Manual, Part III

EIKONA library and sources

Digital image processing package

Version 4.0

Thessaloniki 1997

2

The JPEG input/output routines were based in part on the work of the Independent JPEG Group.

The GIF, BMP, TGA input/output routines were base in part on the PBMPLUS toolkit.

The scanner interface was based in part on the TWAIN Toolkit, Release 1.6.
The TWAIN toolkit is distributed as is. The developer and the distributors of the TWAIN toolkit expressly disclaim all implied, express or statutory warranties including, without limitation, the implied warranties of merchantability, noninfringement of third party rights and fitness for a particular purpose. Neither the developers nor the distributors will be liable for damages, whether direct, indirect, special, incidental, or consequential as a result of the reproduction, modification, distribution or other use of the TWAIN toolkit.

Distributor :

Contents

1

Overview of EIKONA Library

1.1 Introduction

EIKONA is an advanced digital image processing library. It is implemented for both AT/386/486/Pentium or compatible computers under DOS or Microsoft Windows version 3.1 or higher, and for Unix under X-Windows release 4 or higher. The library contains more than 350 routines in the following areas (non-exhaustive list):

- Storage and retrieval of raw/TIFF/TGA/GIF/BMP/JPEG images. Mixing, overlay, display (superVGA) and printing (HPGL) of binary, greyscale and color images. Basic operations, binary operators, geometrical transformations, zooming, color transformations. Various noise generators.
- Various FFT algorithms, periodogram, correlation, convolution, DCT. Linear filtering, Wiener filters, median and related filters, adaptive filters. Histogram computation, display and equalization. Halftoning, pseudocoloring, interpolation.
- Image coding algorithms (e.g. run-length, READ, LZW, DCT coding).
- Edge, line and contour detectors/followers. Region segmentation, texture analysis. Object counting and labeling, feature measurements, contour approximations, pyramids, quadtrees, thinning. Binary and greyscale mathematical morphology.

Hardware requirements: IBM AT/386/486/Pentium with at least 2 MB RAM. A Hard Disk, a math coprocessor and 4-8 MB RAM are recommended for the Microsoft Windows version. No frame grabber is required (optional). Any Unix machine which supports the X-Windows system, release 4 or higher is required for the Unix version. At least 16 MB RAM are recommended in this case.

Software requirements: EIKONA library can be used for developing applications under DOS (version 5.0 or higher) or Microsoft Windows (version 3.1

or higher). X-Windows Release 4 or higher, Xt Toolkit and Motif version 1.2 or higher are required for development under Unix.

EIKONA can operate in three modes: as a library, as DOS interpreter and as a complete Microsoft Windows/Unix application (EIKONA for Windows/Unix). EIKONA can be linked to any program written in Microsoft compatible languages (preferably in Microsoft C). Both the library, the DOS interpreter and EIKONA for Windows are compiled in medium memory model. (*MS-Windows*)

The image processing library EIKONA is contained in the file EIKONA.LIB included in the floppy disk with the indication LIBRARY. This library can be linked to any program written in Microsoft compatible languages (preferably in Microsoft C) and compiled in medium memory model. It can also be used to create applications that run under Microsoft Windows ver. 3.1 or higher.

1.2 The Include file EIKONA.H

The include file EIKONA.H contains a number of type definitions. The most commonly used are the following type definitions:

```
typedef unsigned char far* far* image;
typedef float far* far* matrix;
typedef int far* far* imatrix;
typedef float far* vector;
```

Some other definitions are used in special cases:

```
/* Type definitions for edge following algorithms */
struct pixel {int row; int column; int cost;}
typedef struct pixel CPIXEL;
struct node { CPIXEL info; struct node far* link;}
typedef struct node NODE;

/* Type definitions for chain codes */
typedef unsigned char byte;
typedef struct CHNODE {unsigned char num; struct CHNODE *next;} CHNODE;
typedef struct CHAIN {CHNODE *start; CHNODE *end;} CHAIN;

/* Type definitions for polygon approximations */
typedef struct PIXEL {int x; int y;} PIXEL;
typedef struct LNODE {PIXEL num; struct LNODE *next;} LNODE;
typedef struct LIST {LNODE *start; LNODE *end;} LIST;

/* Type definitions for quadtrees */
typedef struct QNODE { unsigned char color;
```

```
              struct QNODE *father;
              struct QNODE *quad[4];
             } QNODE;

typedef image far* pyramid;

/* Type definitions for Huffman coding */
typedef struct symbol far* SPOINTER;

struct symbol {
           float prob;
           SPOINTER comp[2];
          };


/* Type definitions for LZW coding */
typedef struct llink far*LPOINTER;
struct llink {
 int n;
 LPOINTER next;
};
```

The subroutines in EIKONA library communicate to their environment through the parameters in their definition and through the following global parameters (all are of type int):

```
NMAX : vertical image size
MMAX : horizontal image size
FNMAX : vertical matrix size
FMMAX : horizontal matrix size
INMAX : vertical integer matrix size
IMMAX : horizontal integer matrix size
NWMAX: vertical window size
MWMAX: horizontal window size
NSIG : vector size
```

These global variables are defined in EIKONA.LIB. They are also defined as external in EIKONA.H. They are set by the subroutine init_eikona having arguments the desired values. It is recommended that the vector size NSIG to be 256 or larger.

