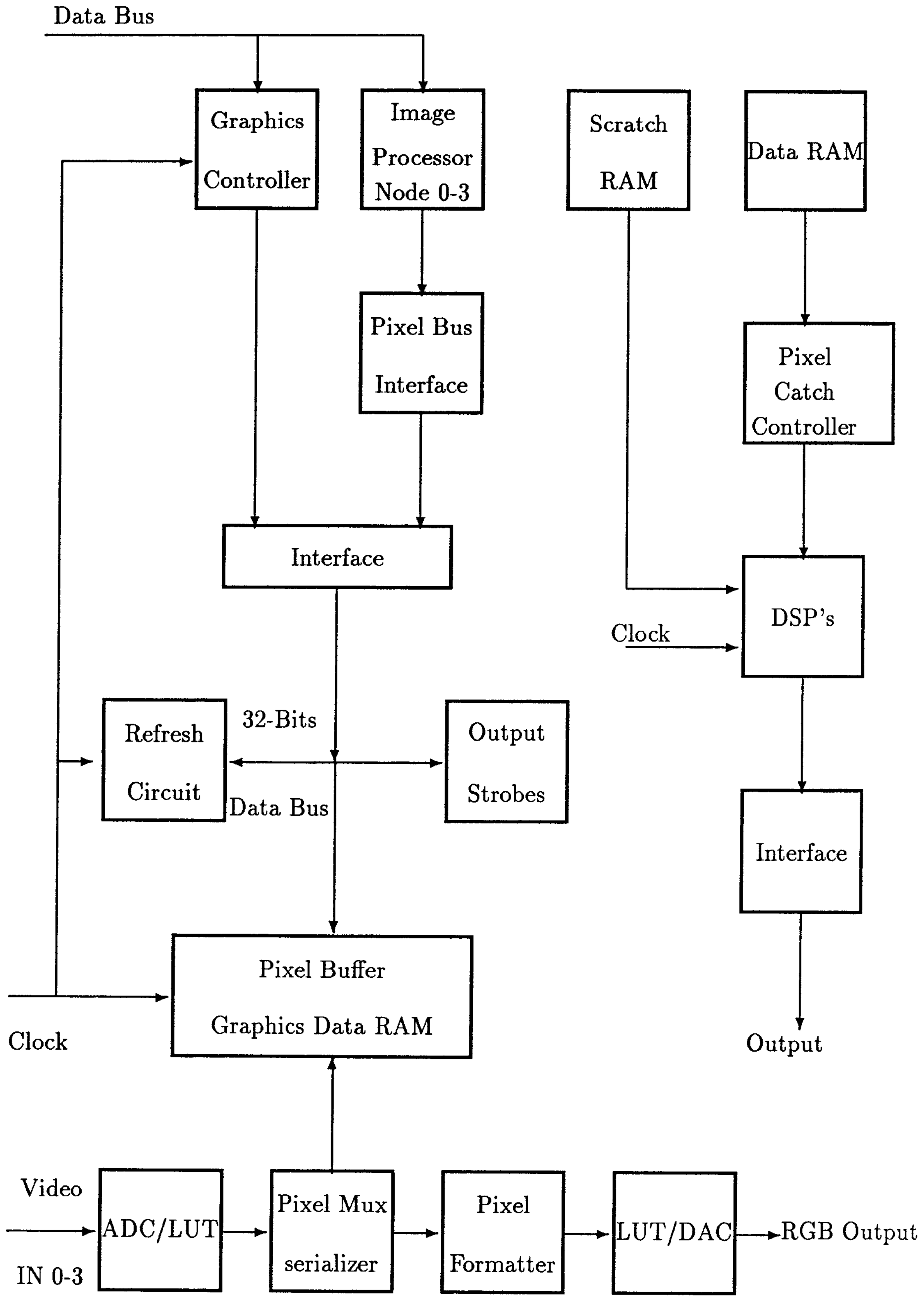


Figure 3.1: The Androx system structure

signal into pixel data. The pixel data can be stored in look up tables (LUTs) or transmitted to pixel serializer and formatter. The formatted pixel data can be transmitted to digital-to-analog converter (DAC) for output. The DSP's can read and process the image data from scratch RAM or data RAM then output the data through data bus interface.

3.2 Memory Configurations

The Androx $ICS - 400XM9$ system has two types of memories: video and scratchpad cache memory. The video memory has eight megabytes memory, which can be addressed as 4096 rows of 2048 bytes. When the user gains control of the ICS with the *attach*() statement, the board is in " $2K$ mode " that is, the eight megabytes video memory is divided into four banks. When displaying the contents of this video memory in RGB mode the four banks are assigned specific color planes:

Bank	Color
0	Red
1	Green
2	Blue
3	Overlay

The three planes of RGB image must be placed in the same relative location within their respective banks. The scratchpad memory has one megabytes of memory, which can be addressed as 512 rows of 2048 bytes and half-megabyte of scratchpad memory is reserved by the graphics processor. Figure 3.2 illustrates the memory mapping of $ICS - 400XM9$.

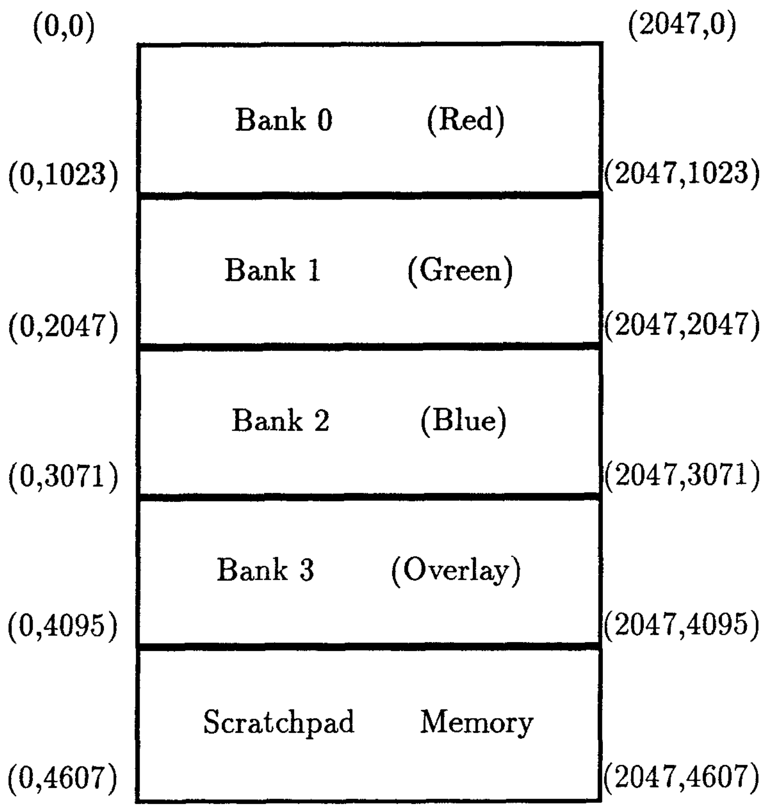


Figure 3.2: The memory mapping of ICS-400XM9

3.3 Event Scheduling

If the host processor program invokes the image processor or graphics processor functions, the ICS generally will execute these functions.

There is a number that allows the user to synchronize execution of image processor and graphics processor functions and the calling host program. This number is called the " event number".

