General Purpose Intelligent Sensor Interface

JRC Research Report No. 90-08
March 1990

Final Report for D.O. 34

Applications of Artificial Intelligence to Space Station
Task 4

Prepared for
Mr. Tim Crumbley

Information and Electronic Systems Lab
Software and Data Management Division
Systems Software Branch
Marshall Space Flight Center

Prepared by
J.W. McKee

The Johnson Research Center
The University of Alabama in Huntsville
Huntsville, Alabama 35899
(205) 895-6257

Table of Contents

List of Figures

1. Purpose

This final report describes the accomplishments in this phase of Task 4 -- General Purpose Intelligent Sensor Interface task of the Applications of Artificial Intelligence to Space Station project for the period July 1989 through January 1990. The accomplishments in the first phase of this task have been reported in UAH research reports 698 and 728.

2. Research Goals

The long range goal of this research at UAH is to develop an intelligent sensor system that will simplify the design and development of expert systems that use sensors of physical phenomena as a source of input data. This phase of the research has concentrated on the integration of image processing sensors with expert system environments. The anticipated result of this research is the ability to design systems in which the user will not need to be an expert in such areas as image processing algorithms, local area networks, image processor hardware selection or interfacing, or television cameras selection. The user will be able to access data from video sensors through standard expert system statements without any need to know about the sensor hardware or software.

3. Research Approach

This research project was performed in two phases. The first phase concentrated on the following areas:

1. Survey the commercial market for image processing hardware.
2. Select an image processing hardware product that will meet the requirements of this project.
3. Survey and select a LISP expert system and machine that will meet the requirements of this project.
4. Determine the requirements of, and implement, a protocol that will allow the image processing hardware and the expert system to operate together.
5. Determine the requirements of, and implement, the image processing software to perform the low level image processing function with the hardware.
6. Document and demonstrate the system.

The results of the first phase are documented in UAH research reports 698 and 728. The phase one system consisted of an image processing card that plugs into a PC and the LISP environment on a Symbolics computer. These two machines were connected via a protocol that allowed LISP statements to perform image processing functions. Low level image processing functions were written on the PC to interface the image processing hardware with the LISP statements.

The second phase of this research project consists of the same goals as phase one but with an expansion of the domain. The domain has been expanded to include color image processing with both the hardware and expert system being on the same computer.

4. Detail Description of the Second Phase

The specific goals of phase two are as follows:
1. Become a resource to NASA/MSFC in the area of developing application on the Macintosh.
2. Locate and assemble color image processing hardware.
3. Develop software primitives to handle color images.
4. Develop the capabilities to process multi-spectral images (color).
5. Develop a stand alone color image processing system on the Macintosh.
6. Develop the capability to interface image processing hardware with CLIPS on the Macintosh.

The Macintosh was chosen as the host computer because of its architecture and power. CLIPS was chosen as the expert system because it is one of the possible expert systems for use on Space Station Freedom. A version of CLIPS that runs on the Macintosh was obtained. The Data Translation DT2270 was chosen as the image processing hardware because at the time of the start of this project the DT2270 was the only color frame grabber that worked in the Macintosh.

A DT2270 hardware card, color camera, and color monitor were borrowed from other projects to test the software developed on this project. The development environment was SYMANTEC's Think's LightspeedC. The user interface for the stand alone image processing system was developed using the Macintosh Toolbox functions.

The result of this project was the assembly and testing of the hardware necessary to acquire and process color image on the Macintosh. Also a base line, stand alone color image processing environment was developed. Its user interface is very similar to the customary Macintosh menu environment. The majority of the effort on this project was expended on developing this user interface using the Toolbox. But this has resulted in the capability to develop other applications for NASA/MSFC on the Macintosh. The remainder of this report goes into more detail about design of the stand alone image processor system, how to use the Toolbox to create the user interface, and interfacing the image processing functions with CLIPS.

4.1 Using the Macintosh Toolbox to create User Interfaces

The Macintosh has a set of callable functions that generate the standard user interfaces that one generally associates with Macintosh applications. These functions are grouped together in what is called the Toolbox. The definitive description of all the Toolbox functions along with a complete description of the structure of the Macintosh line of computers is *Inside Macintosh* vol 1-5 [1], the set of reference manuals produced by Apple. These books are written at the reference level. Therefore if one does not already understand the basics of programming in the Toolbox, these are very difficult to understand. They do not start at the top and work down. All the functions are described in PASCAL.

One level up would be references like *Macintosh Revealed, Unlocking the Toolbox* [2] and *Macintosh Revealed, Programming with the Toolbox* [3]. These references divide programming the Toolbox into logical topics, gives an introduction to each topic, and present the various callable functions. Unfortunately they do not give examples on how to put all of the functions together to make working programs. All the functions are described in PASCAL .

A very good reference that starts at the top is *Macintosh Programming Primer* [4]. This reference explains how to program using the Toolbox by going through examples, written in C, that perform various menu functions. This is probably where someone new to the Macintosh should start. The person could use the lower level references for more detailed information about the various Toolbox functions.

4.2 Image Processing Hardware Description

A block diagram of the Data Translation DT2270 frame grabber card is shown in figure 1. As can be seen this card brings in color video in NTSC format, converts the signal to RGB format, digitizes the signal and stores the result in the hardware frame buffer. The video output comes directly from the hardware frame buffer, is converted to analog RGB signals and then converted to NTSC output.

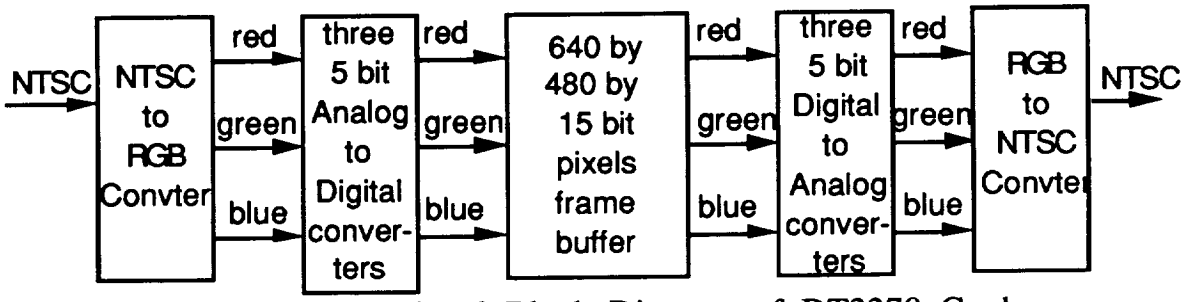

Figure 1 Functional Block Diagram of DT2270 Card

A simplified block diagram of the DT2270 is shown in figure 2. Figure 3 shows the architecture developed in this project to do color image processing using the DT2270. All the added blocks are performed in software on the Macintosh. Three software color frame buffers have been added. Two software color process maps have been added. The RGB/IHS conversion is done in software.

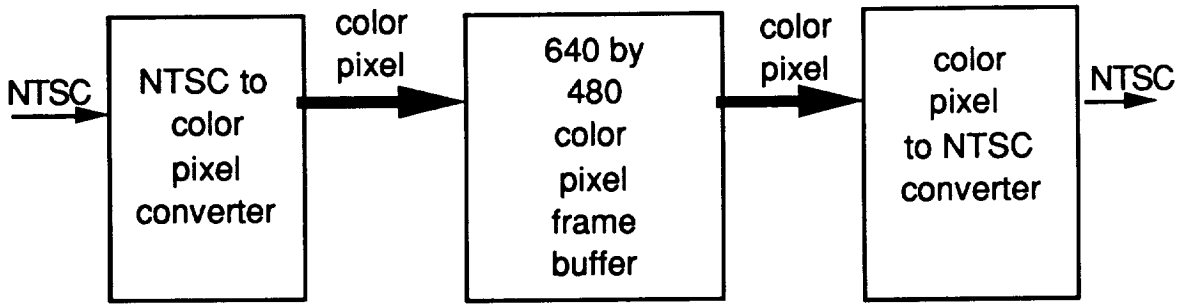

Figure 2 Simplified Block Diagram of DT2270

The extra three frame buffers allow the user to move data around, capture and work on more that one image at a time, decompose color images into their components, and reassemble them. The two process maps allow the user to create whatever mapping functions are needed and to be able to pass the pixels through the maps. This is the minimum system to do any color image processing. Other functions can easily be added to the system.

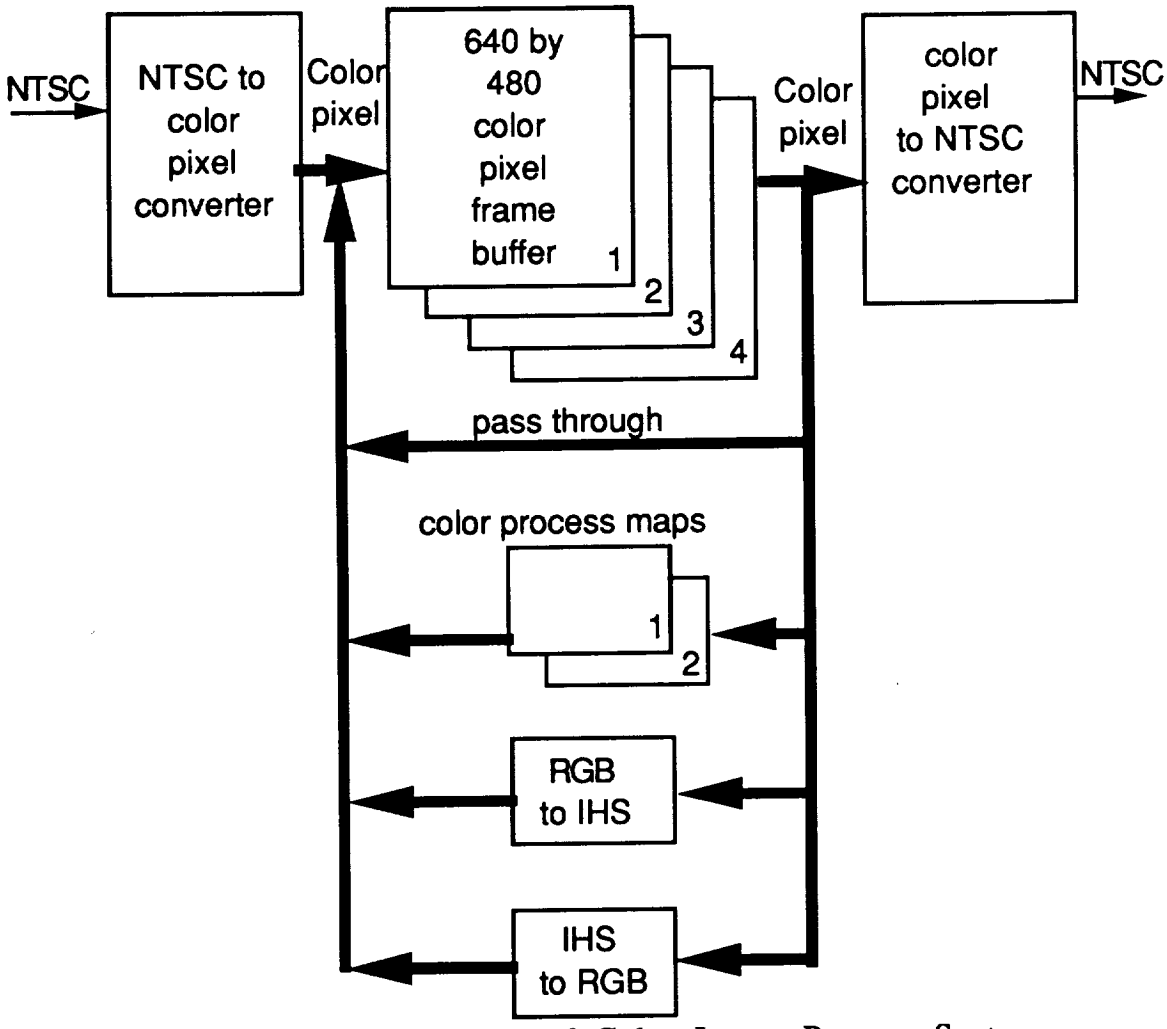

Figure 3 Block Diagram of Color Image Process System

4.3 Image Processing Software Flow Charts and Menus

This section presents a description of the color image processing system software in the form of a structured high level flow chart and the corresponding menus. The source code listings for the user menu interface using the Toolbox is in Appendix A. The source code listing for all the image processing functions described in the following section is in Appendix C. Example rules to invoke the image processing functions from CLIPS are listed in Appendix E. Figure 4 shows the top level of the flow chart. The software is initialized and the menu is presented and serviced. Figure 5 shows how the top level menu appears on the screen of the Macintosh. Across the top are standard drop down menus.

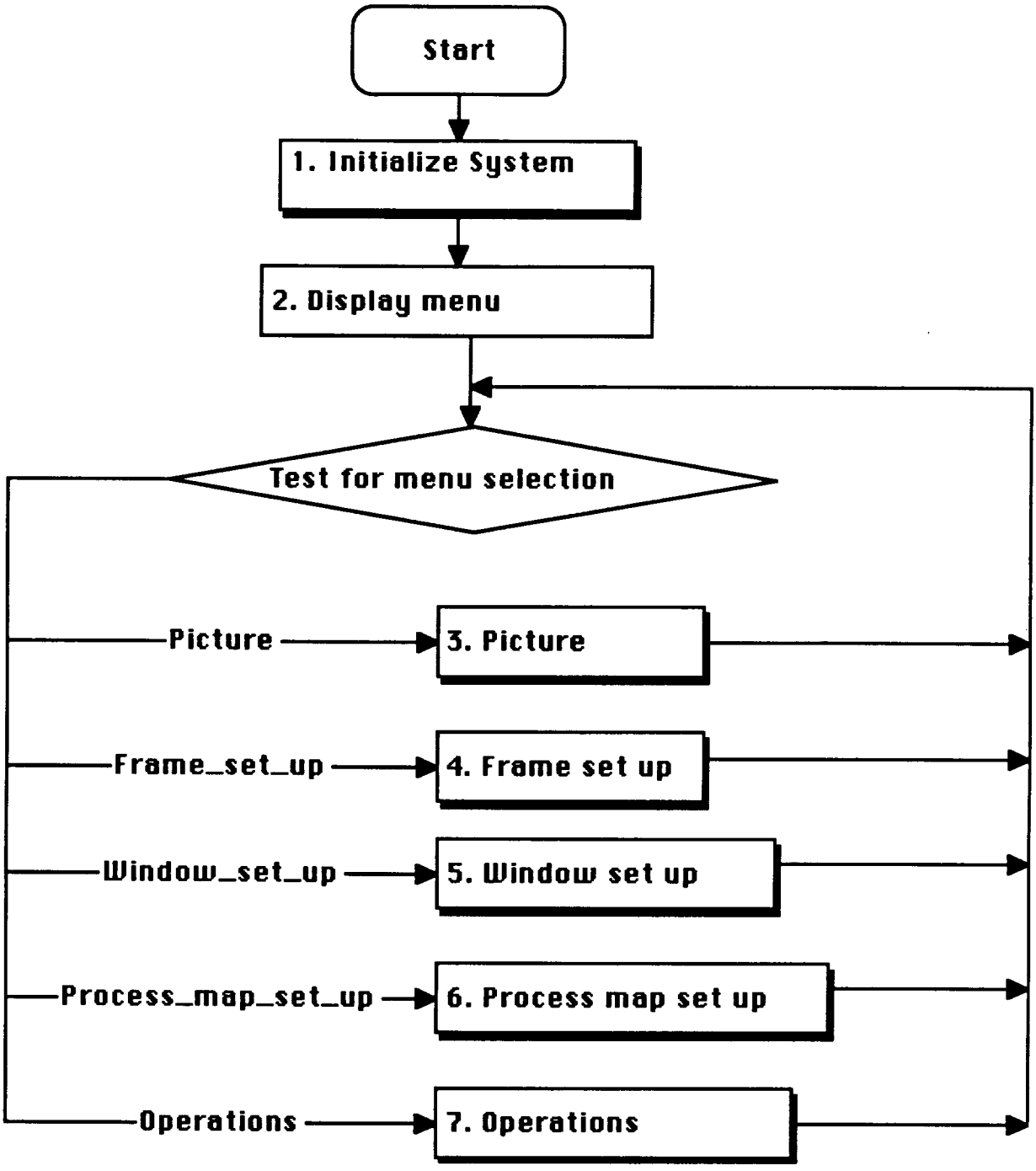

Figure 4 Top Level Flow Chart of the Color Image Processor Software

The Picture menu is used to select image input and output. The Frame menu is used to set up the logical frame operations. The Windows menu is used to set the size of the windows or region of interest to be used with the logical frames. The Maps menu is used to

place data in the color process maps. And Operations menu is used to select which image processing function to perform.