It also contains all function prototypes used in the library. In the following a short description on how to use the EIKONA library is included.

1.3 How to use EIKONA library

1.3.1 Developing under MS–DOS

The following DOS application program uses the subroutines of EIKONA to count particles in the image contained in file PART256.DAT.

```
/* program in file count.c                          */
/* It counts particles in an image                  */
#include <stdio.h>
#include <graph.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(argc,argv)
int argc;
char *argv[];
{ int i,N;
/* define two image buffers  a, b                          */
  image a,b;
/* define one signal buffer h                              */
  vector h;
/* define the image buffer size  256x256,
   the window size  15x15 and the vector size 256         */
  init_eikona(256,256,15,15,256);
/* create two image buffers a, b of size  256x256         */
  a=build_image(256,256);
  b=build_image(256,256);
/* create a vector h of size 256                           */
  h=build_vector(256);
/* load image, initialize graphics and display image       */
  fromdisk(argv[1], a, 0, 0, 256, 256);
  init_graphics("VRES16COLOR");
  _settextposition(1,5);
  printf("original image");
 _settextposition(1,5);printf("original image");
  imdisp(a, 16, 16, 0, 0, 0, 256, 256);

/* take the image negative and display it                  */
  neg(a, b, 0, 0, 256, 256);
  _settextposition(1,40);
  printf("negative  image");
```

```
  imdisp(b, 16, 16, 256, 0, 0, 256, 256);

/* find negative image histogram                              */
  _settextposition(21,1);
  printf("histogram calculation starts...");
  hist(b, h, 0, 0, 256, 256);
  sigshow(h, 256, 10, 10, 400, 300);

/* threshold the image and display it                         */
  _settextposition(1,1);
  printf("negative image histogram (intensities 0-255)-choose a threshold:");
  scanf("%d",&i);
  clearscreen();
  thres(b, b, i, 0, 0, 256, 256);
  _settextposition(1,5);
  printf("original  image");
  imdisp(a, 16, 16, 0, 0, 0, 256, 256);
  _settextposition(1,40);
  printf("thresholded image");
  bindisp(b, 16, 16, 256, 0, 0, 256, 256);

/* perform counting and store the results on buffer h         */
  _settextposition(21,1);
  printf("counting starts....");
  for(i=0; i<256; i++) h[i]=(float)(0.0);
  count(b, &N, h, 0, 0, 256, 256);

/* display the results of counting and store them in a file */
  reset_graphics();
  _settextposition(1,1);
  printf("%d particles have been found", N);
  for(i=0; i<N; i++)
  printf("\n particle %2d has area %4.0f pixels", i, h[i]);
  printf("\n \n the results are stored in file RES.DAT");
  dump_signal(h,"RES.DAT");
  return(0);
}
```

After successful compilation and linking of COUNT.C in medium memory model the program COUNT.EXE can be executed by:

```
count part256.dat
```

A threshold is required during the execution of the program. You can use threshold 128.

1.3.2 Developing under MS–Windows

As it is generally known, developing under MS–DOS imposes severe memory limitations. On the other hand, image processing and analysis software packages usually require large amounts of memory to display and carry out computations on image buffers. Using the EIKONA library under the MS–Windows operating environment raises these limitations to a great extent. However, this increased flexibility results in increased complexity in your source code.

The example program DEMO, which is given below, can serve as an example of how to write programs for the MS–Windows environment. With some slight modifications, you can use it to develop your own programs that utilize the EIKONA library. The source code, DEMO.C, is given below:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "eikona.h"
#include "demo.h"

HANDLE hInst;
HCURSOR hOldCursor, hWait;
HDC hDC;
PBITMAPINFO pDibInfo, p24DibInfo;
LOGPALETTE *pPal;
HPALETTE hPal;
unsigned long PALETTESIZE;

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

matrix mat;
image ima[2];
HANDLE far * hima[2], far * hmat;

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg;

    if (!hPrevInstance)
        if (!InitApplication(hInstance))
            return(FALSE);

    if (!InitInstance(hInstance, nCmdShow)) return(FALSE);
```

```
    /* Initialize EIKONA buffer dimensions */
    init_eikona(-1, -1, -1, -1, -1);

    /* Allocate image buffer ima[0]... */
    if ((ima[0] = win_build_image(&(hima[0]), 256, 256)) == NULL) exit(-10);
    /* ...and store LENNA256.DAT in it */
    fromdisk("lenna256.dat", ima[0], 0, 0, 256, 256);