3.3.1 *PEV*(event)

To enable the user to synchronize function execution, ICS functions return event numbers, which can be used as input to other functions. The format is as follow:

```
event x = function(FUNCTION_OPCODE | BRD(x) | PEV(x) | PROC(x),arg1,argn);
```

The event number is from 1 to 255, which can be specified as input to the *PEV*(previous event) parameter of a subsequent function. This event number's flag is cleared only when the function from which the number was returned completes. For example:

```
/ * previous event * /

int event;

event = isp_event(ICOPY|BRD(0),P8 - 8, 0, 512, 512, 512, 0, 0);

grp_event(DRAW_STRING | BRD(0) | PEV(event),35,40,string_addr);
```

3.3.2 Wait Event

Some functions do not accept a *PEV* parameter such as *release*(). To ensure completion of a function before invocation of one of these functions, the user can call *wait_event*(). The *wait_event*() suspends program execution until the specified event number's flag is cleared. The format of wait event is *wait_event*(*BRD*(0)| event), for example:

```
/ * wait event * /

int event;

event = grp_event(PATNFILL_RECT|BRD(0),400,600,10,10);

wait_event(BRD(0) | event);

release(0);
```

3.3.3 Create Event

The *create_event*() function creates a compound event made up of to four other events. The format is as follow:

```
create_event(BRD(0) | control | num_event, event1, event2, event3, event4);
```

where "control" specifies when the compound event will be considered completed. If zero, it means one of the events has completed. If *CMPND_ALL*, it means all the events have completed. For example:

```
/ * create event * /

int event1, event2, event3;
event1 = isp_event(ROTATE_2D|BRD(0),0,0,512,512,0,512,
         256,256,0,0,128,128,DEGREES(75));
event2 = grp_event(PATNFRAME_RECT|BRD(0), 600,
         450, 0, 0, 5, 7);


event3 = create_event(BRD(0)|CMPND_ALL| 2,event1,event2);

wait_event(BRD(0)|event3);

release(0);
```

The event3 will wait until event1 and event2 have finished before exiting.

3.4 Androx Libraries

There are four types of libraries for the user to program his or her applications. They are: applications library, graphics library, image processing library, and video library.

3.4.1 Applications Library

The applications library contains some useful functions that simplify certain aspects of writing a program. The functions are contained in the file

/usr/lib/androx/libandrox.a. Some of the functions in this library are built from lower-level library functions, and can be written by the user.

3.4.2 Graphics Library

The graphics processor library functions are invoked through a graphic monitor by a call to the host function *grp_event*(). Its format is as follow:

event x = *grp_event(FUNCTION_OPCODE* | *BRD*(0) | *PEV*(*x*), *arg*1, *argn*);

The graphics processor can execute only one function at a time. All the graphics processor library function names are defined in the include file *gxlib.h.* A host processor program that calls graphics processor library functions must include the files:

#include < androx/axdef.h >

#include < androx/gxlib.h >

3.4.3 Image Processing Library

All the image processing library functions are invoked through calls to the host function *isp_event*(). Its format is as follow:

event x = *isp_event(FUNCTION_OPCODE* | *BRD*(0) | *NPROC*(*x*) | *PEV*(*x*), *arg*1, *argn*);

The library routines can return linear data or two-dimensional data to global memory. The global memory consists of two sections: one used for image acquisition and display, referred to as "video memory", and the second used as a scratchpad and communications are for the data processing nodes, referred to as "scratchpad memory". A host processor program that calls image processor library functions must include the files:

#include < androx/axlib.h >

#include < androx/axdef.h >

3.4.4 Video Library

Acquisition and display are programmed with the video library. The video library contains two levels of function calls. The low level functions are scheduled events referred to as *vid_event*() calls. The second level functions simplify the task of programming the video hardware. To accommodate various acquisition and display timing formats, the ICS has a programmable video timing register and a programmable pixel clock. The ICS also has various synchronization sources for horizontal, vertical, composite, and externally generated composite video. The ICS has four independent video input channels from 0 to 3. Acquisition of RGB input uses channels 0, 1, and 2 for red, green, and blue, respectively. Acquisition of monochrome input can be achieved through any of the four channels. The ICS has eight look-up tables(*LUTs*) on each input channel. The input *LUTs* transform incoming video data. All the *LUTs* can be defined by the user.

Chapter 4

Interface Implementation

This chapter describes the mouse-driven interface which integrates the SunView interface and Androx interface. Parmeter-passing between these two interface is also discussed in this section.

4.1 SunView

The SunView interface is composed of menus and selection structure, event structure, and subwindows pop-up structure.

4.1.1 Menus and Selection Structure

The hierarchical menu structure is designed as a recursive structure. The components of the structure are "key", " name", and "next". Key is used as the menu return value when the menu item is selected by the user. The key value is an integer and is predefined by the user. Name is a pointer of the name of the menu. Next is a pointer that point to the next submenu. If a NULL, the terminator of the structure, appears after an item, it means that this item is the end leaf of the recursive structure. The selection menu is starting from the top menu to next submenu. Figure 4.1 is an example of hierarchical menu structure. Following is this example structure algorithm.

```
/ * menu structure algorithm * /

#define NULL 0

#define OPTION struct MENU

#define Item1 1 / * menu return value is 1 * /

#define Item2.1 4 / * menu return value is 4 * /

#define Item2.2 5 / * menu return value is 5 * /

#define Item3.1.1 7 / * menu return value is 7 * /

OPTION {

char key; / * menu return item * /

char *name; / * pointer of menu name * /

char *next; / * pointer of next menu * /

};

static OPTION menu31[] ={

7, "Item3.1.1", NULL,

NULL, NULL, NULL

};

static OPTION menu3[] = {

6, "Item3.1", menu3.1, / * to submenu menu3.1 * /

NULL, NULL, NULL

};

static OPTION menu2[] = {

4, "Item2.1", NULL,

5, "Item2.2", NULL,

NULL, NULL, NULL

};

OPTION Topmenu[] = {
```

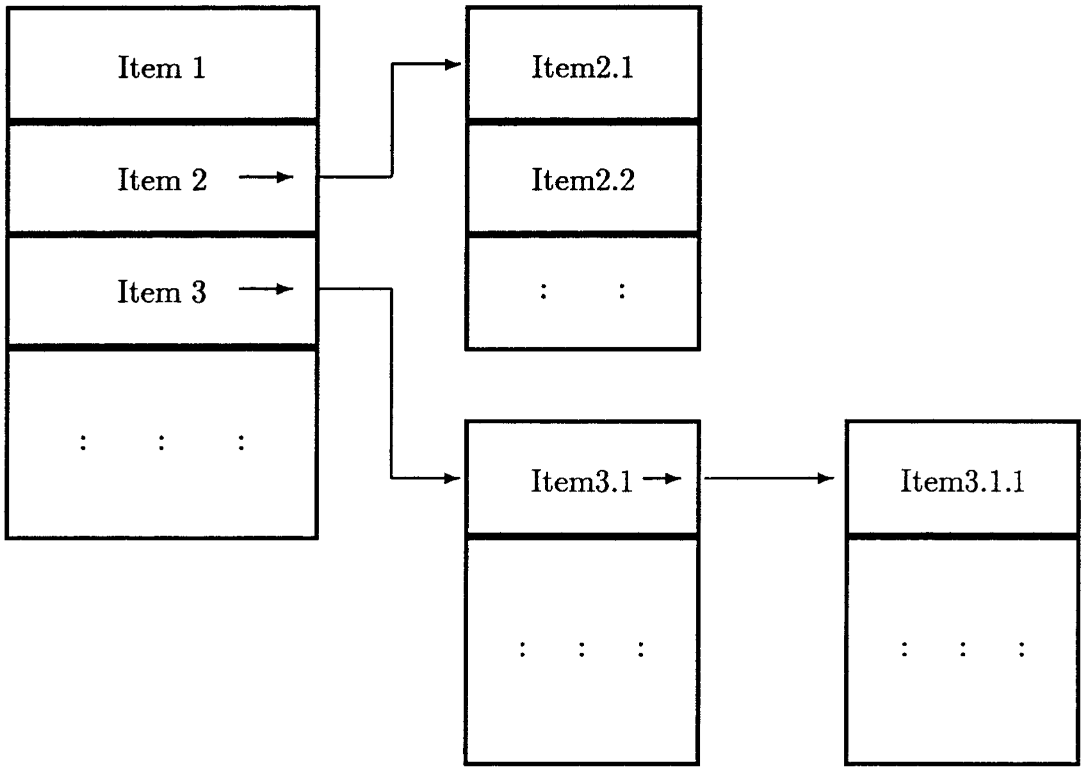


Figure 4.1: Hierarchical menu structure

```
1, "Item1", NULL,
2, "Item2", menu2, / * to submenu menu2 * /
3, "Item3", menu3, / * to submenu menu3 * /
NULL, NULL, NULL
};
```