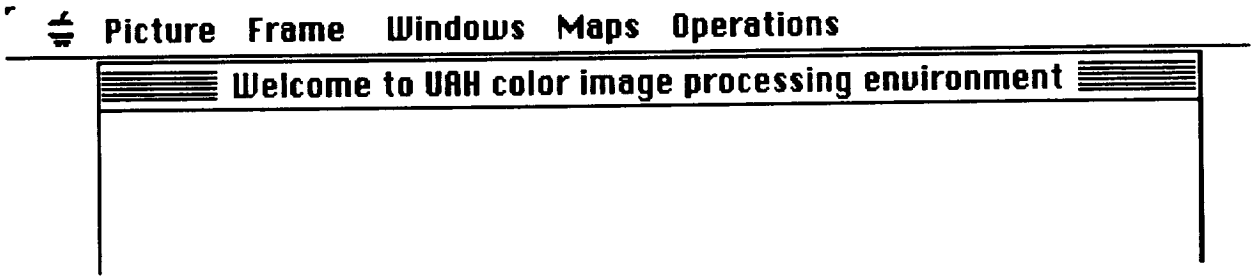

Figure 5 Start up menu selections

Figure 6 shows the various function that are called to initialize the software. The functions toolBoxInit, windowInit, menuBarInit, and setUpDragRect are functions to initialize the Toolbox. The function imageInit initializes the image processing hardware. The function frameInit creates the three added color frame buffers and defines their sizes. The function mapInit creates the two color process maps and the color domain conversion maps. The function imageWindowInit set up the initial window or regions of interest.

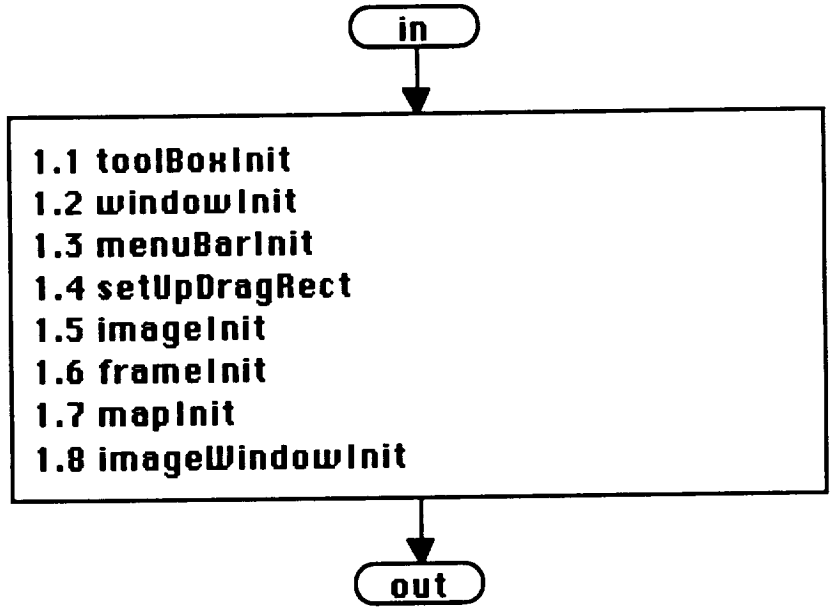

Figure 6 Block 1 -- Initialize System

Figure 7 shows the flow chart for the drop down Picture menu. And figure 8 shows how the menu looks on the screen. There are five selections: set the hardware to display a live image, set the hardware to

snap an image into the hardware frame buffer, store a image into a file, retrieve an image from a file, and quit the program. The store and retrieve menu selections have not been implemented yet.

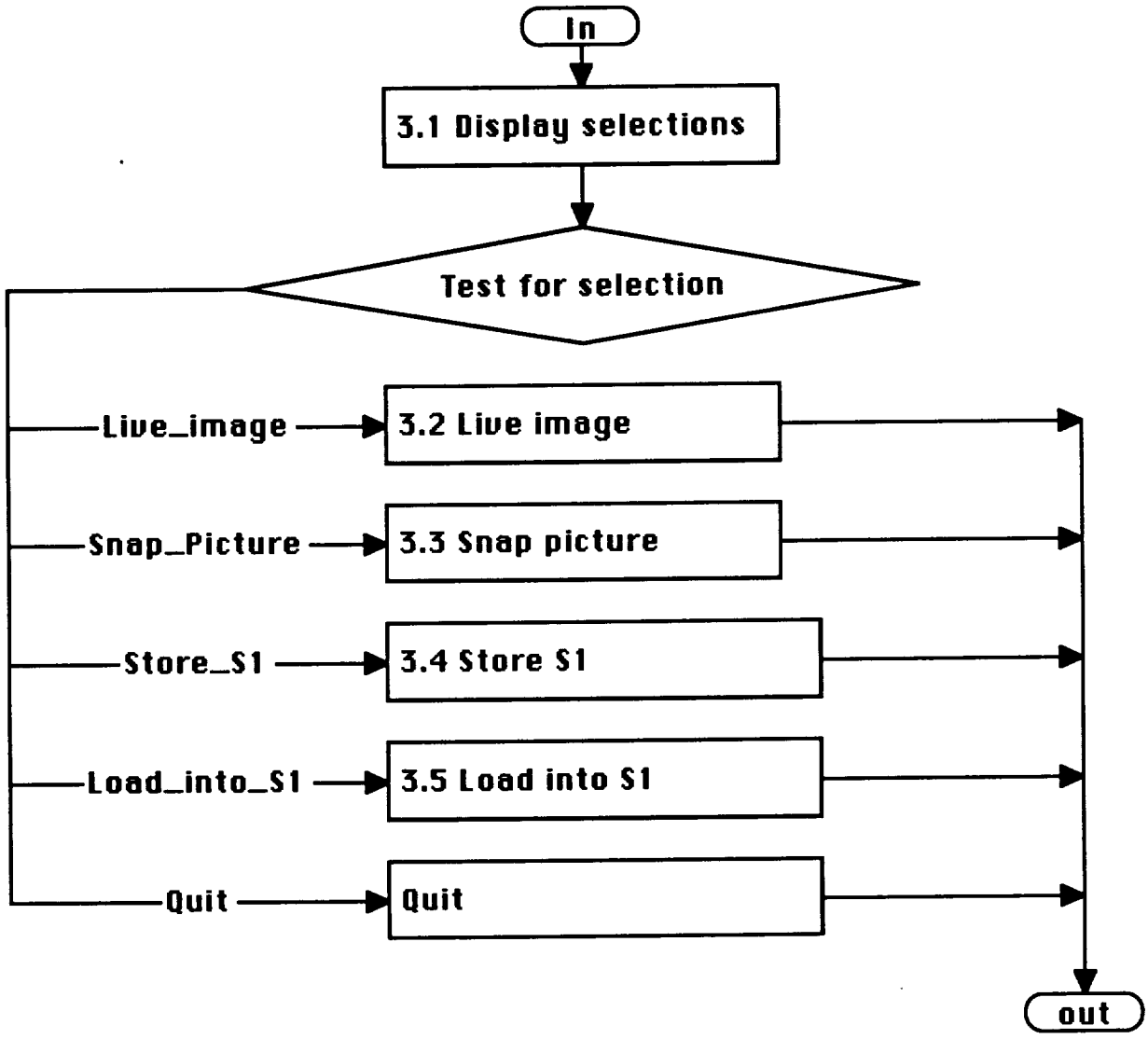

Figure 7 Block 3 -- Picture Menu

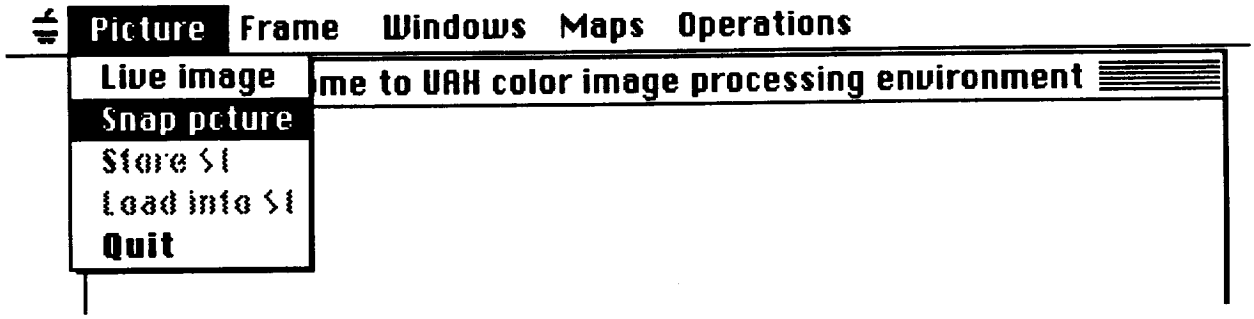

Figure 8 Menu selections to acquire an image

Figure 9 shows the flow chart for the Frame Set Up Menu. There are two classes of image processing operations: unary and binary. Unary operations take place between a source frame and a destination frame. Binary image processing operations take place between two source frames and a destination frame. This color image processing system was implemented with logical image processing frames which allows the user to map actual physical frames into logical frames. This allows the user to use any of the four physical frames as any logical frame. This menu allows the user to assign any of the four physical frames as logical source 1, source 2, and destination frames. Figure 10 shows the dialog box used to make the logical assignments

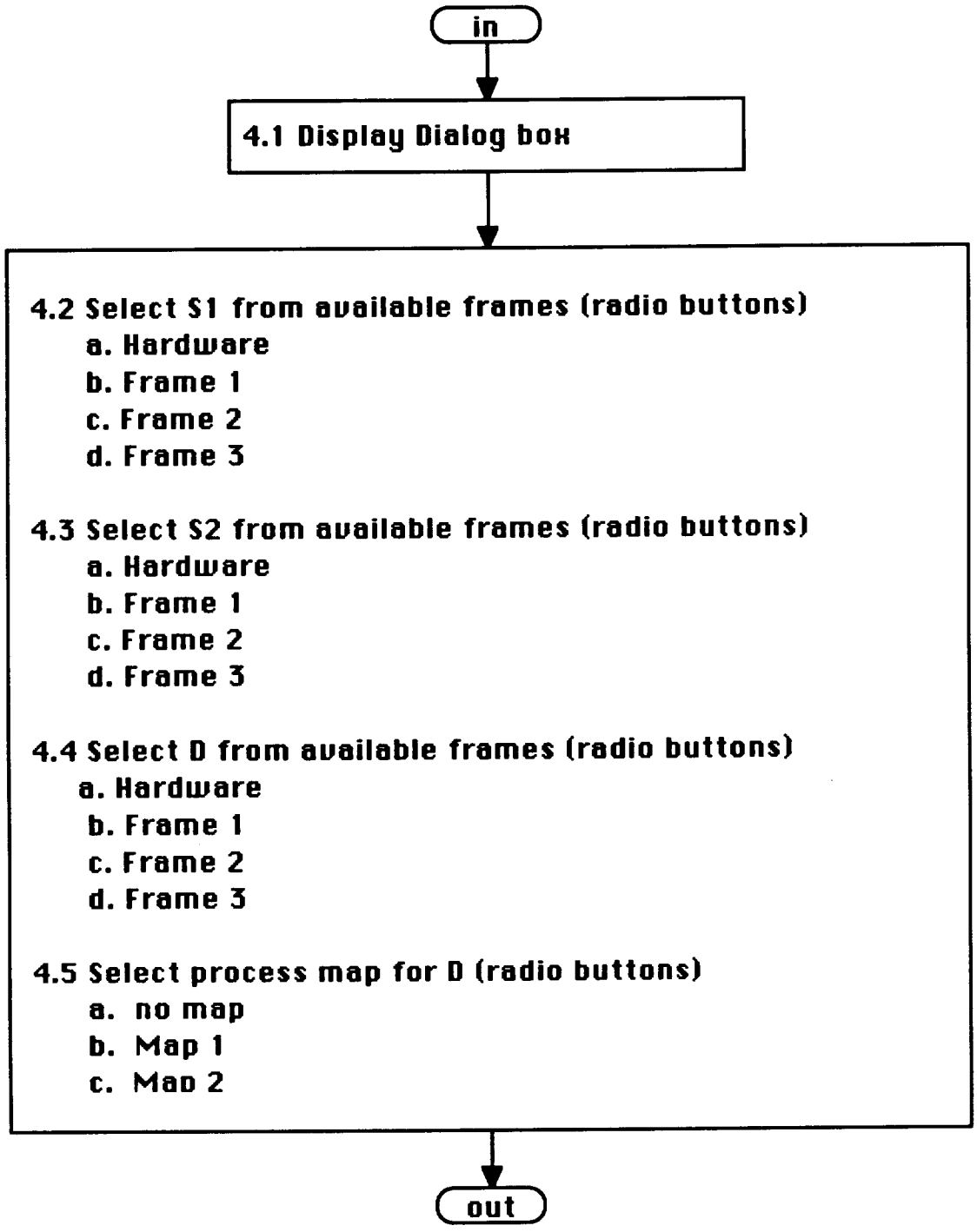

Figure 9 Block 4 -- Frame Set Up Menu

The user selects one of the four radio buttons under Source S1 for the assignment of a physical frame to the logical frame S1. Similarly the S2 and the Destination frames can be assigned. A physical frame

can be assigned to more than one logical frame. The user also selects which color process map to use by selecting the corresponding radio button. The color process map allows the user to transform an image pixel by pixel, e.g., threshold the image, intensity scale the image, select portions of an image by their intensity values, etc.

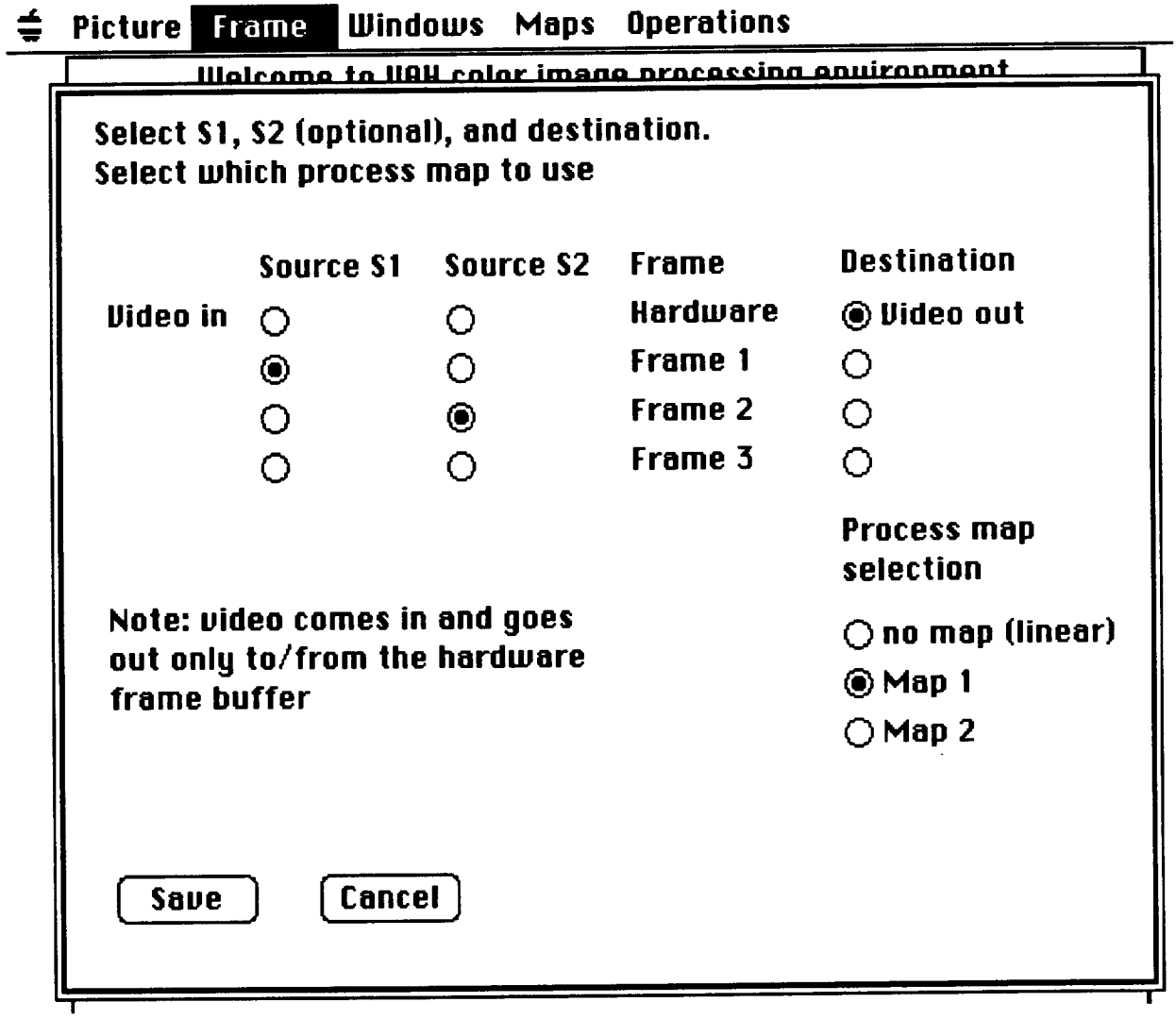

Figure 10 Menu to set up Source 1, Source 2, Destination frames, and process map selection

Figure 11 shows the flow chart to set up the process window or region of interest for each logical frame. The user selects the logical frame, i.e., S1, S2, or D, and then the dialog box shown in Figure 12 appears.