    /* Allocate the other buffers and the logical palette */
    if ((ima[1] = win_build_image(&(hima[1]), 256, 256)) == NULL) goto R1;
    if ((mat = win_build_matrix(&(hmat), 256, 256)) == NULL) goto R2;

    while (GetMessage(&msg, NULL, NULL, NULL)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);}

    return(msg.wParam);

    win_freef2(&(hmat), 256, 256);
    R2: win_freeuc2(&(hima[1]), 256, 256);
    R1: win_freeuc2(&(hima[0]), 256, 256);
    exit(-1);
}
/*************************************************************/
BOOL InitApplication(HANDLE hInstance)
{
    WNDCLASS wc;

    wc.style = NULL;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = NULL;
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = "DemoMenu";
    wc.lpszClassName = "DemoWClass";

    hWait = LoadCursor(NULL, IDC_WAIT);
    return(RegisterClass(&wc));
}
/*************************************************************/
```

```
BOOL InitInstance(HANDLE hInstance, int nCmdShow)
{
    HWND hWnd;

    hInst = hInstance;

    hWnd = CreateWindow("DemoWClass",
    "EIKONA library demo",
    WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME,
    10,
    10,
    264,
    302,
    NULL,
    NULL,
    hInstance,
    NULL);

    if (!hWnd) return(FALSE);

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    /* Create a bitmap that will be used  */
    /* to display the output images       */
    create_disp_bmp(hWnd, &pDibInfo, &p24DibInfo, &PALETTESIZE);
    pPal = alloc_logpalette(&hPal, PALETTESIZE);

    return(TRUE);
}
/*************************************************************/
long FAR PASCAL MainWndProc(HWND hWnd, unsigned message,
    WORD wParam, LONG lParam)
{
    FARPROC lpProcAbout;
    PAINTSTRUCT ps;

    switch (message) {
    case WM_COMMAND:
        if (wParam == IDM_ABOUT) {
            lpProcAbout = MakeProcInstance(About, hInst);
            DialogBox(hInst, "AboutBox", hWnd, lpProcAbout);
            FreeProcInstance(lpProcAbout);
            break;}
```

```
    /* Case of routine selection */
    if (wParam == IDM_THRESHOLD || wParam == IDM_ADDNOISE ||
    wParam == IDM_SOBEL) {
        /* Change the arrow cursor to an hourglass */
        /* cursor to indicate a lengthy operation  */
        SetCapture(hWnd);
        hOldCursor = SetCursor(hWait);
        switch (wParam) {
            case IDM_THRESHOLD:
                /* Perform thresholding, storing the */
                /* result in matrix mat               */
                thres(ima[0] ,ima[1], 100, 0, 0, 255, 255);
                break;

            case IDM_ADDNOISE:
                /* Add impulsive noise to ima[0] and */
                /* store the result in image ima[1]  */
                noise_imp(ima[0],ima[1],0.1f,0,255,0,0,255,255);
                break;

            case IDM_SOBEL:
                /* Perform Sobel edge detection and store */
                /* the resulting image in buffer ima[1]   */
                sobel(ima[0], ima[1], 0, 0, 255, 255);
                break;}

        /* Update the window, displaying the new image */
        InvalidateRect(hWnd, NULL, TRUE);

        /* Restore the cursor to its original shape */
        SetCursor(hOldCursor);
        ReleaseCapture();
        break;}
    else return(DefWindowProc(hWnd, message, wParam, lParam));

case WM_DESTROY:
    /* Free memory buffers */
    LocalUnlock(hPal);
    LocalFree(hPal);
    win_freef2(&(hmat), 256, 256);
    win_freeuc2(&(hima[1]), 256, 256);
    win_freeuc2(&(hima[0]), 256, 256);
    PostQuitMessage(0);
    break;
```

```
    case WM_PAINT:
        /* Repaint the client window */
        hDC = BeginPaint(hWnd, &ps);
        winbwimdisp(hWnd, pPal, pDibInfo, PALETTESIZE,
            ima[1], 0, 0, 0, 0, 255, 255, 1);
        EndPaint(hWnd, &ps);
        break;

    default:
        return(DefWindowProc(hWnd, message, wParam, lParam));}
    return(NULL);
}
/*************************************************************/
BOOL FAR PASCAL About(HWND hDlg, unsigned message, WORD wParam,
    LONG lParam)
{
    switch (message) {
    case WM_INITDIALOG:
        return(TRUE);
    case WM_COMMAND:
        if (wParam == IDOK || wParam == IDCANCEL) {
        EndDialog(hDlg, TRUE);
        return(TRUE);}
        break;}
    return(FALSE);
}
```

The module definition file, DEMO.DEF, is listed below:

```
NAME demo
DESCRIPTION 'EIKONA library demo'
EXETYPE WINDOWS
STUB 'WINSTUB.EXE'
CODE PRELOAD MOVEABLE
SEGMENTS
    IMPRO4_PRG    PRELOAD MOVEABLE
DATA PRELOAD MOVEABLE MULTIPLE
HEAPSIZE  4096
STACKSIZE 8192
EXPORTS
    MainWndProc @1
    About @2
```

The header file, DEMO.H, is given below:

```
#define IDM_ABOUT 100
#define IDM_THRESHOLD 101
#define IDM_ADDNOISE 102
#define IDM_SOBEL 103

int PASCAL WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
long FAR PASCAL MainWndProc(HWND, unsigned, WORD, LONG);
BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);
```

Finally, the resource compiler script, DEMO.RC, is listed below:

```
#include "windows.h"
#include "demo.h"

DemoMenu MENU
BEGIN
    POPUP "&Routines"
    BEGIN
        MENUITEM "&Threshold", IDM_THRESHOLD
        MENUITEM "A&dd noise", IDM_ADDNOISE
        MENUITEM "&Sobel edge detection", IDM_SOBEL
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "About Demo...", IDM_ABOUT
    END
END
AboutBox DIALOG 22, 17, 144, 75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About Demo"
BEGIN
    CTEXT "Eikona for Windows" -1, 0, 5, 144, 8
    CTEXT "Library Demo App" -1, 0, 14, 144, 8
    CTEXT "Version 3.0" -1, 0, 34, 144, 8
    DEFPUSHBUTTON "OK" IDOK, 53, 59, 32, 14, WS_GROUP
END
```

We can obtain an executable code of the program by building a Windows application using Microsoft Visual C++ 1.51. In order to do so we choose Windows application (.EXE) as Project Type in the Project submenu of the Options menu, then choose Release for the Build Mode. The CPU for Code Generation in the Compiler submenu should be set to 80386 and the Disable Stack Checking option should not be set. We choose None in the Debug Information option as well as in

the Browser Information option. The Memory Model should be set to Medium and the Optimizations to Default. Finally, the Distinguish Letter Case check box in the Input category of the Linker submenu should not be set. In the Project menu we choose New, set project type to Windows application (.EXE), and then we choose a name for our project file. In the Edit window we add the source file, the demo.def and demo.rc files, and the eikona.lib, libjpeg.lib and other.lib libraries. To compile the project file, we choose Build in the Project menu. The test program can then be run from the command line.

To run DEMO, enter :

```
win demo.exe
```

at the MS–DOS prompt. Alternatively, start MS–Windows and select **File, Run** from the program manager menu bar. You should note that it is a requirement for the grayscale image LENNA256.DAT to be present in the directory where DEMO.EXE resides.

A DLL library of EIKONA, called `EIKONA.DLL`, is also available. It can be used for developing applications under Microsoft Windows (version 3.1 or higher). A user application can use this DLL by linking the program with the corresponding Import Library, called `EIKONAL.LIB`.

`EIKONA.DLL` contains the functions of the `EIKONA.LIB` library except the following functions which concern Huffman coding:

```
treeaccess()
hufcod()
sendToOutput()
huftree()
initialize()
find2min()
merge()
decode_huffman()
reconstr_tree()
```

Besides, `EIKONA.DLL` contains two additional functions, namely `set_parameters()` and `get_parameters()` for access of the global parameters of the EIKONA library.

1.3.3 Developing under X–Windows

Developing under Unix is easier than developing under MS-DOS, mainly because there are no memory limits imposed. However, Graphical User Interface development is more complicated in Unix machines, because the X–Windows platform provides only low level tools for that purpose. Therefore, a number of high-level tools have been developed, such as OSF/Motif and Xt-Toolkit.

The example program demo.c given below can serve as an example of how to write programs for the X–Windows environment using the Motif library. With some slight modifications, you can use it to develop your own programs that utilize the EIKONA library. The source code, demo.c is given below:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <math.h>
#include <Xm/Xm.h>
#include <X11/Xlib.h>
#include <Xm/DrawingA.h>
#include <Xm/ScrolledW.h>

#include "eikona.h"

Widget shell;

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;


/*-----------------------------------------------------------------*/
/* routine to remove a DialButton from a Dialog */
void ScrubDial(Widget w, int dial)

{
  Widget remove;
  remove = (Widget) XmMessageBoxGetChild(w, dial);
  XtUnmanageChild(remove);
}


/*-----------------------------------------------------------------*/
void CreateErrMessage(char *title_str, char *text, Widget parent)
{
  Widget dialog;
  XmString title;

  title = XmStringCreateSimple(title_str);  /* puts title bar */
    dialog = (Widget)XmCreateErrorDialog ( parent, "display", NULL, 0 );
    XtVaSetValues ( dialog, XmNdialogStyle, XmDIALOG_APPLICATION_MODAL,
    XmNdialogTitle, title, XtVaTypedArg, XmNmessageString, XmRString,
```

```
    text, strlen ( text )+1, NULL );

  /* removes the "cancel" and "help" buttons */
  ScrubDial(dialog, XmDIALOG_CANCEL_BUTTON);
  ScrubDial(dialog, XmDIALOG_HELP_BUTTON);

  XtManageChild(dialog);
}


/*--------------------------------------------------------------------*/
void redraw_BW(Widget widget, XImage *im, XmDrawingAreaCallbackStruct *cbk)
{
  Display *display;
  Window win;
  int screen;
  int width, height; /* image size */

  win = XtWindow(widget);
  display = XtDisplay(widget);
  screen = DefaultScreen(display);

  XMapRaised(display, win);

  width = im->width;
  height = im->height;

  XPutImage(display, win, DefaultGC(display, screen),
    im, 0, 0, 0, 0, width, height);
}


/*--------------------------------------------------------------------*/
Widget disp_grayscale(image a, Display * display, Widget parent,
  int width, int height)
{
  Widget top_shell, scroll, drawarea;
  XColor color;
  int i, k, j;
  char title[20];
  int display_width, display_height;
  unsigned char *data;
  unsigned int depth;
  Colormap colormap;
```

```
GC gc;
Visual *visual;
int screen;
Window root;
XImage * im;

/* get default visual, screen and GC for grayscale display */
screen = DefaultScreen(display);
visual = DefaultVisual(display, screen);
gc = DefaultGC(display, screen);
depth = DefaultDepth(display, screen);

root = RootWindow (display, screen);

/* create the grayscale colormap */
colormap = XCreateColormap(display, root, visual, AllocAll);
color.flags = DoRed | DoGreen | DoBlue;

for (i=0; i<256; i++)
{
  color.pixel = (unsigned long) i; /* cell number to be changed */
  color.red   = i << 8;
  color.green = i << 8;
  color.blue  = i << 8;

  XStoreColor(display, colormap, &color);
}

XInstallColormap(display, colormap);

if ( width > DisplayWidth(display, screen) )
  display_width = DisplayWidth(display, screen) - 50;
else
  display_width = width;

if ( height > DisplayHeight(display, screen) )
  display_height = DisplayHeight(display, screen) - 50;
else
  display_height = height;

strcpy(title, "Test Image Display");

/* create the top shell */
top_shell = XtVaCreatePopupShell("top_shell", topLevelShellWidgetClass,
```

```
    parent, XmNtitle, title, XmNwidth, (display_width)+4, XmNheight,
    (display_height)+4, XmNmaxWidth, (display_width)+4, XmNmaxHeight,
    (display_height)+4, NULL);

  data = (unsigned char *)malloc(width*height*sizeof(unsigned char));

  /* convert 2-D (image) to 1-D */
  k=0;
  for (i=0; i<width; i++)
    for (j=0; j<height; j++)
    {
      data[k] = a[i][j];
      k++;
    }

  /* create the XImage structure */
  im = XCreateImage (display, visual, depth, ZPixmap, 0,
    (unsigned char *)data, width, height, 8, 0);

  /* create scrolled window widget to contain drawing area */
  scroll = XtVaCreateManagedWidget("scroll", xmScrolledWindowWidgetClass,
top_shell, XmNscrollingPolicy, XmAUTOMATIC, XmNscrollBarDisplayPolicy,
XmAS_NEEDED, XmNvisualPolicy, XmCONSTANT, XmNspacing, 0, NULL);

  /* create drawing area widget to contain XImage */
  drawarea = XtVaCreateManagedWidget("drawarea", xmDrawingAreaWidgetClass,
scroll, XmNwidth, width, XmNheight, height, XmNbackground,
        BlackPixel(display, screen), XmNcolormap, colormap,
        XmNdepth, depth, XmNvisual, visual, NULL);

  /* add a callback for exposure events */
  XtAddCallback(drawarea, XmNexposeCallback, (XtCallbackProc) redraw_BW,
    (XtPointer) im);

  XtPopup(top_shell, XtGrabNone);
  XFreeColormap(display, colormap);

  return top_shell;
}



/*****************************************************************/
int main (int argc, char **argv)
```

```
{
  XtAppContext app_context;
  Display *display;
  Widget canvas;
  String app_name;
  int ret;
  image a, b;
  int width, height;

  width=atoi(argv[2]);
  height=atoi(argv[3]);

  NMAX = width;
  MMAX = height;

  /* allocate the images according to size */
  a = matuc2(width, height);
  b = matuc2(width, height);

  ret = fromdisk(argv[1], a, 0, 0, width, height);

  if (ret == -300)
  {
    printf("Invalid filename specified!\n");
    exit(-1);
  }

  XtToolkitInitialize();
  app_context = XtCreateApplicationContext();

  display = XtOpenDisplay (app_context, NULL, argv[0], "XApplication",
NULL, 0, &argc, argv);
  if (!display)
  {
    printf("%s: can't open display, exiting...\n", argv[0]);
    exit (-1);
  }

  shell = XtAppCreateShell ( app_name, "XApplication",
    applicationShellWidgetClass, display, NULL, 0 );

  /* initial image */
  canvas = disp_grayscale(a, display, shell, width, height);
```

```
  /* perform edge detection */
  laplace(a, b, 0, 0, width, height);

  /* edge detected image */
  canvas = disp_grayscale(b, display, shell, width, height);

  XtAppMainLoop (app_context);

  mfreeuc2(a, width);
  mfreeuc2(b, width);

  exit (0);
}
```

The executable code of the program can be obtained by compiling it using cc, the standard C compiler for Unix machines. The Xlib, and Motif libraries must be linked as well. In order to do so, we type the following in a Unix shell:

```
cc -Ae -D_UNIX -O2 -I/usr/include/Xm format_lib.o -o demo demo.c
-L/usr/lib -L./ -lXm -lXt -lX11 -lm -ljpeg -leik
```

The switch $-D_UNIX$ indicates that the program is compiled in a Unix environment. After the -I switch we must type the full path of the include files used. Also after the -L switch we must include the full path of the library files used during the link procedure.