The menu is selected in the canvas by pressing the right mouse button. Once the menu item is selected, the WIN_EVENT_PROC will pass the event to the event procedure, the routine $Testevent()$ and the event return value will be passed to the routine *selection*(). The selection routine will select the desired function to execute according to the return value. Following is the algorithm selection:

```
/ * the algorithm of selection * /
/ * where i is the menu return value * /
:
:
WIN_EVENT_PROC, Testevent,
:
:
Testevent(win, event)
Frame win;
Event *event;
{
int i;
if (win != canvas) return;

switch (event_action(event)) { / * check event code */
        case MS_RIGHT: / * event code is mouse Right button * /
        i = (int)menu_show(top_menu, win, event, 0);
        if (i != 0) selection(i); / * go to selection items * /
        break;
        :
        :
        }
}

selection(i) / * the event code selection items */
int i;
{

switch(i) {
          case Histogram: / * the return value is histogram * /
          androx_mouse();
          break;
          case Laplacian: / * the return value is laplacian * /
          laplacian_image();
          break;
          :
          :
          }
}
```

4.1.2 Event Structure

According to section 2.3, we know that the notifier formats the user inputs into events. In this section, we will discuss the following two events: the panel button events and mouse button events.

Panel Button Events

When the left mouse button is pressed over a button item, the item's rectangle is inverted. When the mouse button is released over a button item, indicating that the item has been selected and the command is being executed. This procedure is specified via the attribute *PANEL_NOTIFY_PROC*. Following is an example of this event. When the "Accept" panel button is pressed, the *load_image_proc* routine will be executed.

```
/ * panel button event procedure * /

panel_create_item(load_image_panel, PANEL_BUTTON,

PANEL_ITEM_X, ATTR_COL(0),

PANEL_ITEM_Y, ATTR_ROW(6),

PANEL_LABEL_IMAGE,panel_button_image(panel,

" Accept ", 0 , 0 ),                    / * create the Accept panel button * /

PANEL_NOTIFY_PROC, load_image_proc, / * the notify event * /

0);

:

:

load_image_proc() / * the file for notifier execution * /

{

FILE *fp, *fopen();

char *val;

val = (char *)panel_get_value(load_fname_item);

if ((fp =fopen(val, "r")) == NULL) {

print_load_msg("can't open file");

return(1); }

load_image_file(val); / * load an image file routine * /

fclose(fp);

}
```

Mouse Button Events

We use mouse buttons and locator as the mouse event inputs. When the event identifying code is accessed by the routine *event_action*(), all the "*MS_RIGHT*", "*MS_MIDDLE*", "*MS_LEFT*", "*LOC_MOVE*", and "*LOC_DRAG*" can be identified and each event will be executed. In the mouse-driven interface, *MS_RIGHT* is used to select menu items. *LOC_MOVE* is used to get the location of the event in the window's coordinate, then the coordinate values will be passed to Androx system for displaying Androx mouse cursor in a RGB monitor. *MS_LEFT* is used to enable or disable the *LOC_DRAG*. If *MS_LEFT* is not in the up event, the user can move the locator to draw a line, a rectangle, or a dot in the RGB monitor. If *MS_LEFT* is in up event a histogram will appear in the RGB monitor. The *MS_MIDDLE* is used to pop up the gray value in *RGB* monitor for a spot area. Following is the structure of these events:

```
/ * structure of mouse events * /
switch(event_action(event)){

case MS_RIGHT: / * the mouse right button event code * /
     i =(int)menu_show(top_menu, win, event, 0);
     if (i !=0) selection(i);

break;

case LOC_MOVE: / * the mouse locator moving event code * /
     if(state == AXMOUSE){
     mx = event_x(event); / * x coordinates value */
     my = event_y(event); / * y coordinates value */
     move_androx_mouse(mx - sun_x, my - sun_y);
     sun_x = mx; sun_y = my; }

break;

case MS_LEFT: / * the mouse left button event code * /
     if( !event_is_up(event)) {
     window_set(win, WIN_CONSUME_PICK_EVENTS,
     LOC_DRAG, WIN_MOUSE_BUTTONS, WIN_UP_EVENTS, 0, 0);
     l_sun_x = sun_x; l_sun_y = sun_y;
     else histogram();

break;
case LOC_DRAG: / * the mouse locator drawing event code * /
     if(window_get(win, WIN_EVENT_STATE, MS_LEFT)){
     mx = event_x(event);
     my = event_y(event);
     drawing(sun_x - l_sun_x, sun_y - l_sun_y,
     mx - l_sun_x, my - l_sun_y);
     sun_x = mx; sun_y = my; }

break;

case MS_MIDDLE: / * the mouse middle button event code * /
     if(!event_is_up(event)) {

mouse_spot(sp1, sp2);
            }


break;

}
```

4.1.3 Subwindow Popup Structure

There are two types of popup windows: non-blocking pop-up and blocking pop-up. Blocking pop-up are not used and will not be described here. The display of a non-blocking pop-up is controlled by the *WIN_SHOW* attribute. If the user want to pop up a subwindow, the *WIN_SHOW* should be set to TRUE. The user can use the routine of *window_destroy*() to destroy the subwindow. The purpose of using popup subwindow is to get inputs from the user to set parameters. There are six types of panel item inputs as in mention section 2.1.2. Following is an example of subwindow popup structure:

```
/ * subwindow pop up structure * /
load_proc()
{
load_image_popup();
/ * set the WIN_SHOW to TRUE * /
window_set(load_image_frame, WIN_SHOW, TRUE, 0);
}
:
load_image_popup() / * create the load file subwindow * /
{
load_image_frame = window_create(frame, FRAME, 0);
load_image_panel = window_create(load_image_frame,PANEL,0);

load_item= panel_create_item(load_image_panel,PANEL_TEXT,
          PANEL_ITEM_X, ATTR_COL(0),
          PANEL_ITEM_Y, ATTR_ROW(0),
        PANEL_VALUE_DISPLAY_LENGTH, 40,
        PANEL_LABEL_STRING,"Load File:",
        PANEL_VALUE, temp,
        0);

panel_create_item(load_image_panel, PANEL_BUTTON,
                PANEL_ITEM_X, ATTR_COL(0),
                PANEL_ITEM_Y, ATTR_ROW(5),
                PANEL_LABEL_IMAGE, panel_button_image(panel,
                "Accept", 0, 0), / * create "Accept" button * /
                PANEL_NOTIFY_PROC, load_image_proc,
                0); / * to execute load a file event * /
                :
                :
window_fit(load_image_panel);
window_fit(load_image_frame);
window_set(load_image_panel, PANEL_CARET_ITEM,
                load_fname_item, 0);


load_image_proc() / * the load file subroutine * /

{

char *val;

val = (char*)panel_get_value(load_fname_item);

load_image_file(val);

}
```

4.2 Androx

The Androx interface consists of the accessing and releasing system and the image processing functions, which are implemented in this mouse-driven interface.