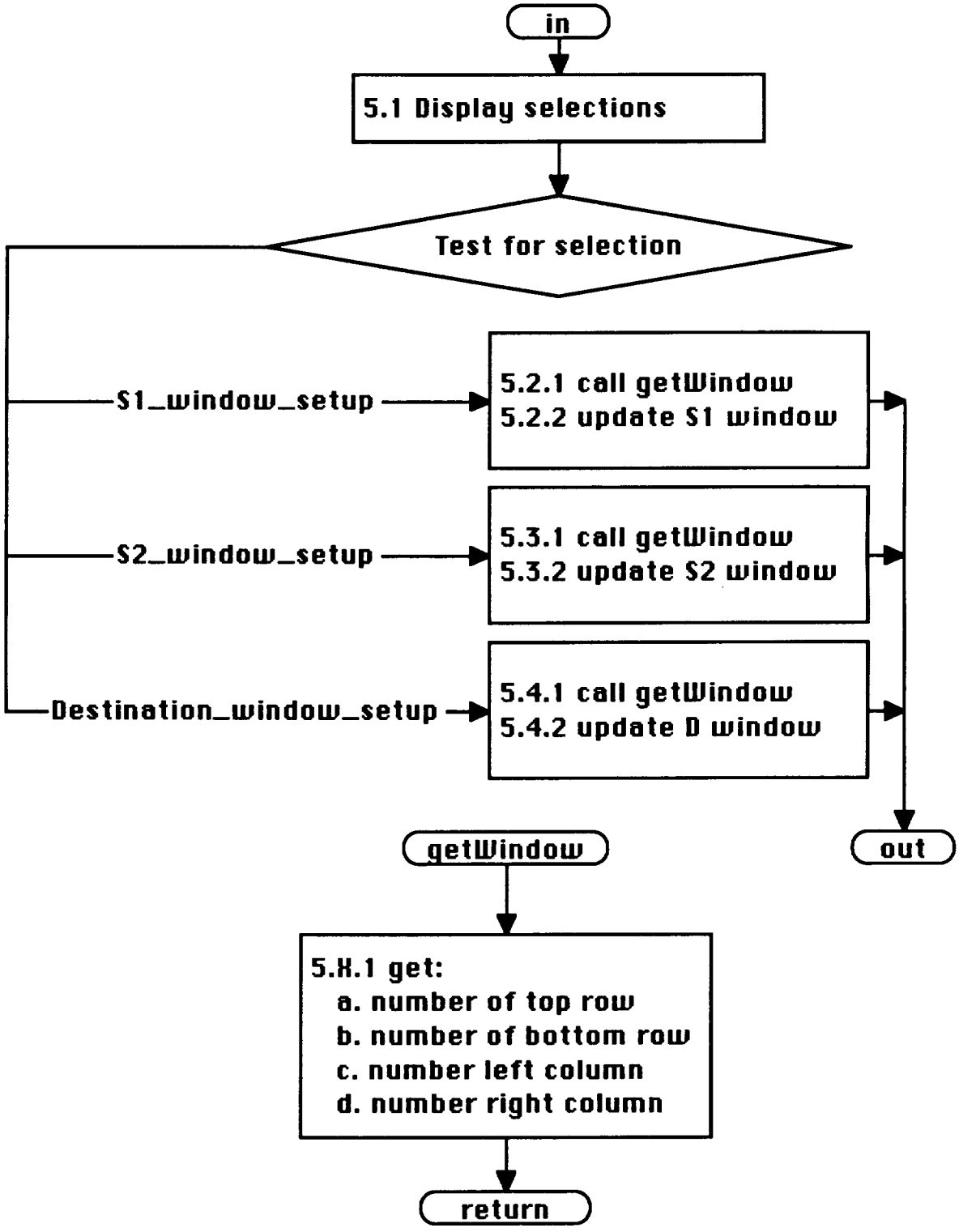

Figure 11 Block 5 -- Process map set up

The user sets up the window for a logical frame by entering the row and column values. These values are checked to insure that they are in range. If the user enters out of range data, it is set to the corresponding largest or smallest in range value. The use of windows allows the image processing to be done only on the portion of the image that is of interest. This can greatly increase the speed of doing image processing.

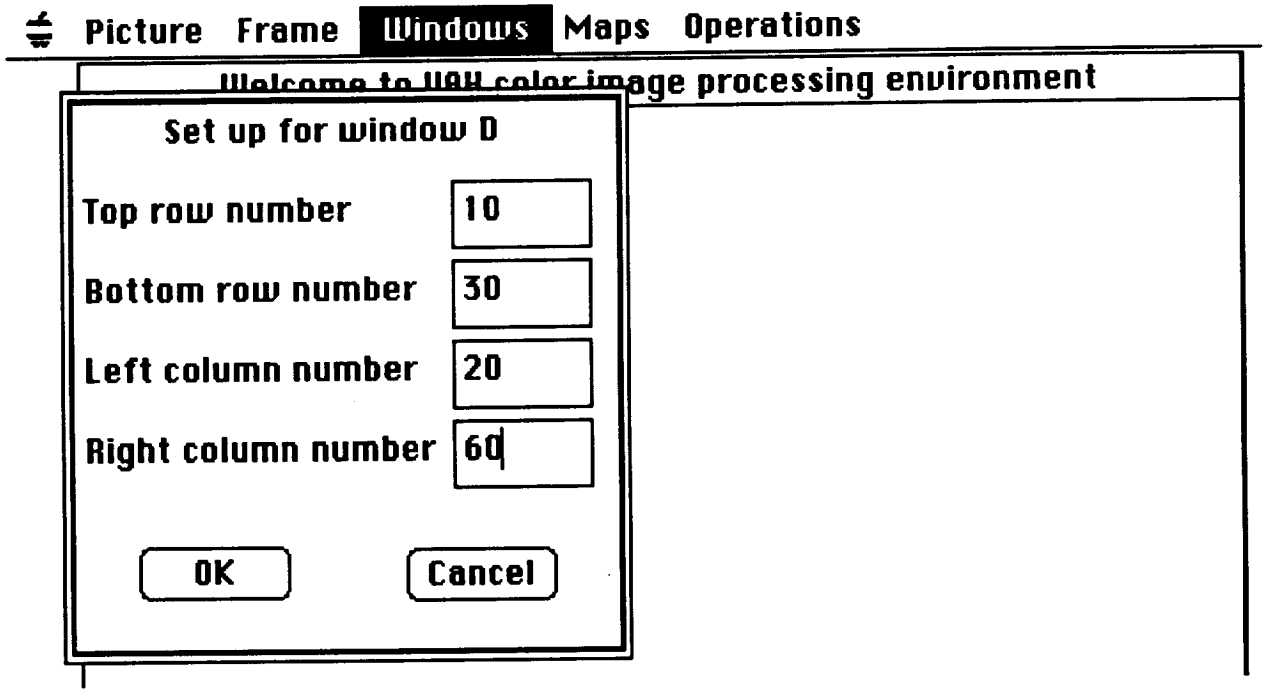

Figure 12 Menu selections to set up the region of interest windows

Figure 13 shows the flow chart to enter values into either of the color process maps. Figure 14 shows the dialog box used to enter the selections.

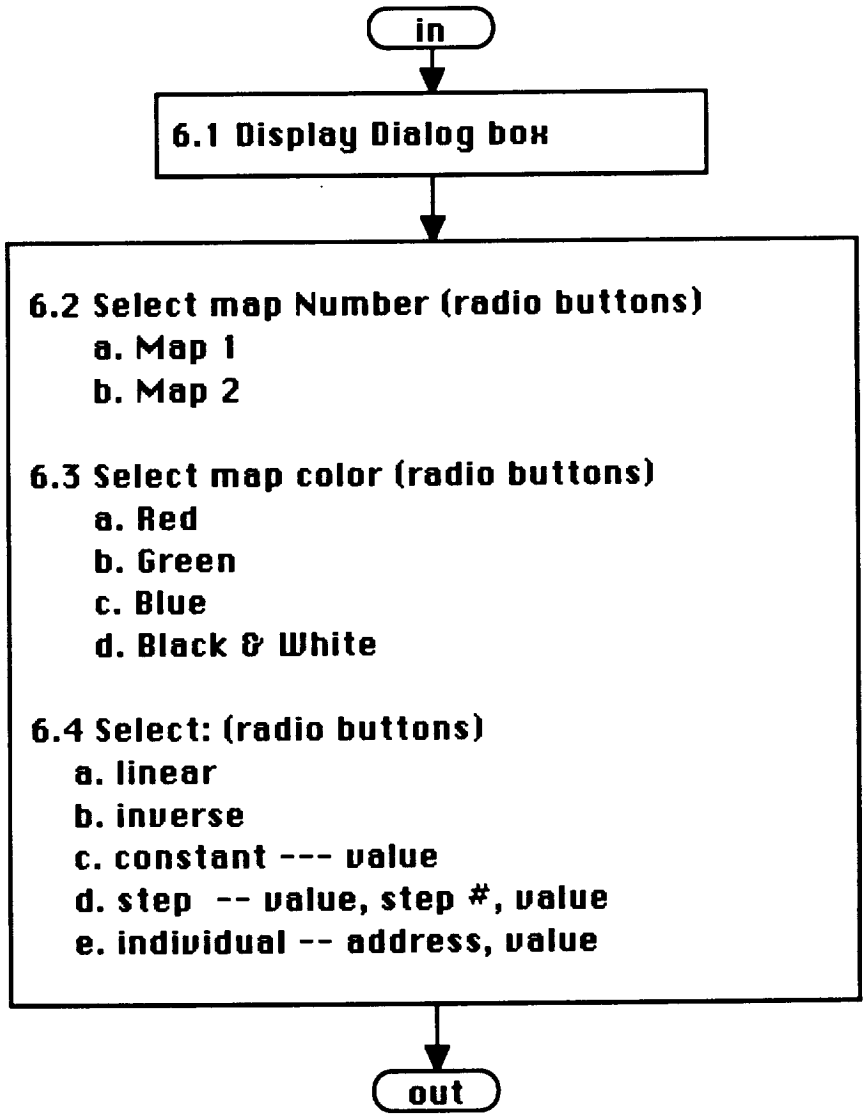

Figure 13 Block 5 -- Window set up

The user selects either Map 1 or Map 2 by clicking one of the radio buttons. Each color map has three parts or sub-maps: the red map, the blue map, and the green map. The user selects one of the sub-maps. Then the user selects an operation on the sub-map. The Linear operation sets the sub-map to linear, i.e., the intensity in will be the intensity out. The Inverse operation sets the sub-map to invert the intensity, i.e., the intensity out is the inverse of the intensity in -- produces a negative image. The constant selection allows the user to set the sub-map to a constant value for all inputs; this is used to threshold images. The Step selection allows the user to set one constant value up

to the step and another value after the step. The Individual selection allows the user to set any location to any value.

The Black & White selection allows the user to set the map up to process only one intensity at a time. This map is used when the user decomposes the color pixels into its individual components and then works on an individual component.

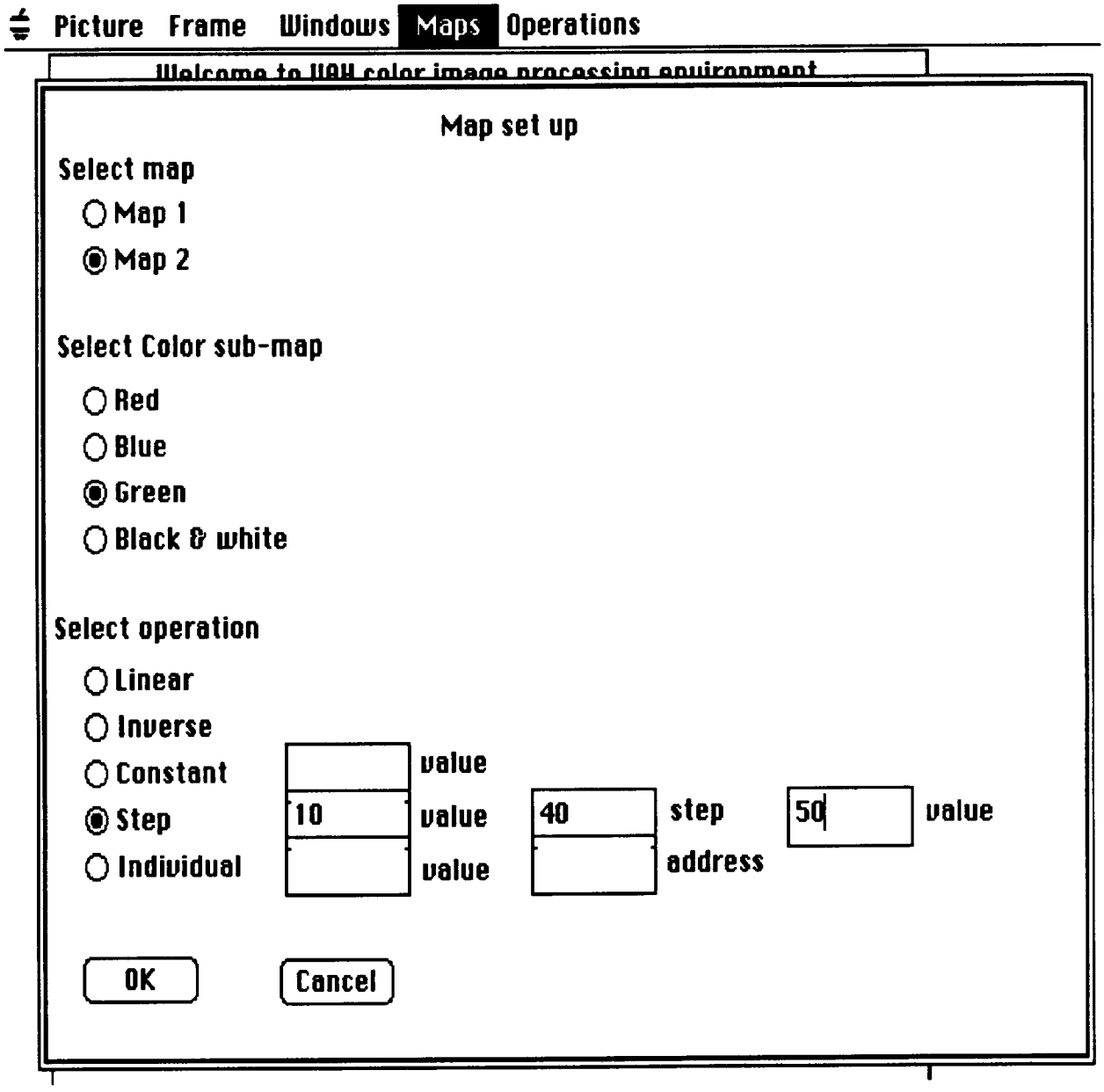

Figure 14 Menu selections to set up the two color process maps

Figure 15 shows the flow chart for the operations menu. Figure 16 is the dialog box used to select an operation.