In order to run the demo, type "demo" in a Unix shell. Note that the object file $format_lib.o$ which includes various image format manipulation routines, as well as library files libeik.a and libjpeg.a must reside in the same directory. If not, we must include the directory they reside after the -L switch in the command line.

1.4 How to extend EIKONA library

A user can easily write its own image processing routines and integrate it in EIKONA library. In the following an image transposition routine is found. It can be used to rotate an image by -90 degrees. Repeated use of this routine can rotate an image by -180 and -270 degrees.

```
/* Subroutine in file transp.c                */
#include <stdio.h>
#include "eikona.h"
```

```
int transp(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

/* subroutine to transpose an image buffer
   a: source image buffer
   b: destination image buffer
   N1,M1: upper left corner coordinates
   N2,M2: lower right corner coordinates       */
{ int k,l;
/* error handling                                              */
  if (a==NULL || b==NULL) return(-1);
  if (a==b) return(-9999);
  if (N1<0 || M1<0 || N2>NMAX || M2>MMAX || N1>N2 || M1>M2)
     return (-21);
/* image transposition                                        */
  for(k=N1;k<N2;k++)
   for(l=M1;l<M2;l++)
    b[k][l]=a[l][k];
  return(0);
}
```

Pay attention to the error handling in this routine and compare it to the error messages in the corresponding section of this manual. This subroutine can be called by the following main program:

```
/* program in file main.c                        */
/* It transposes an image                        */
#include <stdio.h>
#include <graph.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

extern int transp(image a,image b, int N1,int M1,int N2,int M2);

main(argc,argv)
int argc;
char *argv[];
{
/* define two image buffers  a, b                          */
  image a,b;
/* define the image buffer size  256x256,
```

```
   the window size  15x15 and the vector size 256          */
  init_eikona(256,256,15,15,256);
/* create two image buffers a, b of size  256x256          */
  a=build_image(256,256);
  b=build_image(256,256);
/* load image, initialize graphics and display image       */
  fromdisk(argv[1], a, 0, 0, 256, 256);
  init_graphics("VRES16COLOR");
  _settextposition(1,5);
  printf("original image");
 _settextposition(1,5);printf("original image");
  imdisp(a, 16, 16, 0, 0, 0, 256, 256);

/* transpose the image and display it                     */
  transp(a, b, 0, 0, 256, 256);
  _settextposition(1,40);
  printf("transposed  image");
  imdisp(b, 16, 16, 256, 0, 0, 256, 256);
  return(0);
}
```

After successful compilation and linking of MAIN.C and TRANSP.C in medium memory model this program can be executed by:

```
main lenna256.dat
```

Since no reset_graphics routine is used in main, the video mode remains VRES16COLOR after the termination of its execution. The object code TRANSP.OBJ can be incorporated in the EIKDOS.LIB library in the following way:

```
lib eikdos + transp,, eikdos
```

and can be used as integral part of EIKONA afterwards.

2

Routines of EIKONA

2.1 Subroutine description

The following information is written for each subroutine:

```
name
function

LIBRARY MODE:
   definition
   type of parameters

   parameter explanation

INTERPRETER MODE:
   call
   type of parameters in interpreter mode
```

MS-WINDOWS ENVIRONMENT: Sequence of menu choices to reach the desired function

X-WINDOWS ENVIRONMENT: Sequence of menu choices to reach the desired function

SHORT DESCRIPTION: The detailed description of most routines described in this manual can also be found in [PIT90], [PIT93].

RETURN VALUE:

SEE ALSO: (Relevant subroutines)

Most subroutines return 0 in normal operation. In cases of errors they return negative numbers. Their meaning is described in section on errors. The interpreter mode corresponds to the way the subroutines are called in the DOS interpreter.

It also corresponds with the way the subroutines are used within EIKONA for Windows. In the second case, the coordinates of the region of interest (N1,M1), (N2,M2) are passed to the subroutine by the EIKONA environment (through ROI setting by mouse) and they are not given explicitly by the user. The color image buffers (refer to the manual of EIKONA for windows) refer to the r,g,b image buffers with by a single number.

Most of the routines that are described in the manual are described also in full detail either in [PIT93] or in [PIT90] where the interested reader can find more information.