4.2.1 Accessing and Releasing System

To access the Androx system, the program must call the *attach*(0) to gain control of the ICS before calling any routine that access the board and call *release*(0) to relinquish control of the ICS before existing. For example:

```
/ * attach and release the ICS * /
#include < androx/axdef.h >
#include < androx/axlib.h >
```

```
#include < androx/gxlib.h >
#include < androx/ax_color.h >

main()

{

int event;

attach(0); / * access the Androx system */

event = grp_event(INIT_SCREEN| BRD(board#)| PEV(event));

:

:

release(0); / * release the Androx system * /

}
```

4.2.2 Image Processing Functions

The following image processing functions are implemented in this mouse-driven interface. See [3] to [9] for the image processing in details.

Load and Save Image File

The *file_to_rgbimage*() routine is used to load a color (*RGB*) image file from host memory to ICS video memory and the *rgbimage_to_file*() routine is used to save a color (*RGB*) image file from ICS video memory to host memory. The RGB image file in ICS memory is loaded in bank0(R), bank1(G), and bank2(B), respectively.

Convolution

The purpose of convolution is used to accomplish different effects in image filtering and segmentation. The filtering can be used to suppress certain spatial frequencies in an image. A linear operation can be defined as space invariant if the response to $\delta(x-\alpha, y-\beta)$, which is a point source at (α, β) in the *xy_plane*, is given by $h(x-\alpha, y-\beta)$. For a linear sum of point sources $f(x,y)$, we can write $f(x,y)$ as:

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(\alpha - x, \beta - y) d\alpha d\beta$$

Let the response of the operation to the input $f(x,y)$ be denoted by R[f]. We obtain:

$$R[f(x,y)] = R[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\delta(\alpha - x, \beta - y) d\alpha d\beta]$$

The response to $\delta(\alpha - x, \beta - y)$, which is a point source at (α, β), is given by $h(x-\alpha, y-\beta)$. We denote $R[f]$ by g. We obtain:

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha, y-\beta) d\alpha d\beta$$

The right-hand side is called convolution of *f* and h. It also denoted as $g = f * h$. The convolution is also superimposing an $m \times n$ kernel over an $m \times n$ pixel area(window) in the image. In this mouse-driven interface, the $m \times n$ kernel is input by the user. We performed the convolution by calling the function *CONV_SK*.

Laplacian

The purpose of Laplacian is used to find edge elements. It sums the second partial derivatives of the image intensity in the x and y directions. The Laplacian is an orientation-invariant derivative operation defined as:

$$L[f(x,y)] = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

0	1	0
1	-4	1
0	1	0

Table 4.1: Laplacian 3 x 3 mask

Table 4.1 is the Laplacian 3 x 3 mask. If a vector **W** is formed from the coefficients of this mask, we can express the Laplacian in vector form as:

$$L[f(x,y)] = \mathbf{W'X}$$

Where **X** is a vector containing the pixel values. The Laplacian is 0 in constant areas and on the ramp section of an edge, as expected of a second-order derivative. We performed the Laplacian for the edge enhancement by the function LAPLACIAN.

Sobel

Sobel operation yields the magnitude of the brightness gradient as a means of edge detection in an image. Edge detection is an important process employed in image segmentation. The object of an edge operation is to detect the presence and location of gray level change in an image. The Sobel operator using a 3 x 3 window approximate the local edge by gradients. Assume the gradient of an image $f(x,y)$ at location (x,y) is defined as the two-dimensional vector

$$\mathrm{G[f(x,y)]} = \begin{bmatrix} \mathrm{G_x} \\ \mathrm{G_x} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

The magnitude of this vector is

$$\mid \mathrm{G[f(x,y)]} \mid = \sqrt{[\mathrm{G_x^2} + \mathrm{G_x^2}]}$$

x1	x2	x3
x4	x5	x6
x7	x8	x9

Table 4.2: Sobel 3 x 3 image region

-1	-2	-1
0	0	0
1	2	1

Table 4.3: Sobel G_x 3 x 3 mask region

Let $\alpha(x, y)$ represent the direction angle of G at location (x, y), the direction of gradient vector is

$$\alpha(x, y) = \arctan(\frac{\mathbf{G_y}}{\mathbf{G_x}})$$

If we define the component of the gradient vector in the x direction as :

$$\mathbf{G_x = (x_7 + 2x_8 + x_9) - (x_1 + 2x_2 + x_3)}$$

and the y direction as:

$$\mathbf{G_y = (x_3 + 2x_6 + x_9) - (x_1 + 2x_4 + x_7)}$$

where $\mathbf{x_i}$, $i = 1, 2, ..., 9$ is the 3 x 3 image region as shown in table 4.2.

We can obtain the 3 x 3 mask region of G_x and the 3 x 3 mask region of G_y as shown in table 4.3 and table 4.4 respectively. The 3 x 3 mask region of G_x and G_y are the Sobel filter's kernel for the horizontal and vertical direction. We performed the function of "Sobel Horizontal" to enhance horizontal edges,

-1	0	1
-2	0	2
-1	0	1

Table 4.4: Sobel G_y 3 x 3 mask region

the function of "Sobel Vertical" to enhance vertical edges, and the function of "Sobel Mag" for horizontal and vertical convolution, then create the sum of the results' absolute values.

Median

The operation of median filtering is to sort the pixel values under the kernel, locate the middle value in the sequence, and place that value in the pixel position under the center of the kernel. This method is particularly effective when the noise pattern consists of strong, spikelike components, and where the characteristic to be preserved is in edge sharpness. We performed the median filtering under a 3 x 3 kernel by the function $MEDIAN\_3X3$ of the *isp_event*() routine.

Gaussian

The purpose of Gaussian filter is used for smoothing image noise. The Gaussian function is

$$g(\sigma) = \frac{1}{\sqrt{2\pi}} e^{-\sigma^2/2}$$

The Gaussian integral function will be equaled to one, if the integral range is from negative infinity($-\infty$) to positive infinity($+\infty$). Following is its integral function

$$G(\infty) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-\sigma^2/2} d\sigma = 1$$

Histogram

Given an image *f*, let $P_f(z)$ denote the relative frequency with which gray level z occurs in f, for all z in the gray level range $[z_1, z_k]$ of *f*. The graph of $P_f(z)$ as a function of z, normalized so that the $\int_{z_1}^{z_k} P_f(z)dz$ is equal to the area of f is called the histogram of f. The histogram is generated in the following five steps:

Step1 The event routine *event_x*() and *event_y*() are used to obtain the location of the x and y coordinate values of the SunView mouse locator, then these two values will be passed to the Androx.

Step2 In Androx, the *BIT_EXPAND* function is used to set up a mouse cursor in the monitor, which then receive the SunView mouse locator x and y values.

Step3 When the SunView left mouse button is pressed, the mouse locator x and y values will be passed to Androx and these values will be used as the original point for selecting an area of picture in monitor.

Step4 Before the SunView left mouse button is up, the mouse locator x and y values will be passed to Androx continuously when the locator is moving. The x and y values will be checked in Androx when these values are received, then the user can according to the successive x and y values to draw a picture area in monitor. This picture area can be a dot, a line, a rectangle, or a quadrate.

Step5 After the SunView left mouse button is up, the histogram of this picture area will be displayed in monitor.