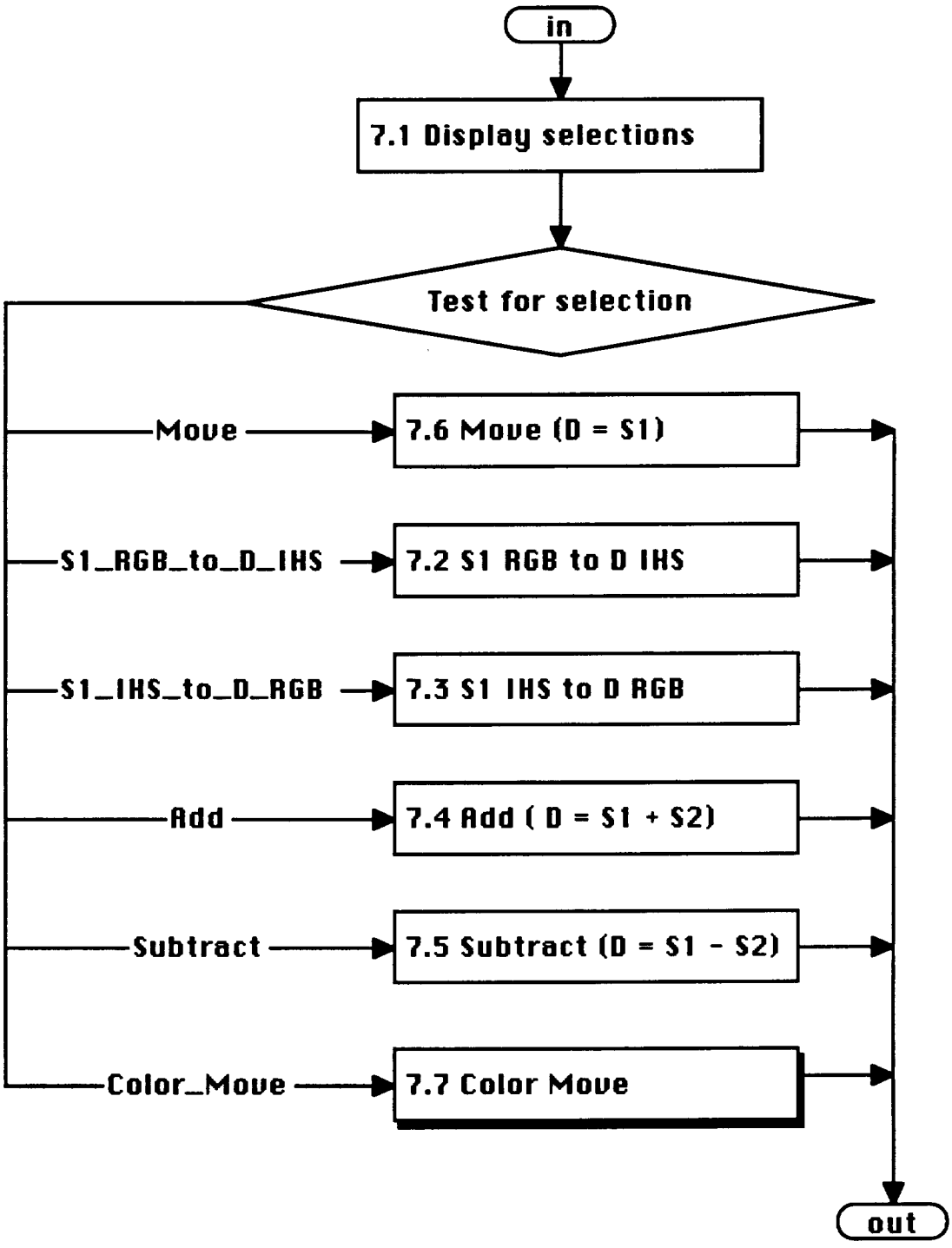

Figure 15 Block 7 -- Operations

At present there are five main image processing operations. The user can select to move an image from one logical frame to another by the Move selection. The pixels in the window of S1 are moved into the window of D. If a process map had been selected, the pixels are passed through the map. The S1 RGB to IHS selection translate the RGB color coordinates of each pixel in the window of S1 to IHS color coordinates and place the pixels in the window in D. The S1 IHS to RGB selection does the inverse. The Add selection adds the corresponding pixels in the window of S1 to the pixels in the window of S2, divides by 2 (to prevent overflow), and places the sum in the window of D. Subtract selection does a pixel by pixel subtraction placing the absolute value in the window of D, similar to the Add selection. The Color Move selection is a hierarchical menu which allows the user to select pixel decomposition operations.

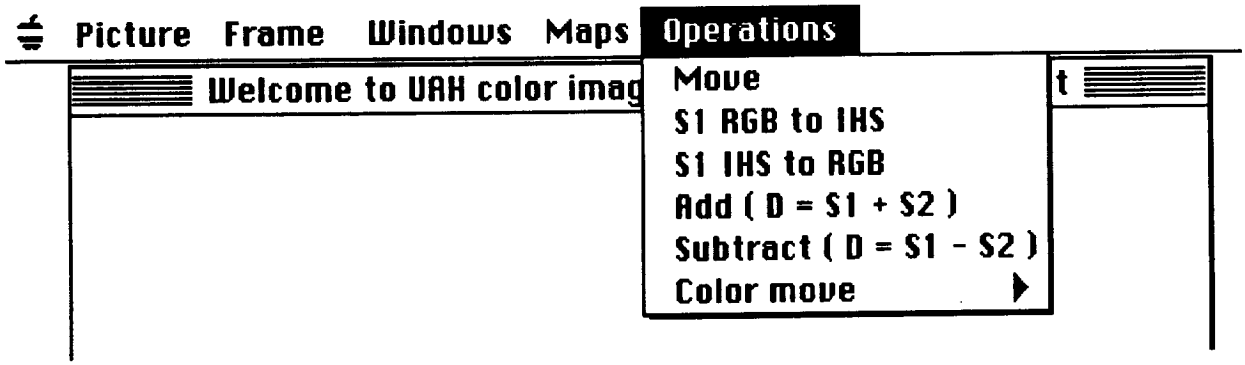

Figure 16 Menu selections to do image processing

Figure 17 shows the flow chart for the hierarchical menu and figure 18 shows the menu selections.

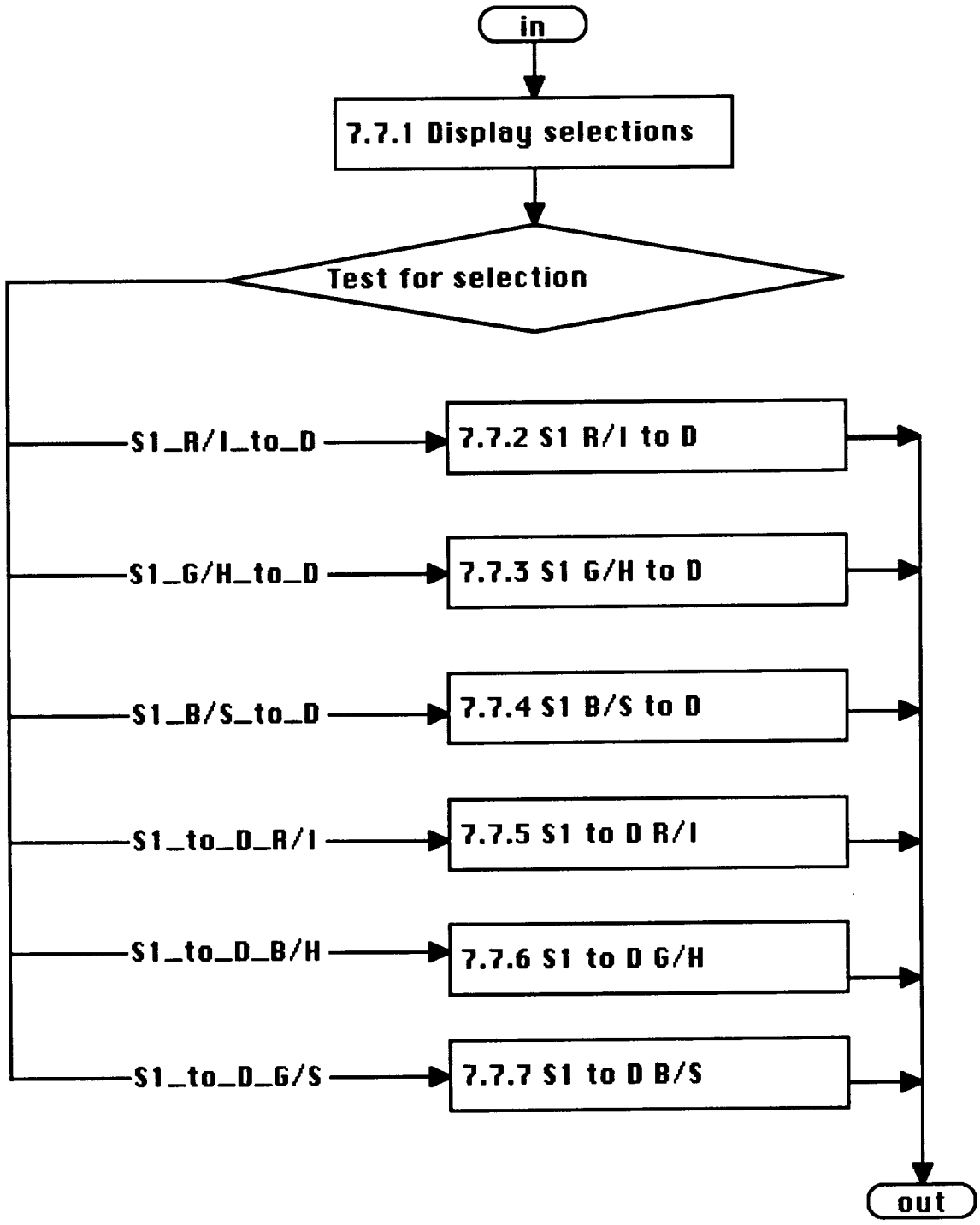

Figure 17 Block 7.7 Color Move

There are three selections that will decompose a color pixel and three selections will rebuild color pixels. S1 Red/I to D, S1 Green/H to D, and S1 Blue/S to D work in a similar way to decompose color pixels into monochrome pixels. S1 Red/I to D will move the red intensity of each

color pixel in the window of S1 to the lower portion of a pixel, set the upper portion to 0 and place the resultant monochrome pixel in the window of D. S1 to D Red/I, S1 to D Green/H, and S1 to D Blue/S work in a similar way to recombine monochrome pixels into color pixels. S1 to D Red/I will insert the monochrome pixels in the window of S1 into the corresponding color pixels in the window of D.