References

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[PIT93] I.Pitas, editor, *Parallel algorithms for digital image processing, computer vision and neural networks*, J. Wiley, 1993.

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[AND77] H.C.Andrews, B.R.Hunt, *Digital image restoration*, Prentice Hall, 1977.

[ANG90] E.Angel, *Computer graphics*, Addison-Wesley, 1990.

[BAL82] D.H.Ballard, C.M.Brown, *Computer vision*, Prentice Hall, 1982.

[FOL90] J.D.Foley, A.van Dam, S.K.Feiner, J.F.Hughes, *Computers graphics: Principles and practice*, Addison-Wesley, 1990.

[GON87] R.C.Gonzalez, P.Wintz, *Digital image processing*, Addison-Wesley, 1987.

[HAR87] S.Harrington, *Computer graphics: A programming approach*, McGraw-Hill, 1987.

[JAI89] A.K.Jain, *Fundamentals of digital image processing*, Prentice Hall, 1989.

[LEV85] M.D.Levine, *Vision in man and machine*, McGraw-Hill, 1985.

[LIN91] C.A.Lindley, *Practical image processing in C*, Wiley, 1991.

[MIC87] *Microsoft C: Runtime library reference*, Microsoft Press, 1987.

[NIB86] W.Niblack, *Digital image processing*, Prentice Hall, 1986.

[PRA91] W.K.Pratt, *Digital image processing*, Wiley, 1991.

[PRE88] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling, *Numerical recipes in C*, Cambridge University Press, 1988.

[RIM90] S.Rimmer, *Bit-mapped graphics*, Windcrest, 1990.

[ROS82] A.Rosenfeld, A.C.Kak, *Digital picture processing*, Academic Press, 1982.

[SCH89] R.J.Schalkof, *Digital image processing and computer vision*, Wiley, 1989.

[SER82] J.Serra, *Image analysis and mathematical morphology*, Academic Press, 1982.

[WIL87] R.Wilto, *Programmer's guide to PC and PS/2 video systems*, Microsoft Press, 1987.

[WYZ67] G.W.Wyzecki, W.S.Stiles, *Color science*, Wiley, 1967.

3

Memory allocation routines

3.1 addchnode()

Subroutine to add a node to a chain
LIBRARY MODE

```
CHNODE far* addchnode(val, end)
byte val;
CHNODE far* end;

val: value of node to be added
end: end of list
```

SHORT DESCRIPTION: This subroutine adds a node to a chain. It returns the new end of chain.
RETURN VALUE: chain node
SEE ALSO: getchnode

3.2 add_list()

Subroutine to add a node to a pixel list
LIBRARY MODE:

```
LNODE far* add_list(val, end)
PIXEL val; LNODE far* end;

val: value of node to be added
end: end of list
```

SHORT DESCRIPTION: This subroutine adds a node to a pixel list.
RETURN VALUE: LNODE (new end of list)
SEE ALSO: getnode

3.3 build_image()

Subroutine to create image buffers
LIBRARY MODE:

```
unsigned char far* far* build_image(N,M)
int N,M;
```

SHORT DESCRIPTION: Function to create an image buffer of size NxM. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: build_matrix(), build_imatrix()

3.4 build_vector()

Subroutine to create a vector buffer
LIBRARY MODE:

```
float far* build_vector(N)
int N;
```

SHORT DESCRIPTION: Function to create a vector buffer of size N. If N is less than or equal to 0, a vector buffer of size NSIG is created.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: build_image(), build_matrix(), build_imatrix()

3.5 build_matrix()

Subroutine to create a matrix (floating point) buffer
LIBRARY MODE:

```
float far* far* build_matrix(N,M)
int N,M;
```

SHORT DESCRIPTION: Function to create a matrix buffer of size NxM. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created. Its return value is of matrix type (float far* far*). Variables of matrix type must be created by using build_matrix before their use. If insufficient memory is available, the function returns NULL. The subroutine init_eikona that determines the size of matrices must be used in advance.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: build_image(), build_imatrix()

3.6 build_imatrix()

Subroutine to build a 2d integer buffer
LIBRARY MODE:

```
int far* far* build_imatrix(i,j)
int i,j;
```

SHORT DESCRIPTION: Function to create an integer buffer of size NxM. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created. Its return value is of imatrix type (int far* far*). Variables of imatrix type must be created by using build_imatrix before their use. If insufficient memory is available, the function returns NULL. RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: build_image(), build_matrix()

3.7 deletel()

Subroutine to delete a node from end of a pixel list
LIBRARY MODE:

```
delete(p, end)
LNODE far* p;
LNODE far* end;
```

SHORT DESCRIPTION: Subroutine to delete NODE *p from end of a pixel list
RETURN VALUE: 0
SEE ALSO: freelist

3.8 deletech()

Subroutine to delete a node from end of chain .
LIBRARY MODE:

```
deletech(p, end)
CHNODE far* p;
CHNODE far* end;
```