We performed the histogram by the *do_histo*() routine.

Thresholding

Thresholding is an important technique for image segmentation. Suppose we have a function T of the form:

$$T = T[x, y, p(x, y), f(x, y)]$$

where *f(x,y)* is the gray level of point *(x,y)* and *p(x,y)* denotes some local property of this point–for example, the average gray level of a neighborhood centered at *(x,y)*. We create a threshold image *g(x,y)* by defining

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \leq T \end{cases}$$

In examining g(x,y), we found that pixels labeled 1 correspond to objets, while pixels labeled 0 correspond to the background. We use the AH(2) function, a two-dimensional to two-dimensional data transfer method of image-read routine, to read image into host memory and compare each gray level with desired threshold level and label each pixel with 1 or 0, then use image-write to output image.

Pseudo Color

The purpose of pseudo color is to convert a black-white image into a color image. It can be defined as follow

$$R(x,y) = O_R\{F(x,y)\}$$

$$G(x,y) = O_G\{F(x,y)\}$$

$$B(x,y) = O_B\{F(x,y)\}$$

where the F(x,y) is an input image. $O_R\{\}$, $O_G\{\}$, and $O_B\{\}$ are the output functions of red, green, and blue respectively. We performed this function by the *load_pseudo_color*() routine.

Zoom and Scroll

The method of bilinear interpolation is used to zoom an area of interest. The bilinear interpolation approach uses the gray levels of the four nearest neighbors. Consider a location (x, y), which yield the spatial mapping into location $(\hat{x}, \hat{y})$ and let the gray-level value of $(\hat{x}, \hat{y})$ be $v(\hat{x}, \hat{y})$. Suppose the integer parts $\lfloor \hat{x} \rfloor$, $\lfloor \hat{y} \rfloor$ of $\hat{x}$ and $\hat{y}$ are x and y, so that the point $(\hat{x}, \hat{y})$ is surrounded by the four integer-coordinate points:

(x,y)		$(x+1,y)$
	$(\hat{x},\hat{y})$	
$(x,y+1)$		$(x+1,y+1)$

Let the fractional parts of $\hat{x}$, $\hat{y}$ be A and B, where $A = \hat{x} - \lfloor \hat{x} \rfloor$ and $B = \hat{y} - \lfloor \hat{y} \rfloor$, thus $0 \leq A, B < 1$, and we have the gray level as follow :

$$\begin{aligned} v(\hat{x},\hat{y}) &= (1-A)(1-B)f(x,y) + (1-A)Bf(x,y+1) \\ &\quad + A(1-B)f(x+1,y) + ABf(x+1,y+1) \end{aligned}$$

We actually interpolate a value at location $(\hat{x}, \hat{y})$ and use this value for the gray-level assignment at (x, y). We use the function "$BILIN$" to perform this zoom operation. Scrolling is the process of moving an area within a rectangular region in any direction. We use the function "SCROLL" to perform this scrolling operation. The direction of scrolling is determined by dx and dy.

dx Horizontal direction. Negative value is scrolling to the left, positive value is scrolling to the right.

dy Vertical direction. Negative value is scrolling up, positive value is scrolling down.

Morphological Operations of Image

Morphological erosion and dilation of binary images are defined from a geometric point of view as set transformations that shrink or expand a set. Algebraically, they are actually Minkowski set subtraction and addition, respectively. Let two sets A and B, vectors a and b belong to the sets A and B. The Minkowski set addition $A \oplus B$ is defined as

$$A \oplus B = a + b : a \in A, b \in B = \bigcup_{b \in B} A_b$$

The Minkowski subtraction $A \ominus B$ is defined as

$$A \ominus B = (A^c \oplus B)^c = \bigcap_{b \in B} A_b$$

Let x be a binary image, B a structuring element, and B^s the symmetric set of B. It means that B^s is rotated 180 degrees. If the erosion of x by B is z, then the subtraction of B^s from x is

$$x \ominus B^s = z, B_z \subseteq x = \bigcap_{b \in B} x_{-b}$$

If the dilation of x by B is z, then that sum of x and B^s is

$$x \oplus B^s = z : B_z \cap x \neq \phi = \bigcup_{b \in B} x_{-b}$$

Based on the Minkowski set of subtraction and addition, we can use the *isp_event*() routine to perform binary erosion and dilation respectively. The binary open is performed by first erosing and then dilating. The binary close is performed by first dilating and then erosing. The gray scale morphological operations such as dilation, erosion, open, and close are also implemented. The gray scale morphological operations are performed by using structuring element and a gray image. The user should input the structuring element data file from the panel before executing the binary or gray scale morphological operations. In order to represent the structural shape of a plane region, an important approach is to reduce it to a graph. This approach is often accomplished by obtaining the skeleton of the region via thinning algorithm. We performed the thinning algorithm by the routine *skeletonize*().

Digitize MONO and RGB Image

The method for digitizing monochrome image is to set up the camera, monochrome acquisition and display the video memory bank into which live video will be written. We performed this function by *mono_init*() and grab() routines. The method for digitizing color image is also to set up camera, color acquisition and display the video memory bank into which live video will be written. We performed this function by *rgb_init*() and *grab*() routines.

Look up table

There are eight look-up tables (*LUTs*) on each input channel. The input LUTs can transform incoming data. The *LUTs* can be programmed by the user. We performed this function by *program_lut*() and *select_lut*() routines.

Logical and Arithmetic Operations

The logical operations AND, OR and XOR are for operation in two areas of interest. We also implemented the arithmetic operations of the pixel values in two areas of interest by the functions of ADD, SUB, and SUM.

Max Min and Rank Image

The purpose of MAX is to determine the maximum value of corresponding pixels in two areas of interest. We performed it by the function of MAX2. The purpose of MIN is to determine the minimum value of corresponding pixels in two areas of interest. We performed it by the function of MIN2. The purpose of RANK is to specify the rank order filter with a $n \times m$ area and operate this rank in it. The user must specify the rank order a $n \times m$ window before executing.

Data Move and Reset

The purpose of DATA MOVE is to move an image data from one place to another in the video memory. This function is performed by using the $pan()$ routine. The purpose of RESET is to reset all image data in the video memory.

4.3 Passing Parameter

In the Androx system, a dedicated image program is called the subroutine. The purpose of a subroutine is to perform some image algorithm. A subroutine may take the number x in its function operation and the operation of this subroutine function depends on x. In logic, we can write this subroutine function as f(x), the variable x is called an argument or a parameter.

When a subroutine is called, certain parameters are specified; these may be variables, values or names. These are called the "actual parameter" associated with this particular call to the subroutine. To determine what the subroutine does to these parameters, we check the definition of the subroutine. Associated with this definition there will be certain parameter names, which are the "formal parameters" or "dummy parameters". When the subroutine is called, each actual parameter is associated with or bound to one of the formal parameters. For example:

```
/* actual parameters val, i1, j1 */
main()
{
char *val;
int i1, j1;
load_image_file(val, i1, j1);
}

/ * formal parameters val, i1, j1 * /

load_image_file(val, i1, j1)

char *val;

int i1, j1;

{

file_to_rgbimage(0, i1, j1, 512, 512, P8 - 8, val);

}
```

4.3.1 Variables

There are two types of variables in subroutine: local and global variable. If a variable is recognized only by the function which included this variable, this variable is called a local variable. If a variable can be recognized by all functions, this variable is called global variable. Any reference to a global variable or global symbol from outside the object program which contains, it is known as an external reference. For example:

```
/ * reference external file * /
```

```
extern move_androx_mouse(); / * define external file * /
int sun_x, sun_y; / * define global variables as integer * /
int mx, my;
:
:
Testevent()
{
:

case LOC_MOVE :
     move_androx_mouse(mx - sun_x, my - sun_y);/ * global variables * /



:


}

/ * This file is called by external * /

move_androx_mouse(dx, dy)

int dx, dy;

{

int event; / * local variable * /

:

wait_event(BRD(0) | event); / * using local variable * /

}
```