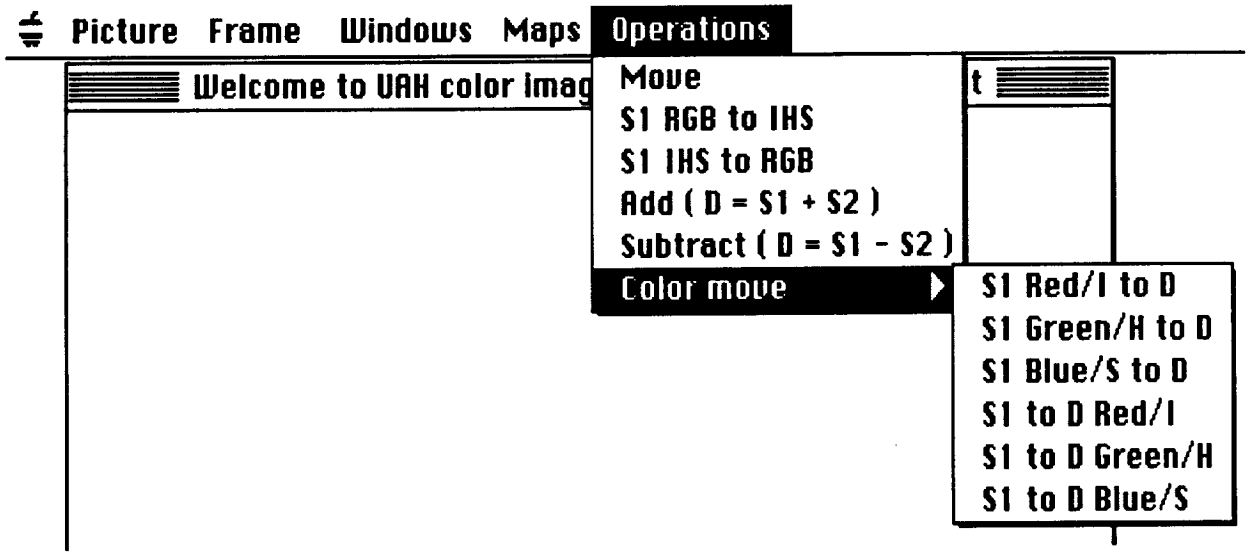

Figure 18 Menu selections to decompose and reassemble color pixels

Appendix A Source code for user menu interface

```
/*****************************************************************************/
/***********                    ColorImage.c                        **********/
/*****************************************************************************/

/******* last modified 14 march 1990                                 *****/

/****** this is the main of the code, also includes the menu code      */

#include "ColorStructs.h"

/* definition of global defines                                      */

#define     NIL_POINTER                 0L
#define     MOVE_TO_FRONT               -1L
#define     APPLE_MENU_ID               500
#define COLOR_MOVE_ID                   100

/*  definition of global parameters                                  */

Rect        gDragRect;
MenuHandle      gAppleMenu;
Boolean         gDone, gWNEImplemented;
EventRecord     gTheEvent;
COLOR_SYSTEM colorSystem;

/**********          main                        *********/

main()
{

toolBoxInit();                                          /* 1.1   */
windowInit();                                           /* 1.2   */
menuBarInit();                                          /* 1.3   */
setUpDragRect();                                        /* 1.4   */
imageInit();                                            /* 1.5   */
```

```
frameInit(&colorSystem);                    /* 1.6   */
mapInit(&colorSystem);                      /* 1.7   */
imageWindowInit(&colorSystem);              /* 1.8   */

mainLoop();
}


/*********          1.1 ToolBoxInit                 *********/

#define     REMOVE_ALL_EVENTS       0

toolBoxInit()
{

InitGraf( &thePort );
InitFonts();
FlushEvents( everyEvent, REMOVE_ALL_EVENTS );
InitWindows();
InitMenus();
TEInit();
InitDialogs( NIL_POINTER );
InitCursor();
return;
}


/**********         1.2 windowInit                  *********/

#define WINDOW_RES_ID       400

windowInit()
{

WindowPtr       Window;

Window =
GetNewWindow(WINDOW_RES_ID,NIL_POINTER,MOVE_TO_FRONT);
SetPort(Window);
ShowWindow(Window);

return;
}
```

```
/**********      1.3     menuBarInit                              *******/

#define  MENU_RES_ID              500
#define NOT_A_NORMAL_MENU    -1

menuBarInit()
{
MenuHandle   colorMove;
Handle           MenuBar;

MenuBar = GetNewMBar( MENU_RES_ID );
SetMenuBar( MenuBar );
/* add all of apple choices to current apple menu */
gAppleMenu = GetMHandle( APPLE_MENU_ID );
AddResMenu( gAppleMenu, 'DRVR' );
/* add hierarchical menu to color move selection */
colorMove = GetMenu( COLOR_MOVE_ID );
InsertMenu ( colorMove, NOT_A_NORMAL_MENU );
DrawMenuBar();
return;
}


/**********          1.4 setUpDragRect                          *********/

#define DRAG_THRESHOLD                    100

setUpDragRect()
{

gDragRect  =  screenBits.bounds;
gDragRect.left += DRAG_THRESHOLD;
gDragRect.right -= DRAG_THRESHOLD;
gDragRect.bottom -= DRAG_THRESHOLD;
return;
}

/**********          2.0  mainLoop                              **********/

#define     WNE_TRAP_NUM          0x60
#define     UNIMPL_TRAP_NUM       0x9F
```

```
mainLoop()
{
gDone = FALSE;
gWNEImplemented = ( NGetTrapAddress( WNE_TRAP_NUM, ToolTrap )
!=
      NGetTrapAddress( UNIMPL_TRAP_NUM, ToolTrap ) );

while ( gDone == FALSE )
      {
      handleEvent();
      }
/* end of program  */
}


/**********          2.1   handleEvent                    *********/

#define MIN_SLEEP              0L
#define    NIL_MOUSE_REGION    0L

handleEvent()
{
char theChar;

if ( gWNEImplemented )
      WaitNextEvent( everyEvent, &gTheEvent, MIN_SLEEP,
NIL_MOUSE_REGION );
else
      {
      SystemTask();
      GetNextEvent( everyEvent, &gTheEvent );
      }

switch ( gTheEvent.what )
      {
      case nullEvent:
      /*    handleNull();     */
            break;
      case mouseDown:
            handleMouseDown();          /* 2.1.2    */
            break;
      case keyDown:
```

```
        case autoKey:
                theChar = gTheEvent.message & charCodeMask;
                if (( gTheEvent.modifiers & cmdKey ) != 0)
                        {
                        handleMenuChoice( MenuKey( theChar ) );
                                /* 2.1.2.1    */
                        }
                        break;
        case updateEvt:
                BeginUpdate( gTheEvent.message );
                EndUpdate( gTheEvent.message );
                break;

        }
return;
}


/**********          2.1.2 handleMouseDown                  *******/

handleMouseDown()
{
WindowPtr       whichWindow;
short int   thePart;
long int    menuChoice, windSize;

thePart = FindWindow( gTheEvent.where, &whichWindow );
switch ( thePart )
        {
        case inMenuBar:
                menuChoice = MenuSelect( gTheEvent.where );
                handleMenuChoice( menuChoice );         /* 2.1.2.1   */
                break;
        case inSysWindow:
                SystemClick( &gTheEvent, whichWindow );
                break;
        case inDrag:
                DragWindow( whichWindow, gTheEvent.where,
                &gDragRect );
                break;
        case inGoAway:
                DisposeWindow( whichWindow );
                break;
```

```
        }
return;
}

/**********          2.1.2.1 handleMenuChoice                ********/


#define     PICTURE_MENU_ID         501
#define     FRAME_MENU_ID           502
#define     WINDOW_MENU_ID          504
#define     MAP_MENU_ID             505
#define     OPERATIONS_MENU_ID      506

handleMenuChoice( menuChoice )
long int    menuChoice;
{
int   theMenu;
int   theItem;

if ( menuChoice != 0 )
      {
      theMenu = HiWord( menuChoice );
      theItem = LoWord( menuChoice );
      switch ( theMenu )
            {
            case APPLE_MENU_ID :
                  handleAppleChoice(theItem);       /*  2.1.2.1.1 */
                  break;
            case PICTURE_MENU_ID :
                  handlePictureChoice(theItem);     /*  2.1.2.1.2 */
                  break;
            case FRAME_MENU_ID :
                  handleFrameChoice(theItem);       /*  2.1.2.1.3 */
                  break;
            case WINDOW_MENU_ID :
                  handleWindowChoice(theItem);      /*  2.1.2.1.4 */
                  break;
            case MAP_MENU_ID :
                  handleMapChoice(theItem);         /*  2.1.2.1.5 */

                  break;
            case OPERATIONS_MENU_ID :
                  handleOperationsChoice(theItem); /*  2.1.2.1.6 */
```

```
                break;
            case COLOR_MOVE_ID :
                handleColorMoveChoice(theItem); /*   2.1.2.1.7 */
                break;
            }
        HiliteMenu( 0 );
        }
return;
}

/***********          2.1.2.1.1   handleAppleChoice          *******/

#define ABOUT_ITEM_NUM          1
#define ABOUT_ITEM_DIALOG       700

handleAppleChoice(theItem)
int  theItem;
{
Str255  accName;
int  accNumber;

switch(theItem)
  {
   case ABOUT_ITEM_NUM:
        NoteAlert(ABOUT_ITEM_DIALOG,NIL_POINTER);
        break;
    default:
          GetItem(gAppleMenu,theItem,accName);
        break;
  }
return;
}

/***********          2.1.2.1.2   handlePictureChoice
***********/

#define LIVE_IMAGE_NUM          1
#define SNAP_PICTURE_NUM        2
#define STORE_S1_NUM            3
#define LOAD_S1_NUM             4
#define QUIT_ITEM_NUM           5

handlePictureChoice(theItem)
```

```
int  theItem;
{
switch(theItem)
      {
      case LIVE_IMAGE_NUM:
            imageLive();
      break;
      case SNAP_PICTURE_NUM:
            imageSnap();
      break;
      case STORE_S1_NUM:

      break;
   case LOAD_S1_NUM:

      break;
      case QUIT_ITEM_NUM:
            gDone = TRUE;
      break;
      }
return;
}


/***********          2.1.2.1.3   handleFrameChoice          ********/

#define FRAME_SET_UP_NUM                  1

handleFrameChoice(theItem)
int  theItem;
{
switch(theItem)
      {
      case FRAME_SET_UP_NUM:
            handleFrameSetUp();                  /* 2.1.2.1.3.1  */
      break;
      }
return;
}

/***********          2.1.2.1.3.1   handleFrameSetUp          ***********/

#define FRAME_DIALOG_RES_ID               600
```

```
#define FRAME_ON                    1
#define FRAME_OFF                   0

#define HARDWARE                    0
#define  FRAME1                     1
#define  FRAME2                     2
#define  FRAME3                     3

#define NO_MAP                      0
#define  MAP1                       1
#define  MAP2                       2


#define FRAME_SAVE_BUTTON           1
#define FRAME_CANCEL_BUTTON         2
#define FRAME_S1_HARD_RADIO         3
#define FRAME_S1_F1_RADIO           4
#define FRAME_S1_F2_RADIO           5
#define    FRAME_S1_F3_RADIO        6

#define FRAME_S2_HARD_RADIO         7
#define FRAME_S2_F1_RADIO           8
#define FRAME_S2_F2_RADIO           9
#define    FRAME_S2_F3_RADIO        10

#define FRAME_D_HARD_RADIO          11
#define FRAME_D_F1_RADIO            12
#define FRAME_D_F2_RADIO            13
#define    FRAME_D_F3_RADIO         14

#define FRAME_NO_MAP_RADIO          15
#define FRAME_MAP1_RADIO            16
#define FRAME_MAP2_RADIO            17


handleFrameSetUp()
{

short           source1, source2, destination, map;
int             itemHit, dialogDone = FALSE;
int             itemType;
Rect            itemRect;
```

```
Handle          itemHandle;
DialogPtr       frameDialog;


frameDialog = GetNewDialog( FRAME_DIALOG_RES_ID, NIL_POINTER,
MOVE_TO_FRONT );

/* set radio buttons to proper state */
source1 = colorSystem.Source1.Frame;
source2 = colorSystem.Source2.Frame;
destination = colorSystem.Destination.Frame;
map = colorSystem.ProcessMap;

handleSource1(frameDialog, source1);          /* 2.1.2.1.3.1.1  */
handleSource2(frameDialog, source2);          /* 2.1.2.1.3.1.2  */
handleDestination(frameDialog, destination);  /* 2.1.2.1.3.1.3  */
handleMap(frameDialog, map);                  /* 2.1.2.1.3.1.4  */
ShowWindow( frameDialog );

while ( dialogDone == FALSE )
      {
      ModalDialog( NIL_POINTER, &itemHit );
      switch ( itemHit )
            {
            case FRAME_SAVE_BUTTON:
                  HideWindow( frameDialog );
                  dialogDone = TRUE;
                  colorSystem.Source1.Frame = source1;
                  colorSystem.Source2.Frame = source2;
                  colorSystem.Destination.Frame = destination;
                  colorSystem.ProcessMap = map;
                  break;
            case FRAME_CANCEL_BUTTON:
                  HideWindow( frameDialog );
                  dialogDone = TRUE;
                  break;

            case FRAME_S1_HARD_RADIO:
                  source1 = HARDWARE;
                  handleSource1(frameDialog, source1);
                  /* 2.1.2.1.3.1.1  */
                  break;
            case FRAME_S1_F1_RADIO:
```

```
            source1 = FRAME1;
            handleSource1(frameDialog,  source1);
            break;
        case FRAME_S1_F2_RADIO:
            source1 = FRAME2;
            handleSource1(frameDialog,  source1);
            break;
        case FRAME_S1_F3_RADIO:
            source1 = FRAME3;
            handleSource1(frameDialog,  source1);
            break;

        case FRAME_S2_HARD_RADIO:
            source2 = HARDWARE;
            handleSource2(frameDialog,  source2);
            /* 2.1.2.1.3.1.2  */
            break;
        case FRAME_S2_F1_RADIO:
            source2 = FRAME1;
            handleSource2(frameDialog,  source2);
            break;
        case FRAME_S2_F2_RADIO:
            source2 = FRAME2;
            handleSource2(frameDialog,  source2);
            break;
        case FRAME_S2_F3_RADIO:
            source2 = FRAME3;
            handleSource2(frameDialog,  source2);
            break;

        case FRAME_D_HARD_RADIO:
            destination = HARDWARE;
            handleDestination(frameDialog,  destination);
            /* 2.1.2.1.3.1.3  */
            break;
        case FRAME_D_F1_RADIO:
            destination = FRAME1;
            handleDestination(frameDialog,  destination);
            break;
        case FRAME_D_F2_RADIO:
            destination = FRAME2;
            handleDestination(frameDialog,  destination);
            break;
```

```
            case FRAME_D_F3_RADIO:
                destination = FRAME3;
                handleDestination(frameDialog,  destination);
                break;

            case FRAME_NO_MAP_RADIO:
                map = NO_MAP;
                handleMap(frameDialog,  map);
                /* 2.1.2.1.3.1.4  */
                break;
            case FRAME_MAP1_RADIO:
                map = MAP1;
                handleMap(frameDialog,  map);
                break;
            case FRAME_MAP2_RADIO:
                map = MAP2;
                handleMap(frameDialog,  map);
                break;
            }
        }

return;
}

/********          2.1.2.1.3.1.1  handleSource1          **********/

handleSource1(frameDialog,  source1)
DialogPtr           frameDialog;
short               source1;

{
int                 itemType;
Rect                itemRect;
Handle              itemHandle;

/* clear all the radio buttons      */
GetDItem( frameDialog, FRAME_S1_HARD_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_S1_F1_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
```

```
GetDItem( frameDialog, FRAME_S1_F2_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_S1_F3_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, FRAME_OFF );

switch( source1)
      {
      case HARDWARE:
            GetDItem( frameDialog, FRAME_S1_HARD_RADIO,
&itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME1:
            GetDItem( frameDialog, FRAME_S1_F1_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME2:
            GetDItem( frameDialog, FRAME_S1_F2_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME3:
            GetDItem( frameDialog, FRAME_S1_F3_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      }
return;
}

/********          2.1.2.1.3.1.2 handleSource2          **********/

handleSource2(frameDialog, source2)
DialogPtr         frameDialog;
short                   source2;

{
int                     itemType;
Rect                    itemRect;
```

```
Handle                  itemHandle;


GetDItem( frameDialog, FRAME_S2_HARD_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_S2_F1_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_S2_F2_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_S2_F3_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );

switch(  source2)
      {
      case HARDWARE:
            GetDItem( frameDialog, FRAME_S2_HARD_RADIO,
&itemType,
                  &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME1:
            GetDItem( frameDialog, FRAME_S2_F1_RADIO, &itemType,
                  &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME2:
            GetDItem( frameDialog, FRAME_S2_F2_RADIO, &itemType,
                  &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME3:
            GetDItem( frameDialog, FRAME_S2_F3_RADIO, &itemType,
                  &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      }
return;
}
```

```
/********          2.1.2.1.3.1.3 handleDestination          **********/

handleDestination(frameDialog, destination)
DialogPtr               frameDialog;
short                   destination;

{
int                     itemType;
Rect                    itemRect;
Handle                  itemHandle;

/* clear all the radio buttons      */
GetDItem( frameDialog, FRAME_D_HARD_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_D_F1_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_D_F2_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_D_F3_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );

switch( destination)
      {
      case HARDWARE:
            GetDItem( frameDialog, FRAME_D_HARD_RADIO,
                  &itemType, &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME1:
            GetDItem( frameDialog, FRAME_D_F1_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME2:
            GetDItem( frameDialog, FRAME_D_F2_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case FRAME3:
```

```
            GetDItem( frameDialog, FRAME_D_F3_RADIO, &itemType,
                        &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      }
return;
}

/********          2.1.2.1.3.1.4 handleMap          **********/

handleMap(frameDialog, map)
DialogPtr               frameDialog;
short                   map;

{
int                     itemType;
Rect                    itemRect;
Handle                  itemHandle;

/* clear all the radio buttons      */
GetDItem( frameDialog, FRAME_NO_MAP_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_MAP1_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );
GetDItem( frameDialog, FRAME_MAP2_RADIO, &itemType,
      &itemHandle, &itemRect );
SetCtlValue( itemHandle, FRAME_OFF );

switch( map)
      {
      case NO_MAP:
            GetDItem( frameDialog, FRAME_NO_MAP_RADIO,
            &itemType, &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case MAP1:
            GetDItem( frameDialog, FRAME_MAP1_RADIO,
            &itemType, &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
      case MAP2:
```

```
            GetDItem( frameDialog, FRAME_MAP2_RADIO,
            &itemType, &itemHandle, &itemRect );
            SetCtlValue( itemHandle, FRAME_ON );
            break;
        }
return;
}

/***********          2.1.2.1.4   handleWindowChoice          ***********/

#define WINDOW_DIALOG_RES_ID            601

#define WINDOW_S1_NUM                   1
#define WINDOW_S2_NUM                   2
#define WINDOW_D_NUM                    3

#define WINDOW_SAVE_BUTTON              1
#define WINDOW_CANCEL_BUTTON            2
#define WINDOW_TOP_ROW_TEXT             3
#define WINDOW_BOTTOM_ROW_TEXT          4
#define WINDOW_LEFT_COLUMN_TEXT         5
#define WINDOW_RIGHT_COLUMN_TEXT        6
#define WINDOW_TITLE_TEXT               7

handleWindowChoice(theItem)
int  theItem;
{

long            topRowNum, bottomRowNum, leftColumnNum,
                rightColumnNum;
Str255          topRow, bottomRow, leftColumn, rightColumn;
int             itemHit, dialogDone = FALSE;
int             itemType;
Rect            itemRect;
Handle          itemHandle;
DialogPtr       windowDialog;

windowDialog = GetNewDialog( WINDOW_DIALOG_RES_ID, NIL_POINTER,
MOVE_TO_FRONT );
GetDItem( windowDialog, WINDOW_TITLE_TEXT, &itemType,
&itemHandle, &itemRect );

/* get the initial values  */
```

```
switch(theItem)
      {
      case WINDOW_S1_NUM:
            NumToString ( (long)colorSystem.Source1.TopRow,
            topRow);
            NumToString ( (long)colorSystem.Source1.BottomRow,
            bottomRow);
            NumToString ( (long)colorSystem.Source1.LeftColumn,
            leftColumn);
            NumToString ( (long)colorSystem.Source1.RightColumn,
            rightColumn);
            SetIText( itemHandle, "\pSet up for window S1");
      break;
      case WINDOW_S2_NUM:
            SetIText( itemHandle, "\pSet up for window S2");
            NumToString ( (long)colorSystem.Source2.TopRow,
            topRow);
            NumToString ( (long)colorSystem.Source2.BottomRow,
            bottomRow);
            NumToString ( (long)colorSystem.Source2.LeftColumn,
            leftColumn);
            NumToString ( (long)colorSystem.Source2.RightColumn,
            rightColumn);
      break;
      case WINDOW_D_NUM:
            SetIText( itemHandle, "\pSet up for window D");
            NumToString ( (long)colorSystem.Destination.TopRow,
            topRow);
            NumToString ( (long)colorSystem.Destination.BottomRow,
                  bottomRow);
            NumToString ( (long)colorSystem.Destination.LeftColumn,
                  leftColumn);
            NumToString (
      (long)colorSystem.Destination.RightColumn, rightColumn);
      break;
      }

/* display current values   */
GetDItem( windowDialog, WINDOW_TOP_ROW_TEXT, &itemType,
&itemHandle, &itemRect );
SetIText( itemHandle, topRow);
GetDItem( windowDialog, WINDOW_BOTTOM_ROW_TEXT, &itemType,
&itemHandle, &itemRect );
```

```
SetIText( itemHandle, bottomRow);
GetDItem( windowDialog, WINDOW_LEFT_COLUMN_TEXT, &itemType,
&itemHandle, &itemRect );
SetIText( itemHandle, leftColumn);
GetDItem( windowDialog, WINDOW_RIGHT_COLUMN_TEXT, &itemType,
&itemHandle, &itemRect );
SetIText( itemHandle, rightColumn);

ShowWindow( windowDialog );

while ( dialogDone == FALSE )
      {
      ModalDialog( NIL_POINTER, &itemHit );
      switch ( itemHit )
            {
            case WINDOW_SAVE_BUTTON:
                  HideWindow( windowDialog );
                  dialogDone = TRUE;

                  /*get the strings from the respective boxes */
                  GetDItem( windowDialog, WINDOW_TOP_ROW_TEXT,
&itemType,
                        &itemHandle, &itemRect );
                  GetIText( itemHandle, &topRow);
                  GetDItem( windowDialog,
WINDOW_BOTTOM_ROW_TEXT, &itemType,
                        &itemHandle, &itemRect );
                  GetIText( itemHandle, &bottomRow);
                  GetDItem( windowDialog,
WINDOW_LEFT_COLUMN_TEXT, &itemType,
                        &itemHandle, &itemRect );
                  GetIText( itemHandle, &leftColumn);
                  GetDItem( windowDialog,
WINDOW_RIGHT_COLUMN_TEXT, &itemType,
                        &itemHandle, &itemRect );
                  GetIText( itemHandle, &rightColumn);

                  /* convert the strings to numbers */
                  StringToNum ( topRow, &topRowNum);
                  StringToNum ( bottomRow, &bottomRowNum);
                  StringToNum ( leftColumn, &leftColumnNum);
                  StringToNum ( rightColumn, &rightColumnNum);
```

```
                    /* do range checking on numbers */
                    if(topRowNum < 0) topRowNum = 0;
                    if(topRowNum > MAX_ROWS -1) topRowNum =
MAX_ROWS -1;

                    if(bottomRowNum <= topRowNum) bottomRowNum =
topRowNum + 1;
                    if(bottomRowNum > MAX_ROWS) bottomRowNum =
MAX_ROWS ;

                    if(leftColumnNum < 0) leftColumnNum = 0;
                    if(leftColumnNum > MAX_COLUMNS -1)
leftColumnNum = MAX_COLUMNS -1;

                    if(rightColumnNum <= leftColumnNum)
rightColumnNum = leftColumnNum + 1;
                    if(rightColumnNum > MAX_COLUMNS)
rightColumnNum = MAX_COLUMNS;

                    switch(theItem)
                          {
                          case WINDOW_S1_NUM:
                                colorSystem.Source1.TopRow =
                                topRowNum;
                                colorSystem.Source1.BottomRow =
                                bottomRowNum;
                                colorSystem.Source1.LeftColumn =
                                leftColumnNum;
                                colorSystem.Source1.RightColumn =
                                rightColumnNum;
                          break;
                          case WINDOW_S2_NUM:
                                colorSystem.Source2.TopRow =
                                topRowNum;
                                colorSystem.Source2.BottomRow =
                                bottomRowNum;
                                colorSystem.Source2.LeftColumn =
                                leftColumnNum;
                                colorSystem.Source2.RightColumn =
                                rightColumnNum;
                                break;
                          case WINDOW_D_NUM:
```

```
                                colorSystem.Destination.TopRow =
                                topRowNum;
                                colorSystem.Destination.BottomRow =
                                bottomRowNum;
                                colorSystem.Destination.LeftColumn =
                                leftColumnNum;
                                colorSystem.Destination.RightColumn =
                                rightColumnNum;
                                break;
                            }
                        break;
                    case WINDOW_CANCEL_BUTTON:
                        HideWindow( windowDialog );
                        dialogDone = TRUE;
                        break;
                    }
            }

return;
}

/***********          2.1.2.1.5   handleMapChoice          ***********/

#define MAP_DIALOG_RES_ID                   602

#define MAP_OFF                             0
#define MAP_ON                              1

#define MAP_COLOR_MAX_VALUE                 31
#define MAP_BandW_MAX_VALUE                 255

#define MAP_SAVE_BUTTON                     1
#define MAP_CANCEL_BUTTON                   2
#define MAP_CONSTANT_VALUE_TEXT             14
#define MAP_STEP_VALUE_FIRST_TEXT           15
#define MAP_STEP_STEP_TEXT                  16
#define MAP_STEP_VALUE_LAST_TEXT            17
#define MAP_INDIVIDUAL_VALUE_TEXT           18
#define MAP_INDIVIDUAL_ADDRESS_TEXT         19


handleMapChoice(theItem)
int  theItem;
```

```
{

Str255          value1Str, value2Str, addressStr;
short           radioMap =0, radioColor =0, radioOperation =0,
                maxValue;
long            value1 =0, value2 =0, address =0;
int             itemHit, dialogDone = FALSE;
int             itemType;
Rect            itemRect;
Handle          itemHandle;
DialogPtr       windowDialog;

windowDialog = GetNewDialog( MAP_DIALOG_RES_ID, NIL_POINTER,
MOVE_TO_FRONT );

/* get the initial values  */

ShowWindow( windowDialog );

while ( dialogDone == FALSE )
      {
      ModalDialog( NIL_POINTER, &itemHit );
      switch ( itemHit )
            {
            case MAP_SAVE_BUTTON:
                  HideWindow( windowDialog );
                  dialogDone = TRUE;
                  if (radioMap == 0 | radioColor == 0 | radioOperation
                  == 0) break;

                  /*get the strings from the respective boxes */

                  switch  (radioOperation)
                        {
                        case MAP_CONSTANT_RADIO:
                        GetDItem( windowDialog,
                        MAP_CONSTANT_VALUE_TEXT,
                  &itemType,&itemHandle, &itemRect );
                        GetIText( itemHandle, &value1Str);
                        StringToNum ( value1Str, &value1);
                        break;

                        case MAP_STEP_RADIO:
```

```
        GetDItem( windowDialog,
        MAP_STEP_VALUE_FIRST_TEXT,
&itemType,&itemHandle, &itemRect );
        GetIText( itemHandle, &value1Str);
        GetDItem( windowDialog,
        MAP_STEP_STEP_TEXT, &itemType,
                &itemHandle, &itemRect );
        GetIText( itemHandle, &addressStr);
        GetDItem( windowDialog,
        MAP_STEP_VALUE_LAST_TEXT,
&itemType, &itemHandle, &itemRect );
        GetIText( itemHandle, &value2Str);
        /* convert the strings to numbers */
        StringToNum ( value1Str, &value1);
        StringToNum ( value2Str, &value2);
        StringToNum ( addressStr, &address);
        break;

        case MAP_INDIVIDUAL_RADIO:
        GetDItem( windowDialog,
        MAP_INDIVIDUAL_VALUE_TEXT,
&itemType, &itemHandle, &itemRect );
        GetIText( itemHandle, &value1Str);
        GetDItem( windowDialog,
        MAP_INDIVIDUAL_ADDRESS_TEXT,
&itemType, &itemHandle, &itemRect );
        GetIText( itemHandle, &addressStr);
        StringToNum ( value1Str, &value1);
        StringToNum ( addressStr, &address);
        break;
        }

/* check for correct bounds  */
if(value1 < 0) value1 = 0;
if(value1 > maxValue) value1 = maxValue;
if(value2 < 0) value2 = 0;
if(value2 > maxValue) value2 = maxValue;
if(address < 0) address = 0;
if(address > maxValue) address = maxValue;

/* call the functions to operate on the selected map */
doMapOperation(&colorSystem, radioMap,
radioColor, radioOperation,
```

```
                (short)value1, (short)address, (short)value2);
        break;

case MAP_CANCEL_BUTTON:
        HideWindow( windowDialog );
        dialogDone = TRUE;
        break;

case MAP_MAP1_RADIO:
        radioMap = MAP_MAP1_RADIO;
        handleMapSelection(windowDialog, radioMap);

        break;
case MAP_MAP2_RADIO:
        radioMap = MAP_MAP2_RADIO;
        handleMapSelection(windowDialog, radioMap);

        break;

case MAP_RED_RADIO:
        maxValue = MAP_COLOR_MAX_VALUE;
        radioColor = MAP_RED_RADIO;
        handleColorSelection(windowDialog, radioColor);

        break;
case MAP_BLUE_RADIO:
        maxValue = MAP_COLOR_MAX_VALUE;
        radioColor = MAP_BLUE_RADIO;
        handleColorSelection(windowDialog, radioColor);

        break;
case MAP_GREEN_RADIO:
        maxValue = MAP_COLOR_MAX_VALUE;
        radioColor = MAP_GREEN_RADIO;
        handleColorSelection(windowDialog, radioColor);

        break;
case MAP_BLACK_WHITE_RADIO:
        maxValue = MAP_BandW_MAX_VALUE;
        radioColor = MAP_BLACK_WHITE_RADIO;
        handleColorSelection(windowDialog, radioColor);

        break;
```

```
            case MAP_LINEAR_RADIO:
                radioOperation = MAP_LINEAR_RADIO;
                handleOperationSelection(windowDialog,
                radioOperation);
                break;
            case MAP_INVERSE_RADIO:
                radioOperation = MAP_INVERSE_RADIO;
                handleOperationSelection(windowDialog,
                radioOperation);
                break;
            case MAP_CONSTANT_RADIO:
                radioOperation = MAP_CONSTANT_RADIO;
                handleOperationSelection(windowDialog,
                radioOperation);
                break;
            case MAP_STEP_RADIO:
                radioOperation = MAP_STEP_RADIO;
                handleOperationSelection(windowDialog,
                radioOperation);
                break;
            case MAP_INDIVIDUAL_RADIO:
                radioOperation = MAP_INDIVIDUAL_RADIO;
                handleOperationSelection(windowDialog,
                radioOperation);
                break;
            }
        }

return;
}
/***********          2.1.2.1.5.1   handleMapSelection    ***********/

handleMapSelection(windowDialog,  selection)
DialogPtr       windowDialog;
short           selection;

{
int             itemType;
Rect            itemRect;
Handle          itemHandle;

/* clear all the radio buttons      */
```

```
GetDItem( windowDialog, MAP_MAP1_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_MAP2_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );

GetDItem( windowDialog, selection, &itemType, &itemHandle, &itemRect
);
SetCtlValue( itemHandle, MAP_ON );
return;
}

/***********          2.1.2.1.5.2  handleColorSelection  ***********/

handleColorSelection(windowDialog, selection)
DialogPtr            windowDialog;
short                selection;

{
int                  itemType;
Rect                 itemRect;
Handle               itemHandle;

/* clear all the radio buttons      */
GetDItem( windowDialog, MAP_RED_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_BLUE_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_GREEN_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_BLACK_WHITE_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, MAP_OFF );

GetDItem( windowDialog, selection, &itemType, &itemHandle, &itemRect
);
SetCtlValue( itemHandle, MAP_ON );
return;
}
```

```
/***********          2.1.2.1.5.3    handleOperationSelection  ******/

handleOperationSelection(windowDialog,  selection)
DialogPtr              windowDialog;
short                  selection;

{
int                    itemType;
Rect                   itemRect;
Handle                 itemHandle;

/* clear all the radio buttons      */
GetDItem( windowDialog, MAP_LINEAR_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_INVERSE_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_CONSTANT_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_STEP_RADIO, &itemType, &itemHandle,
&itemRect );
SetCtlValue( itemHandle, MAP_OFF );
GetDItem( windowDialog, MAP_INDIVIDUAL_RADIO, &itemType,
&itemHandle, &itemRect );
SetCtlValue( itemHandle, MAP_OFF );

GetDItem( windowDialog, selection, &itemType, &itemHandle, &itemRect
);
SetCtlValue( itemHandle, MAP_ON );
return;
}


/***********          2.1.2.1.6    handleOperationsChoice      ******/

#define MOVE_NUM                    1
#define RGBtoYIQ_NUM                2
#define YIQtoRGB_NUM                3
#define ADD_NUM                     4
#define SUBTRACT_NUM                5
```

```
handleOperationsChoice(theItem)
int  theItem;
{
switch(theItem)
      {
      case MOVE_NUM:
            doMove(&colorSystem);
      break;
      case RGBtoYIQ_NUM:
            doRGBtoYIQ(&colorSystem);
      break;
      case YIQtoRGB_NUM:
            doYIQtoRGB(&colorSystem);
      break;
      case ADD_NUM:
            doAdd(&colorSystem);
      break;
   case SUBTRACT_NUM:
            doSubtract(&colorSystem);
      break;
      }
return;
}
/***********          2.1.2.1.7   handleColorMoveChoice      ***********/

#define S1_RYtoD_NUM                  1
#define S1_GItoD_NUM                  2
#define S1_BQtoD_NUM                  3
#define S1toD_RY_NUM                  4
#define S1toD_GI_NUM                  5
#define S1toD_BQ_NUM                  6

handleColorMoveChoice(theItem)
int  theItem;
{
switch(theItem)
      {
      case S1_RYtoD_NUM:
            doS1_RYtoD(&colorSystem);
      break;
      case S1_GItoD_NUM:
            doS1_GItoD(&colorSystem);
```

```
        break;
        case S1_BQtoD_NUM:
              doS1_BQtoD(&colorSystem);
        break;
        case S1toD_RY_NUM:
              doS1toD_RY(&colorSystem);
        break;
        case S1toD_GI_NUM:
              doS1toD_GI(&colorSystem);
        break;
        case S1toD_BQ_NUM:
              doS1toD_BQ(&colorSystem);
        break;
        }
return;
}

/******************          end of file          **********/
```