4.3.2 Subroutine Call

In subroutine call procedure, we use two types of parameter-passing methods: passing the pointer and passing the value. When a subroutine is called, a value or a pointer is assigned to a formal parameter, this value is stored in the corresponding actual parameter. For example:

```
/ * passing the pointer val to subroutine * /
load_image_proc()
{
char *val
val = (char*)panle_get_value(load_fname_item);
load_iamge_file(val); / * call subroutine and pass pointer val * /
}
load_iamge_file(val) / * the called subroutine * /
char *val;
{
```

```
file_to_rgbimage(0, i1, j1, 512, 512, P8 - 8, val);
}
/ * passing the value s to subroutine * /
thresh_image_proc()
{
int s = (int)panel_get(thresh_item1, PANEL_VALUE);
xthresh(s); / * call subroutine and pass integer value s * /
}

xthresh(s) / * the called subroutine * /

int s;

{

}
```

4.4 Software Compile and Link

The command for compiling the program in SunView is

cc −o −g myprog myprog.c −lsuntool −lsunwindow −lpixrect

The command for compiling the program in Androx is

cc −o −g myprog myprog.c −L/usr/lib/androx −landrox

A makefile is used to compile and link all programs. For example:

```
/ * the makefile contents * /

AXLIB = −L/usr/lib/androx −landrox

SUNLIB = −lsuntool −lsunwindow −lpixrect

CFLAGS = −c

OBJ = menu.o ax2.o help.o spot.o erode2.o rank.o logical.o

.SUFFIXES: .C .O

.C.O :

cc $(CFLAGS)$?

all : (OBJ)

cc (OBJ) −o menu (AXLIB) (SUNLIB)
```

The AXLIB represents the Androx library. The SUNLIB represents the Sun-

View library. The makefile will trace the latest object file, if a program was changed contents then it will be compiled to generate a new object file. The makefile will link all programs together and generate an execution file.

Chapter 5

Operational Instructions

There are two types of file which must be created before executing the mouse-driven interface. They are image file and data or kernel file.

Image File The user can load or save image file into the memory banks. For color image, the three planes of RGB image must be placed in the relative location within their respective banks. The video memory is divided into eight banks. Each bank has three 512 x 512 locations.

Data File The user must load data or kernel coefficients file in some operations. The format of the data or kernel coefficients should be created correctly before loading for execution. The general format of the file is as follows

```
n       m
A11 A21 ................An1
A12 A22 ................An2
:                       :
:                       :
A1m A2m ................Anm
```

where the n and m are the two-dimensional array of rows and columns. A_{ij} is the value of kernel coefficients at the location of the i^{th} row and j^{th} column.