Appendix B Source code for image processing hardware interface

```
/*****************************************************************/
/************          HardwareFunctions.c                  **/
/*****************************************************************/

/*******      last modified 8 march 1990 by jm        *****/

/*****  these are the functions to initialize the frame grabber, snap an
image and    */
/* set the image to live                                          **/


#define    CONTROL_REG              0xb00600
#define    PAN_REG                  0xb00604
#define    CHAN_REG            0xb00608

/***************          imageLive                ***************/
void  imageLive()
{
int *cp;

cp = (int *)CONTROL_REG;
*cp  =  0x314;
return;
}

/***************          imageSnap                ************/
void  imageSnap()
{
int *cp;

cp = (int *)CONTROL_REG;
*cp  =  0x0114;
return;
}

/***************          1. 5 imageInit           ************/
void  imageInit()

{
```

```
int *cp;

cp = (int *)CONTROL_REG;
*cp = 0x80e0;

cp = (int *)PAN_REG;
*cp = 0x0000;

cp = (int *)CHAN_REG;
*cp = 0xff04;

imageLive();

return;
}


/*****************************************************************/
```

Appendix C Source code for image processing functions

```
/*************************************************************************/
/************          ImageProcessing.c                  ********/
/*************************************************************************/

/*******      last modified 29 march 1990 by jm       *****/

#include <stdio.h>
#include <storage.h>
#include "ColorStructs.h"
#include <math.h>


/**************          1.6  frameInit          ************/

#define min(x,y)  (((x) < (y)) ? (x):(y))
#define Min(x, y, z)   (min( (min(x,y)),(z)))

#define HARDWARE_FRAME  0
#define BASE_ADDRESS        0xb00000

frameInit(colorSystem)
COLOR_SYSTEM *colorSystem;

{
short       loop;

loop = HARDWARE_FRAME;
colorSystem->Frame[loop].NumRows = MAX_ROWS;
colorSystem->Frame[loop].NumColumns = MAX_COLUMNS;
colorSystem->Frame[loop].RowIncrement =
HARDWARE_ROW_INCREMENT;
colorSystem->Frame[loop].Pointer = (short *)BASE_ADDRESS;

for (loop = HARDWARE_FRAME + 1; loop < NUMBER_OF_FRAMES; loop++)
      {
      colorSystem->Frame[loop].NumRows = MAX_ROWS;
      colorSystem->Frame[loop].NumColumns = MAX_COLUMNS;
      colorSystem->Frame[loop].RowIncrement = MAX_COLUMNS;
      colorSystem->Frame[loop].Pointer =
```

```
            (short *)mlalloc ( (unsigned long) sizeof(short) * MAX_ROWS
* MAX_COLUMNS);
      if (colorSystem->Frame[loop].Pointer == NULL)
            printf("\ninsufficient memory for allocation of software
frame %d", loop);
      }

return;
}

/*************          1.7   mapInit          ************/

#define NO_MAP                              0

mapInit(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short loop;

colorSystem->ProcessMap = NO_MAP;

for (loop = 0; loop < NUMBER_OF_MAPS; loop++)
      {
      colorSystem->Map[loop] =
            (short *)mlalloc ( (unsigned long) sizeof(short) * 32 * 32 *
32);
      if (colorSystem->Map[loop] == NULL)
            printf("\ninsufficient memory for allocation of map %d",
loop);
      }
colorSystem->RGBtoYIQ = colorSystem->Map[NUMBER_OF_MAPS - 2];
colorSystem->YIQtoRGB = colorSystem->Map[NUMBER_OF_MAPS - 1];

map3Init(colorSystem->RGBtoYIQ);
map4Init(colorSystem->YIQtoRGB);

return;
}

/*************          1.7.1   map3Init          ************/

map3Init(Map_PNT)
short *Map_PNT;
```

```
{
short       Redbit, Greenbit, Bluebit;
short       R, G, B, H, S, I, rgb_value;
double      K, F, Sqrt3, Constant, DegPerRadian;

Sqrt3 = sqrt( 3.0);
Constant = 31.0/360.0;
DegPerRadian = 180.0/3.14159;

for(R = 0; R < 32; R++)
      {
      Redbit = R <<10;

      for(G=0; G<32; G++)
            {
            Greenbit = G<<5;

            for(B=0; B < 32; B++)
                  {
                  Bluebit = B;
                  rgb_value = Redbit|Greenbit|Bluebit;
                  K = (R + G + B + 0.0001)/3.0;
                  I= K;
                  S = (1.0 - Min(R,G,B)/K) * 31.0;
                  if( G == B ) H = 0;
                  else
                        {
                        F = (2.0*R - G - B)/(G - B);
                        if( F < 0.0)
                              H = (270.0 - atan2(F, Sqrt3)*DegPerRadian)
                                    * Constant;
                        else
                              H = (90.0 - atan2(F, Sqrt3)*DegPerRadian)
                                    * Constant;
                   Map_PNT[rgb_value] = (I<<10) | (H<<5) | S;

            }
      }
return;
}

/*************          1.7.2   map4Init          ************/
```

```
map4Init(Map_PNT)
short *Map_PNT;

{

short       Redbit, Greenbit, Bluebit, Hbit, Sbit, Ibit;
short       R, G, B,H, S, I, hsi_value;
double      K, P, Constant;

Constant = 360.0/31.0;

for(H = 0; H < 32; H++)
      {
      Hbit = H << 5;
      for(I = 0; I < 32; I++)
            {
            Ibit = I < 10;

            for( S = 0; S < 32; S++)
                  {
                  Sbit = S;
                  hsi_value = Hbit | Sbit | Ibit;
                  P = H * Constant;
                  if( P > 240.0)
                        {
                        K = cos(P - 240.0)/cos(120.0 - P);
                        G = I - I * S/31.0;
                        B = I + I * S * K/31.0;
                        R = I*(1.0 + S *(1.0 - K)/31.0);
                        }
                  else
                        if(P > 120.0 )
                              {
                              K = cos(P - 120)/cos(-60.0 - P);
                              R = I - I * S/31.0;
                              G = I + I * S * K/31.0;
                              B = I*(1.0 + S * (1.0 - K)/31.0);
                              }
                        else
                              {
                              K = cos(P)/cos(60.0 - P);
                              B = I - I * S/31.0;
                              R = I + I * S *K /31.0;
```

```
                                G = I*(1.0 + S * (1.0 - K)/31.0);
                                }
                        if ( R < 0) R = 0; else if(R > 31) R = 31;
                        if ( G < 0) G = 0; else if(G > 31) G = 31;
                        if ( B < 0) B = 0; else if(B > 31) B = 31;
                        Map_PNT[hsi_value] = (R<<10) | (G<<5) | B;
                        }
                }
        }
return;
}


/**************          1.8   imageWindowInit     *************/

#define HARDWARE      0

imageWindowInit(colorSystem)
COLOR_SYSTEM *colorSystem;
{

colorSystem->Source1.Frame = HARDWARE;
colorSystem->Source1.TopRow = 0;
colorSystem->Source1.BottomRow = MAX_ROWS;
colorSystem->Source1.LeftColumn = 0;
colorSystem->Source1.RightColumn = MAX_COLUMNS;

colorSystem->Source2.Frame = HARDWARE;
colorSystem->Source2.TopRow = 0;
colorSystem->Source2.BottomRow = MAX_ROWS;
colorSystem->Source2.LeftColumn = 0;
colorSystem->Source2.RightColumn = MAX_COLUMNS;

colorSystem->Destination.Frame = HARDWARE;
colorSystem->Destination.TopRow = 0;
colorSystem->Destination.BottomRow = MAX_ROWS;
colorSystem->Destination.LeftColumn = 0;
colorSystem->Destination.RightColumn = MAX_COLUMNS;
return;
}

/*****************      global variables      ************************/
```

```
#define RED_MASK                0x7c00
#define GREEN_MASK              0x03e0
#define BLUE_MASK               0x001f
#define COM_RED_MASK            !RED_MASK
#define COM_GREEN_MASK          !GREEN_MASK
#define COM_BLUE_MASK           !BLUE_MASK
#define BW_MASK                 0x00ff
#define COM_BW_MASK             !BW_MASK

short   *S1_Pointer,*S2_Pointer, *D_Pointer,
         S1_row_increment, S2_row_increment, D_row_increment;
int     rownum,columnnum;

/*****************     GetS1DPtr          ************************/

GetS1DPtr(colorSystem)

COLOR_SYSTEM *colorSystem;
{

int   s1_rownum, s1_columnnum, d_rownum, d_columnnum;
long  s1offset, doffset;

S1_Pointer = colorSystem->Frame[colorSystem->Source1.Frame].Pointer;
D_Pointer = colorSystem->Frame[colorSystem-
>Destination.Frame].Pointer;

s1offset = colorSystem->Source1.TopRow;
doffset  = colorSystem->Destination.TopRow;

S1_row_increment = colorSystem->Frame[colorSystem-
>Source1.Frame].RowIncrement;
D_row_increment  = colorSystem->Frame[colorSystem-
>Destination.Frame].RowIncrement;

S1_Pointer += s1offset * S1_row_increment
      + colorSystem->Source1.LeftColumn;
D_Pointer +=  doffset * D_row_increment
      + colorSystem->Destination.LeftColumn;

s1_columnnum = MAX_COLUMNS - colorSystem->Source1.LeftColumn;
d_columnnum = colorSystem->Destination.RightColumn
      - colorSystem->Destination.LeftColumn;
```

```
s1_rownum = MAX_ROWS - colorSystem->Source1.TopRow;
d_rownum = colorSystem->Destination.BottomRow
      - colorSystem->Destination.TopRow;

rownum = min(s1_rownum, d_rownum);
columnnum = min(s1_columnnum, d_columnnum);

return;
}

/*****************      GetS1S2DPtr          ************************/

GetS1S2DPtr(colorSystem)
COLOR_SYSTEM *colorSystem;
{
int         s1_rownum, s1_columnnum,
            s2_rownum, s2_columnnum,
            d_rownum, d_columnnum;
long s1offset, s2offset, doffset;

S1_Pointer = colorSystem->Frame[colorSystem->Source1.Frame].Pointer;
S2_Pointer = colorSystem->Frame[colorSystem->Source2.Frame].Pointer;
D_Pointer = colorSystem->Frame[colorSystem-
>Destination.Frame].Pointer;

S1_row_increment =
      colorSystem->Frame[colorSystem->Source1.Frame].RowIncrement;
S2_row_increment =
      colorSystem->Frame[colorSystem->Source2.Frame].RowIncrement;
D_row_increment =
      colorSystem->Frame[colorSystem->Source2.Frame].RowIncrement;

s1offset = colorSystem->Source1.TopRow;
s2offset = colorSystem->Source2.TopRow;
doffset = colorSystem->Destination.TopRow;

S1_Pointer += s1offset * S1_row_increment
      + colorSystem->Source1.LeftColumn;
S2_Pointer += s2offset * S2_row_increment
      + colorSystem->Source2.LeftColumn;
D_Pointer += doffset * D_row_increment
      + colorSystem->Destination.LeftColumn;
```

```
s1_columnnum = MAX_COLUMNS - colorSystem->Source1.LeftColumn;
s2_columnnum = MAX_COLUMNS - colorSystem->Source2.LeftColumn;
d_columnnum  = colorSystem->Destination.RightColumn
      - colorSystem->Destination.LeftColumn;

s1_rownum = MAX_ROWS - colorSystem->Source1.TopRow;
s2_rownum = MAX_ROWS - colorSystem->Source2.TopRow;
d_rownum   = colorSystem->Destination.BottomRow
      - colorSystem->Destination.TopRow;

rownum = Min( s1_rownum, s2_rownum, d_rownum);
columnnum = Min( s1_columnnum, s2_columnnum, d_columnnum);

return;
}

/**************          2.1.2.1.6.1   doRGBtoYIQ    *************/

doRGBtoYIQ(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short *S1PNT, *DPNT, *Map_PNT;
int i, j;

GetS1DPtr(colorSystem);
Map_PNT = colorSystem->RGBtoYIQ ;

for (i=0; i< rownum; i++)
      {
      S1PNT =  S1_Pointer;
      DPNT  =  D_Pointer;

      for (j=0; j< columnnum; j++)
            {
            *DPNT++ = *( Map_PNT + (*S1PNT++ & 0x7fff));
            }
      S1_Pointer  +=  S1_row_increment;
      D_Pointer   +=  D_row_increment;
      }
return;
}

/**************          2.1.2.1.6.2   doYIQtoRGB    *************/
```

```
doYIQtoRGB(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short *S1PNT, *DPNT, *Map_PNT;
int i, j;

GetS1DPtr(colorSystem);
Map_PNT = colorSystem->YIQtoRGB ;
for (i=0; i< rownum; i++)
      {
      S1PNT =  S1_Pointer;
      DPNT  =  D_Pointer;
      for (j=0; j< columnnum; j++)
      {
      *DPNT++ = *( Map_PNT + (*S1PNT++ & 0x7fff));
      }
      S1_Pointer  +=  S1_row_increment;
      D_Pointer   +=  D_row_increment;
      }
return;
}

/**************          2.1.2.1.6.3   doAdd     *************/

doAdd(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *S2PNT, *DPNT, *Map_PNT;
short       data;

int         i, j;

GetS1S2DPtr(colorSystem);
if( colorSystem->ProcessMap == NO_MAP)
      {
      for (i=0; i< rownum; i++)
            {
            S1PNT =  S1_Pointer;
            S2PNT =  S2_Pointer;
            DPNT  =  D_Pointer;
            for (j=0; j< columnnum; j++)
```

```
                    {
                    *DPNT++ = ((((*S1PNT)&RED_MASK) +
                          ((*S2PNT)&RED_MASK))>>1 & RED_MASK)
                          |((((*S1PNT)&GREEN_MASK) +
                          ((*S2PNT)&GREEN_MASK))>>1 & GREEN_MASK)
                          |((*S1PNT + *S2PNT)>>1 & BLUE_MASK);
                    S1PNT++;
                    S2PNT++;
                    }
                    S1_Pointer  +=  S1_row_increment;
                    S2_Pointer  +=  S2_row_increment;
                    D_Pointer   +=  D_row_increment;
               }
          }
          else
               {
               Map_PNT  =  colorSystem->Map[colorSystem->ProcessMap -
1];
               for (i=0; i< rownum; i++)
               {
               S1PNT =   S1_Pointer;
               S2PNT =   S2_Pointer;
               DPNT  =  D_Pointer;
               for (j=0; j< columnnum; j++)
                    {
                    data = ((((*S1PNT)&RED_MASK) +
                          ((*S2PNT)&RED_MASK))>>1 & RED_MASK)
                          |((((*S1PNT)&GREEN_MASK) +
                          ((*S2PNT)&GREEN_MASK))>>1 & GREEN_MASK)
                          |((*S1PNT + *S2PNT)>>1 & BLUE_MASK);

                    *DPNT++ = *(Map_PNT + data);
                    S1PNT++;
                    S2PNT++;
                    }

               S1_Pointer  +=  S1_row_increment;
               S2_Pointer  +=  S2_row_increment;
               D_Pointer   +=  D_row_increment;
               }
          }
return;
}
```

```
/**************          2.1.2.1.6.4   doSubtract     *************/


doSubtract(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *S2PNT, *DPNT, *Map_PNT;
short       data;
int         i, j;

GetS1S2DPtr(colorSystem);
if( colorSystem->ProcessMap == NO_MAP)
      {
      for (i=0; i< rownum; i++)
            {
            S1PNT =  S1_Pointer;
            S2PNT =  S2_Pointer;
            DPNT  =  D_Pointer;
                  for (j=0; j< columnnum; j++)
                  {
                  * DPNT++  = (abs(((*S1PNT)&RED_MASK) -
                        ((*S2PNT)&RED_MASK))&RED_MASK)
                        |(abs(((*S1PNT)&GREEN_MASK) -
                        ((*S2PNT)&GREEN_MASK))&GREEN_MASK)
                        |(abs(*S1PNT - *S2PNT)&BLUE_MASK);
                  S1PNT++;
                  S2PNT++;
                  }

            S1_Pointer  +=  S1_row_increment;
            S2_Pointer  +=  S2_row_increment;
            D_Pointer   +=  D_row_increment;
            }
      }
else
      {
      Map_PNT = colorSystem->Map[colorSystem->ProcessMap - 1];
      for (i=0; i< rownum; i++)
            {
            S1PNT =  S1_Pointer;
            S2PNT =  S2_Pointer;
            DPNT  =  D_Pointer;
```

```
            for (j=0; j< columnnum; j++)
                  {
                  data =  (abs(((*S1PNT)&RED_MASK) -
                        ((*S2PNT)&RED_MASK))&RED_MASK)
                        |(abs(((*S1PNT)&GREEN_MASK) -
                        ((*S2PNT)&GREEN_MASK))&GREEN_MASK)
                        |(abs(*S1PNT - *S2PNT)&BLUE_MASK);

                  *DPNT++ = *(Map_PNT + data);
                  S1PNT++;
                  S2PNT++;
                  }

            S1_Pointer  +=  S1_row_increment;
            S2_Pointer  +=  S2_row_increment;
            D_Pointer   +=  D_row_increment;
            }
      }
return;
}

/**************          2.1.2.1.6.5   doMove     *************/


doMove(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT, *Map_PNT;
int         i, j;

GetS1DPtr(colorSystem);
if( colorSystem->ProcessMap == NO_MAP)
      {
      for (i=0; i< rownum; i++)
            {
            S1PNT =  S1_Pointer;
            DPNT  =  D_Pointer;
            for (j=0; j< columnnum; j++)
                  {
                  *DPNT++ = *S1PNT++;
                  }
            S1_Pointer  +=  S1_row_increment;
            D_Pointer   +=  D_row_increment;
```

```
            }
        }
    else
        {
        Map_PNT = colorSystem->Map[colorSystem->ProcessMap - 1] ;
        for (i=0; i< rownum; i++)
            {
            S1PNT =  S1_Pointer;
            DPNT  =  D_Pointer;
            for (j=0; j< columnnum; j++)
                {
                *DPNT++ = *( Map_PNT + *S1PNT++);
                }
            S1_Pointer  +=  S1_row_increment;
            D_Pointer   +=  D_row_increment;
            }
        }
    return;
    }

    /**************          2.1.2.1.7.1   doS1_RYtoD    *************/

    doS1_RYtoD(colorSystem)
    COLOR_SYSTEM *colorSystem;
    {
    short       *S1PNT, *DPNT;
    int         i, j;

    GetS1DPtr(colorSystem);
    for (i=0; i< rownum; i++)
        {
        S1PNT =  S1_Pointer;
        DPNT  =  D_Pointer;
        for (j=0; j< columnnum; j++)
            {
            *DPNT++ = ( *DPNT & COM_BLUE_MASK)|(( *S1PNT &
                RED_MASK) >>10);
            S1PNT++;
            }
        S1_Pointer  +=  S1_row_increment;
        D_Pointer   +=  D_row_increment;
        }
    return;
```

```
}

/**************          2.1.2.1.7.2   doS1_GItoD      *************/

doS1_GItoD(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT;
int         i, j;

GetS1DPtr(colorSystem);
for (i=0; i< rownum; i++)
      {
      S1PNT =  S1_Pointer;
      DPNT  =  D_Pointer;
      for (j=0; j< columnnum; j++)
            {
            *DPNT++ = ( *DPNT&COM_BLUE_MASK)|((
                  *S1PNT&GREEN_MASK) >> 5);
            S1PNT++;
            }
            S1_Pointer  +=  S1_row_increment;
            D_Pointer   +=  D_row_increment;
      }
return;
}

/**************          2.1.2.1.7.3   doS1_BQtoD      *************/

doS1_BQtoD(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT;
int         i, j;

GetS1DPtr(colorSystem);
for (i=0; i< rownum; i++)
      {
      S1PNT =  S1_Pointer;
      DPNT  =  D_Pointer;
      for (j=0; j< columnnum; j++)
            {
```

```
                *DPNT++ = ( *DPNT&COM_BLUE_MASK)|((*S1PNT) &
                        BLUE_MASK);
                S1PNT++;
                }
        S1_Pointer  +=  S1_row_increment;
        D_Pointer   +=  D_row_increment;
        }
return;
}

/**************          2.1.2.1.7.4   doS1toD_RY      *************/

doS1toD_RY(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT;
int         i, j;

GetS1DPtr(colorSystem);
for (i=0; i< rownum; i++)
        {
        S1PNT =  S1_Pointer;
        DPNT  =  D_Pointer;
        for (j=0; j< columnnum; j++)
                {
                *DPNT++ = (*DPNT&COM_RED_MASK)
                        |((*S1PNT&BLUE_MASK)<<10);
                S1PNT++;
                }
        S1_Pointer  +=  S1_row_increment;
        D_Pointer   +=  D_row_increment;
        }
return;
}

/**************          2.1.2.1.7.5   doS1toD_GI      *************/

doS1toD_GI(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT;
int         i, j;
```

```
GetS1DPtr(colorSystem);
for (i=0; i< rownum; i++)
        {
        S1PNT =  S1_Pointer;
        DPNT  =  D_Pointer;
        for (j=0; j< columnnum; j++)
                {
                *DPNT++ = (*DPNT & COM_GREEN_MASK)
                        |(( *S1PNT & 0x001f) << 5);
                S1PNT++;
                }
        S1_Pointer  +=  S1_row_increment;
        D_Pointer   +=  D_row_increment;
        }
return;
}

/**************          2.1.2.1.7.6   doS1toD_BQ     *************/

doS1toD_BQ(colorSystem)
COLOR_SYSTEM *colorSystem;
{
short       *S1PNT, *DPNT;
int         i, j;

GetS1DPtr(colorSystem);
for (i=0; i< rownum; i++)
        {
        S1PNT =  S1_Pointer;
        DPNT  =  D_Pointer;
        for (j=0; j< columnnum; j++)
                {
                *DPNT++ =(*DPNT&COM_BLUE_MASK)
                        |((*S1PNT) & BLUE_MASK);
                S1PNT++;
                }
        S1_Pointer  +=  S1_row_increment;
        D_Pointer   +=  D_row_increment;
        }
return;
}

/**************          2.1.2.1.5.1   doMapOperation  *************/
```

```
doMapOperation( colorSystem, radioMap, radioColor, radioOperation,
      value1, address, value2)
COLOR_SYSTEM  *colorSystem;
short                  radioMap, radioColor, radioOperation, value1,
address, value2;

{
short *MAP_PNT;

MAP_PNT = colorSystem->Map[ radioMap - MAP_MAP1_RADIO];
switch(radioOperation)
      {
      case MAP_LINEAR_RADIO:
            linearFun( MAP_PNT, radioColor);
            break;
      case MAP_INVERSE_RADIO:
            inverseFun( MAP_PNT, radioColor);
            break;
      case MAP_CONSTANT_RADIO:
            constantFun(MAP_PNT, radioColor, value1);
            break;
      case MAP_STEP_RADIO:
            stepFun(MAP_PNT, radioColor, value1, address, value2);
            break;
      case MAP_INDIVIDUAL_RADIO:
            individualFun(MAP_PNT, radioColor, address, value1);
            break;
      }
return;
}

/**************       2.1.2.1.5.1.1   linearFun  *************/

linearFun( MAP_PNT, radioColor)
short *MAP_PNT, radioColor;
{
short i, j, k, Redbit, Greenbit, Bluebit, Value;

if(radioColor == MAP_BLACK_WHITE_RADIO)
      for(i = 0; i<256; i++) MAP_PNT[i] = i;
else
      {
```

```
        for(i = 0; i< 32; i++)
                {
                Redbit = i<<10;
                for(j=0; j<32; j++)
                        {
                        Greenbit = j<<5;
                        for(k=0; k<32; k++)
                {
                                Bluebit = k;
                                Value = Redbit| Greenbit|Bluebit;
                                switch(radioColor)
                                        {
                                        case MAP_RED_RADIO:
                                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_RED_MASK) | Redbit;
                                                break;
                                        case MAP_GREEN_RADIO:
                                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_GREEN_MASK) | Greenbit;
                                                break;
                                        case MAP_BLUE_RADIO:
                                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_BLUE_MASK) | Bluebit;
                                                break;
                                        }
                                }
                        }
                }
        }
return;
}

/**************      2.1.2.1.5.1.2   inverseFun  *************/

inverseFun( MAP_PNT, radioColor)
short *MAP_PNT, radioColor;
{

short i, j, k, Redbit, Greenbit, Bluebit, Value;

if(radioColor == MAP_BLACK_WHITE_RADIO)
        for(i = 0; i<256; i++)  MAP_PNT[i] =  255-i;
else
```

```
        {
        for(i = 0; i< 32; i++)
                {
                Redbit = (i<<10) & RED_MASK;
                for(j=0; j<32; j++)
                        {
                        Greenbit = (j<<5) & GREEN_MASK;
                        for(k=0; k<32; k++)
                                {
                                Bluebit = k&BLUE_MASK;
                                Value = Redbit| Greenbit| Bluebit;
                                switch(radioColor)
                                        {
                                        case MAP_RED_RADIO:
                                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_RED_MASK) | ((31-i)<<10)&RED_MASK;
                                                break;
                                        case MAP_GREEN_RADIO:
                                                MAP_PNT[Value] =
         (MAP_PNT[Value]&COM_GREEN_MASK)
        | ((31-j)<<5)&GREEN_MASK;
                                                break;
                                        case MAP_BLUE_RADIO:
                                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_BLUE_MASK)
        | (31-k)&BLUE_MASK;
                                                break;
                                        }
                                }
                        }
                }
        }
return;
}

/**************      2.1.2.1.5.1.3   constantFun  *************/

constantFun(MAP_PNT, radioColor, value1)
short  *MAP_PNT,  radioColor, value1;

{
short i, j, k, Redbit, Greenbit, Bluebit, rgb_Value;
```

```
if(radioColor == MAP_BLACK_WHITE_RADIO)
        for(i = 0; i<256; i++) MAP_PNT[i] = value1;
else
        {
        for(i = 0; i< 32; i++)
                {
                Redbit = (i<<10) & RED_MASK;
                for(j=0; j<32; j++)
                        {
                        Greenbit = (j<<5) & GREEN_MASK;
                        for(k=0; k<32; k++)
                                {
                                Bluebit = k&BLUE_MASK;
                                rgb_Value = Redbit| Greenbit| Bluebit;
                                switch(radioColor)
                                        {
                                        case MAP_RED_RADIO:
                                                MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_RED_MASK)
        |(((value1)<<10)&RED_MASK);
                                                break;
                                        case MAP_GREEN_RADIO:
                                                MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_GREEN_MASK)
        |(((value1)<<5)&GREEN_MASK);
                                                break;
                                        case MAP_BLUE_RADIO:
                                                MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_BLUE_MASK)
        ((value1)&BLUE_MASK);
                                                break;
                                        }
                                }
                        }
                }
        }
return;
}

/**************      2.1.2.1.5.1.4   stepFun *************/

stepFun(MAP_PNT, radioColor, value1, step_value, value2)
short *MAP_PNT, radioColor, value1, step_value, value2;
```

```
{

short i, j, k, Redbit, Greenbit, Bluebit, rgb_Value;

if(radioColor == MAP_BLACK_WHITE_RADIO)
        for(i = 0; i<256; i++)
                if( i < step_value) MAP_PNT[i] = value1;
                else MAP_PNT[i] = value2;
else
        {
        for(i = 0; i< 32; i++)
                {
                Redbit = (i<<10) & RED_MASK;
                for(j=0; j<32; j++)
                        {
                        Greenbit = (j<<5) & GREEN_MASK;
                        for(k=0; k<32; k++)
                                {
                                Bluebit = k&BLUE_MASK;
                                rgb_Value = Redbit| Greenbit| Bluebit;
                                switch(radioColor)
                                        {
                                        case MAP_RED_RADIO:
                                                if(i< step_value)
                                                        MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_RED_MASK)
        |(((value1)<<10)&RED_MASK);
                                                else
                                                        MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_RED_MASK)
        |(((value2)<<10)&RED_MASK);
                                                break;
                                        case MAP_GREEN_RADIO:
                                                if(j<step_value)
                                                        MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_GREEN_MASK)
        |(((value1)<<5)&GREEN_MASK);
                                                else
                                                        MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_GREEN_MASK)
        |(((value2)<<5)&GREEN_MASK);
                                                break;
```

```
                                case MAP_BLUE_RADIO:
                                        if(k<step_value)
                                                MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_BLUE_MASK)
        |((value1)&BLUE_MASK);
                                        else
                                                MAP_PNT[rgb_Value] =
        (MAP_PNT[rgb_Value]&COM_BLUE_MASK)
        |((value2)&BLUE_MASK);
                                        break;
                                }
                        }
                }
        }
return;
}

/**************      2.1.2.1.5.1.5   individualFun  *************/

individualFun(MAP_PNT, radioColor, address, value1)
short *MAP_PNT, radioColor,  address, value1;

{
short       i, j, k, Redbit, Greenbit, Bluebit, Value;

switch(radioColor)
        {
        case MAP_RED_RADIO:
                Redbit = address << 10;
                for(j=0; j<32; j++)
                        {
                        Greenbit = j << 5;
                        for(k=0; k<32; k++)
                                {
                                Bluebit = k;
                                Value =  Redbit|Greenbit|Bluebit;
                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_RED_MASK)
                                        |((value1<<10)&RED_MASK);
                                }
                        }
                break;
```

```
        case MAP_GREEN_RADIO:
                Greenbit = address << 5;
                for(i=0; i<32; i++)
                        {
                        Redbit = i<<10;
                        for(k=0; k<32; k++)
                                {
                                Bluebit = k;
                                Value = Redbit|Greenbit| Bluebit;
                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_GREEN_MASK)
                                        |((value1<<5)&GREEN_MASK);
                                }
                        }
                break;

        case MAP_BLUE_RADIO:
                Bluebit = address;
                for(i=0; i <32; i++)
                        {
                        Redbit = i<<10;
                        for(j=0; j<32; j++)
                                {
                                Greenbit = j<<5;
                                Value = Redbit|Greenbit| Bluebit;
                                MAP_PNT[Value] =
        (MAP_PNT[Value]&COM_BLUE_MASK)
                                        |(value1&BLUE_MASK);
                                }
                        }
                break;

        default:
                *(MAP_PNT + address) =value1;
                break;
        }
return;
}

/***********          end of file          ****************/
```