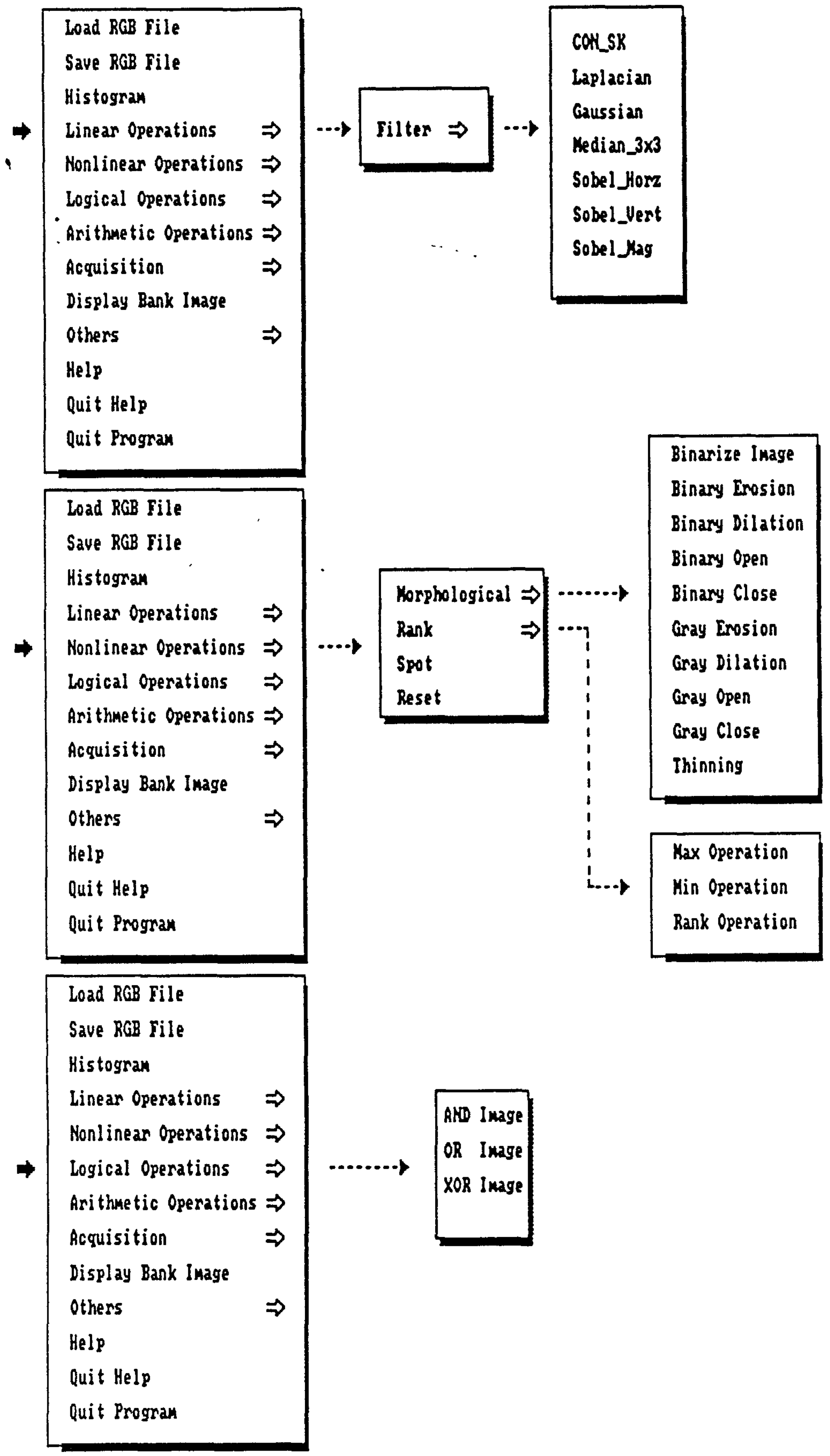


Figure 5.1: The menu items of mouse-driven interface

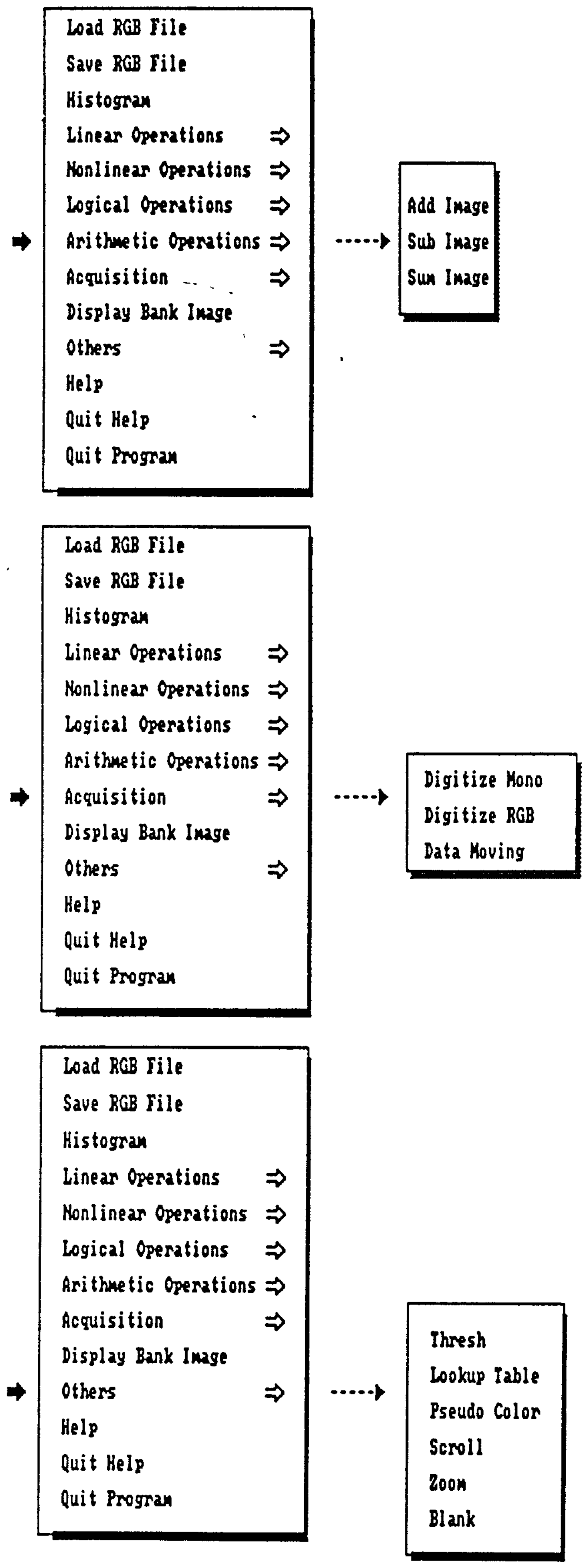
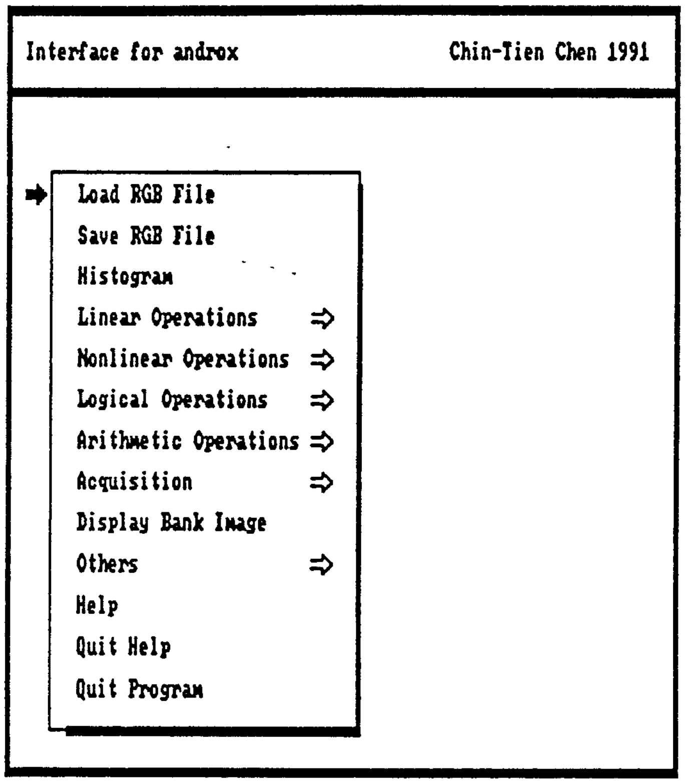
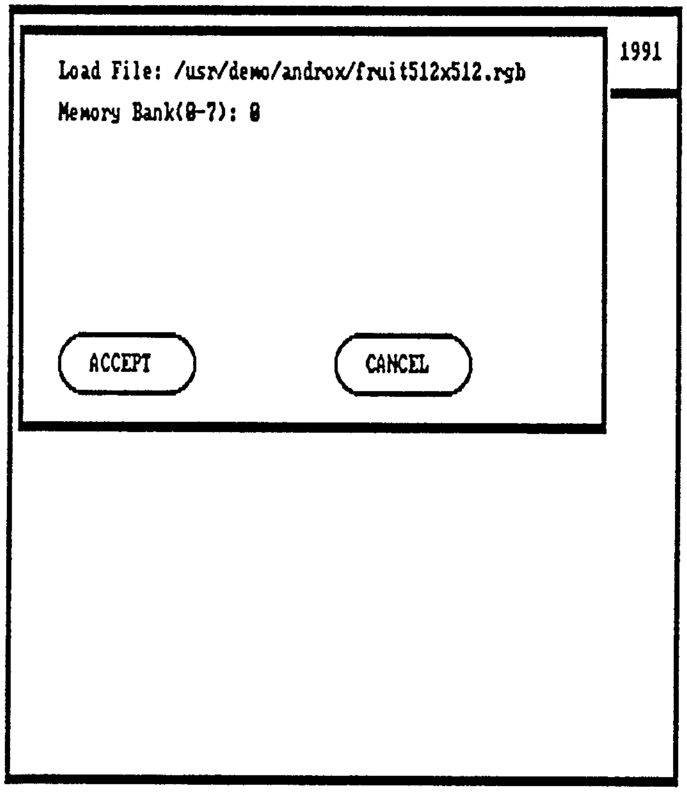


Figure 5.2: The menu items of mouse-driven interface (continued)





Load File: /usr/demo/androx/fruit512x512.rgb
Memory Bank(0-7): 0

Figure 5.3: An example of menu item operational instruction

Figure 5.1 and figure 5.2 illustrate the menu items of this mouse-driven interface. Some of the items have an arrow pointing to the right. This indicates to the user that if he or she slides the mouse to the right of that item, a pull-right menu will appear. Figure 5.3 illustrates an example of subwindow popup for the operational instruction of "Load RGB File" menu item. The user can type in file name and memory bank in panel, then select the "Accept" button to execute or select "Cancel" button to quit subwindow. Following are the operational instructions of the mouse-driven interface.

5.1 Load RGB File

Select the **Load RGB File** menu item from menu item streams. Type in **File Name** or the **Path** if needed and select the **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.2 Save RGB File

Select the **Save RGB File** menu item from menu item streams. Type in **File Name** or the **Path** if needed and select the **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.3 Histogram

Select the **Histogram** menu item. Move the Sun mouse in the canvas area(must be in range), the Androx mouse cursor will move simultaneously. Select one point and press the **Left** mouse button to draw a dot, a line, or a rectangle in monitor. The histogram will appear in monitor when the left button is released.

5.4 Linear Operations

All the linear operations are pull-right menu items. Select the Linear Operations and pull this item to right, then select the **Filter** menu item streams.

5.4.1 Convolution

Select the **Convolution** menu item from menu item streams, Type in the coefficients of **Data File Name** and select the **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.4.2 Laplacian

Select the **Laplacian** menu item from menu item streams and select the **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.4.3 Median

Select the **Median** menu item from menu item streams and select the Memory Bank(0-7). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.4.4 Sobel

Select the **Sobel** menu item from menu item streams and select the Memory Bank(0-7). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.4.5 Gaussian

Select the **Gaussian** menu item from menu item streams and select the **Memory Bank**$(\mathbf{0-7})$. Type in the standard deviation σ(0.0 - 5.0). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.5 Nonlinear Operations

The nonlinear operations also have pull-right menu items. Select the Nonlinear Operations and pull this item to right, then select the Morphological or Rank menu item to pull right, else select **Spot** or **Reset** item.

5.5.1 Morphological

There are two type of morphological operations: binary and gray scale. The user should binarize an image before execution the binary morphological operations Select the **Binarize** menu item from menu item streams. Type in the value of **High Thresh** and **Low Thresh**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Binary Erosion

Select the **Binary Erosion** menu item from menu item streams, Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Binary Dilation

Select the **Binary Dilation** menu item from menu item streams, Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Binary Open

Select the **Binary Open** menu item from menu item streams. Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Binary Close

Select the **Binary Close** menu item from menu item streams. Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Gray Erosion

Select the **Gray Erosion** menu item from menu item streams. Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Gray Dilation

Select the **Gray Dilation** menu item from menu item streams. Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Gray Open

Select the **Gray Open** menu item from menu item streams. Type in the coefficients of **Data File Name**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Gray Close

Select the **Gray Close** menu item from menu item streams. Type in the coefficients of **Data File Name.** Press the **Accept** button to execute or **Cancel** button to cancel the command.

Thinning

Select the **Thinning** menu item from menu item streams. Type in the coefficients of **Data File Name.** Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.5.2 Rank

The **Rank** menu item is to operate pixels value of two areas.

Max

Select the **Max** menu item from menu item streams, select first Memory Bank(0-7) and select the second **Memory Bank(0 – 7)**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Min

Select the **Min** menu item from menu item streams, select first Memory Bank(0-7) and select the second **Memory Bank(0 – 7)**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

Rank

Select the **Rank** menu item from menu item streams, select first Memory Bank(0-7) and select the second **Memory Bank(0 – 7)**. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.5.3 Spot

Select the **Spot** menu item. Moving the Sun mouse in the canvas area(must be in the range), the Androx mouse cursor will move simultaneously. Select one point and press the **Middle** mouse button to display gray level value in monitor. If middle button is up then gray level value will disappear.

5.5.4 Reset

Select the **Reset** menu item. All the video memory banks will be clear.

5.6 Logical Operations

All the logical operations are pull-right menu item. Select the Logical Operations and pull this item to right, then select the menu item streams.

5.6.1 AND Image

Select the **AND Image** menu item from menu item streams, select first **Memory Bank**$(0-7)$ and select the second **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.6.2 OR Image

Select the **OR Image** menu item from menu item streams, select first Memory Bank(0-7) and select the second **Memory Bank**$(0-7)$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.6.3 XOR Image

Select the **XOR Image** menu item from menu item streams, select first Memory Bank(0-7) and select the second Memory Bank(0-7). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.7 Arithmetic Operations

All the arithmetic operations are pull-right menu item. Select the Arithmetic Operations and pull this item to right, then select the menu item streams.

5.7.1 Add Image

Select the **Add Image** menu item from menu item streams, select first Memory Bank(0-7) and select the second Memory Bank(0-7). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.7.2 Sub Image

Select the **Sub Image** menu item from menu item streams, select first Memory Bank(0-7) and select the second Memory Bank(0-7). Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.7.3 Sum Image

Select the **Sum Image** menu item from menu item streams, select **Memory Bank(0 – 7)** Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.8 Acquisition

All the acquisition is a pull-right menu item. Select the **Acquisition** and pull this item to right, then select the menu item streams.

5.8.1 Digitize Mono

Select the **Digitize Mono** menu item from menu item streams, select the **Sync Video Channel**$(\mathbf{0-3})$ and press the **Acquisition** button to execute or **Stop/Cancel** button to stop or cancel the command.

5.8.2 Digitize RGB

Select the **Digitize RGB** menu item from menu item streams, select the **Sync Video Channel**$(\mathbf{0-3})$ and press the **Acquisition** button to execute or **Stop/Cancel** button to stop or cancel the command.

5.8.3 Data Move

Select the **Data Move** menu item from menu item streams, select source **Memory Bank**$(\mathbf{0-7})$ and select destination **Memory Bank**$(\mathbf{0-7})$. Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.9 Display Bank Image

Select the **Display Bank Image** menu item and select the desired displaying **Memory Bank**$(\mathbf{0-7})$ Press the **Accept** button to execute or **Cancel** button to cancel the command.

5.10 Others

All the others are pull-right menu item. Select the **Others** and pull this item to right, then select the menu item streams.

5.10.1 Thresh

Select the **Thresh** menu item from menu item streams, select gray level value in panel **Slider** then the thresholding will display in monitor. Press the **Return RGB** button to return original image or **Cancel** button to cancel the command.

5.10.2 Lookup Table

Select the **Lookup Table** menu item from menu item streams, select the **Sync Video Channel(0 – 3)** and press the **Accept** button to select or program look up tables or **Cancel** button to cancel the command.

5.10.3 Pseudo Color

Select the **Pseudo Color** menu item from menu item streams. The pseudo color will display in monitor.

5.10.4 Scroll

Select the **Scroll** menu item from menu item streams, select the Horizontal Slider **dx** and select the Vertical Slider **dy** press the **Accept** button to execute or **Cancel** button to cancel the command.

5.10.5 Zoom

Select the **Zoom** menu item from menu item streams, select the **Zoom of x** and the **Zoom of y** then, select the **Length of x** and the **Length of y** press the **Accept** button to execute or **Cancel** button to cancel the command.

5.10.6 Blank

Select the **Blank** menu item. The video memory banks will be clear.

5.11 Help

Select the **Help** menu to popup the help window. Press the right mouse button to select the instruction of help message from the help selection menu items.

5.12 Quit Help

Select the **Quit Help** item to quit help window.

5.13 Quit Program

Select the **Quit Program** item to quit this software.

Chapter 6

Conclusions

A mouse-driven interface is extremely useful for any software performing image processing tasks. A mouse-driven interface for integrating the SunView window system and the Androx image processing system is presented in this thesis. SunView is an object-oriented system, where windows and menus can be created. The user can create mouse-driven interface for his or her application algorithms under the SunView windows system. The Andorx system is a parallel programmable image array processor, which provides four types of image libraries: application library, image processing library, graphics library, and video library. In this thesis, we designed and implemented a mouse-driven interface for performing image processing tasks for the Androx system. The operational instructions are implemented in this thesis and an on line help is also available.

Bibliography

[1] Sun Microsystems, *SunView Programmer's Guide*, Sun Microsystems, May 1988.

[2] Androx Corporation, *ICS-400 Library Programmer's Reference Manual*, Androx, Feb. 1989.

[3] R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison-Wesley, 1987.

[4] P. A. Maragos and R. W. Schafer, "Morphological Skeleton Representation and Coding of Binary Images," *IEEE Trans. ASSP*, Vol. 34, No. 5, Page 1228, October. 1986.

[5] Yiher Chang, *Practical VLSI relations of Morphological Operations*, Master Thesis, Electrical and Computer Engineering Dept. NJIT, Dec. 1990.

[6] A. Rosenfeld and A. C. Kak, *Digital Picture Processing* , Academic Press, 1982.

[7] E. J. Wegman and D. J. Depriest, *Statistical Image Processing and Graphics*, Marcel Dekker, 1986.

[8] N. Zuech, *Applying Machine Vision*, John Wiley & Sons, 1988.

[9] G. Y. Tang, *Digital Image Processing* , World Computer Center in Taiwan, 1986.