Appendix D Include file

```
/*****************************************************************/
/********        ColorStructs.h                           ******/
/*****************************************************************/

/**********     last modified 14 march 1990     by jm        */

/* this is the include file for all the image processing structs          */


#define MAX_ROWS                        480
#define MAX_COLUMNS                     640
#define HARDWARE_ROW_INCREMENT          1024

#define NUMBER_OF_FRAMES                4
#define NUMBER_OF_MAPS                  4

/* defines for the map operations, needed also in the function that does
it */
/***********          2.1.2.1.5   handleMapChoice        ***********/
#define MAP_MAP1_RADIO                          3
#define MAP_MAP2_RADIO                          4
#define MAP_RED_RADIO                           5
#define MAP_BLUE_RADIO                          6
#define MAP_GREEN_RADIO                         7
#define MAP_BLACK_WHITE_RADIO                   8
#define MAP_LINEAR_RADIO                        9
#define MAP_INVERSE_RADIO                       10
#define MAP_CONSTANT_RADIO                      11
#define MAP_STEP_RADIO                          12
#define MAP_INDIVIDUAL_RADIO                    13


typedef struct Window
      {
      short             Frame,
                        TopRow,
                        BottomRow,
```

```
                    LeftColumn,
                    RightColumn;
    } WINDOW;


typedef struct Frame
    {
    short           NumRows,
                    NumColumns,
                    RowIncrement;
    short           *Pointer;
    } FRAME;

typedef struct ColorSystem
    {
    FRAME           Frame[NUMBER_OF_FRAMES];
    WINDOW          Source1,
                    Source2,
                    Destination;
    short           *Map[NUMBER_OF_MAPS],
                    *RGBtoYIQ,
                    *YIQtoRGB;
    short           ProcessMap;
    } COLOR_SYSTEM;


/***********          end of file          ************/
```

Appendix E Examples of CLIPS Rules for image processing

```
;;;;**************************************************************************
;;;;THIS IS A EXAMPLE FOR USING IMAGEPROCESS FUNCTION IN CLIPS
;;;;**************************************************************************


(defrule livepic
  ?start <- ( start fact)
  ( show live picture)
=>
   (retract ?start)
   (livefun)
   (assert (live picture))
   (assert (snap picture)))


(defrule store_pic
   ?f1<-(live picture)
   ?f2<-(snap picture)
=>
   (retract ?f1 ?f2)
   (snapfun)
   (assert (saved picture))
   (fprintout t " live picture was snaped " crlf))


(defrule set_up_window
   ?f1 <- (window set up)
=>
   (retract ?f1)
   (fprintout t " the name of the window is (type S1 , S2 or D):" crlf)
   (bind ?name (readline))
   (assert (set up window ?name))
   (fprintout t " the number of the toprow is (0 - 479):" crlf)
   (bind ?toprow(readline))
   (fprintout t " the number of the bottomrow is (1 - 480):" crlf)
   (bind ?bottomrow(readline))
   (fprintout t " the number of the leftcolumn is (0 - 639):" crlf)
   (bind ?leftcolumn(readline))
   (fprintout t " the number of the rightcolumn is (1 - 640):" crlf)
   (bind ?rightcolumn(readline))
```

```
    (assert (window ?name size ?toprow ?bottomrow ?leftcolumn
?rightcolumn ))
    (set_up_window_fun ?name ?toprow ?bottomrow ?leftcolumn
?rightcolumn)
   (fprintout t "Do you want to set up another window ? (yes/no)" crlf)
    (bind ?anwser(readline))
    (assert (another window anwser ?anwser)))



(defrule  test_another_window
     ?f1 <- (another window anwser ?anwser)
=>
    (retract ?f1)
    if( (eq ?anwser "yes")
     then
    (assert  (window set up))))



(defrule  fram_set_up
     (set up fram)
=>
     (fprintout t "select the window(s1, s2,or D):" crlf)
      (bind ?window(readline))
     (fprintout t "select the window(Hardware, F1, F2 or F3): crlf)
      (bind ?Fram_number(readline))
     (assert ( ?window from ?Fram_numbe ))
      (fram_set_up ?window ?Fram_numbe)
     (fprintout t "another to set up?" crlf)
      (bind ?reply(readline))
    (if (eq  ?reply "yes")
      then
       (assert (set up fram))))


(defrule  do_move
   ?f1 <- (move picture)
=>
    (retract ?f1)
    (doMove_fun))
```

```
(defrule  do_s1_RGBtoD_YIQ
   ?f1 <- ( transfer RGB to YIQ)
=>
    (retract ?f1)
   (do_RGBtoYIQ_fun))


(defrule  do_s1_YIQtoD_RGB
   ?f1 <- (transfer YIQ to RGB)
=>
   (retract ?f1)
  (do_YIQtoRGB_fun))


(derule  display_pic
  ?f2 <- (display s1 picture)
=>
  (fram_set_up_fun  D  Hardware)
  (doMove_fun))
```

Appendix F Bibliography

1 Apple Computer, Inc. *Inside Macintosh*, Vol 1-5, Reading Mass., Addison-Wesley, 1985 - 1988.

2 Chernicoff, Stephen, *Macintosh Revealed, Unlocking the Toolbox* Volume I, 2nd edition, Indianapolis, Hayden, 1987.

3 Chernicoff, Stephen, *Macintosh Revealed, Programming with the Toolbox* , Volume II, 2nd edition, Indianapolis, Hayden, 1987.

4 Mark, Dave and Reed, Cartwright, *Macintosh Programming Primer*, Reading, Massachusetts, Addison-Wesley, 1989.