SHORT DESCRIPTION: This subroutine deletes NODE *p from end of chain.
RETURN VALUE: 0
SEE ALSO: addchnode, getchnode

3.9 freelist()

Subroutine to free all allocated pixel list space
LIBRARY MODE:

```
void freelist(l)
LIST l;
```

SHORT DESCRIPTION: This subroutine frees all allocated pixel list space.
RETURN VALUE:
SEE ALSO: deletel, freechain

3.10 freechain()

Subroutine to free all allocated chain node space
LIBRARY MODE:

```
void freechain(ch)
CHAIN ch;
```

SHORT DESCRIPTION: This subroutine frees all allocated chain node space
RETURN VALUE:
SEE ALSO:freelist, deletel

3.11 get_parameters()

Subroutine to get the EIKONA library parameters.
LIBRARY MODE:

```
int get_parameters( pnmax, pmmax, pnwmax,pmwmax, pnsig,
   pinmax,pimmax, pfnmax,pfmmax)
int  *pnmax, *pmmax, *pnwmax, *pmwmax, *pnsig;
int *pinmax,*pimmax, *pfnmax, *pfmmax;
```

SHORT DESCRIPTION: Subroutine to retrieve the values of the EIKONA parameters: maximum image size (*pnmax=NMAX, *pmmax=MMAX), maximum window size (*pnwmax=NWMAX, *pmwmax=MWMAX), maximum signal size (*pnsig=NSIG), maximum integer matrix size (*pinmax=INMAX, *pimmax=IMMAX) and maximum float matrix size (*pfnmax=FNMAX, *pfmmax=FMMAX). If an argument of this fuction is a NULL pointer, the function does not set the corresponding value for this pointer.
RETURN VALUE: 0
SEE ALSO: `set_parameters()`

3.12 getnode()

Subroutine to allocate space for a LNODE
LIBRARY MODE:

```
LNODE far* getnode()
```

SHORT DESCRIPTION: This subroutine allocates space for a pixel list node.
RETURN VALUE: a node LNODE
SEE ALSO:add_list

3.13 getchnode()

Subroutine to allocate space for a CHNODE
LIBRARY MODE:

```
CHNODE far* getchnode()
```

SHORT DESCRIPTION: This subroutine allocates space for a chain element.
RETURN VALUE: chain element
SEE ALSO: addchnode

3.14 init_eikona()

Subroutine to initialize EIKONA
LIBRARY MODE:

```
int init_eikona(nmax,mmax,nwmax,mwmax,nsig)
int nmax,mmax,nwmax,mwmax,nsig;
```

SHORT DESCRIPTION: Subroutine to give initial values to maximum image size (nmax, mmax), maximum window size (nwmax, mwmax) and signal size nsig.
RETURN VALUE: 0

3.15 list_decode()

Subroutine to display a list on an image buffer.
LIBRARY MODE:

```
int list_decode(image,LIST)
image a;
LIST list;

a: output image buffer
list: input list
```

SHORT DESCRIPTION: This subroutine displays all list nodes of a list on an image buffer.
RETURN VALUE: 0, -1, -2
SEE ALSO: list_connect

3.16 list_connect()

Subroutine to connect list nodes with straight lines and displays them to an image buffer
LIBRARY MODE:

```
int list_connect(image,LIST)
image a;
LIST list;

a: output image buffer
list: input list
```

SHORT DESCRIPTION: This subroutine connects list nodes with straight lines and prints them to an image buffer.
RETURN VALUE: 0, -1, -2
SEE ALSO: list_decode

3.17 matuc2()

Subroutine to allocate a 2d unsigned char array
LIBRARY MODE:

```
unsigned char far* far* matuc2(i,j)     internal use only
unsigned int i,j;

i: number of rows
j: number of columns
```

SHORT DESCRIPTION: Function to create a 2-d matrix of size ixj and of type unsigned char. It can be used to create image buffers of dimensions different than NMAXxMMAX. However the user must be vary conscious about using buffers of unequal dimensions as arguments in the subroutines of this library. They may return errors. Careful study of a subroutine is needed before its use in this case. !!!!!
RETURN VALUE: It returns NULL if unsuccessful.

3.18 mfreeuc2()

Subroutine to free an unsigned char buffer
LIBRARY MODE:

```
void mfreeuc2(m,i)              internal use only
unsigned char far* far* m;
unsigned int i;

m: array name
i: number of rows
```

SHORT DESCRIPTION: Subroutine to free a 2-d array of type unsigned char. This array must have been created by either matuc2(), or build_image().
SEE ALSO: mfreef2(), mfreein2()

3.19 matsc2( )

Subroutine to allocate a 2d signed char array.

LIBRARY MODE:

```
signed char far* far* matsc2(i, j)
int i,j;
i: number of rows
j: number of columns
```

SHORT DESCRIPTION : This function creates a 2d matrix (by using _fmalloc) of size i×j and of type signed char.
RETURN VALUE : NULL if unsuccesfull.
SEE ALSO : mfreesc2( )

3.20 matf1( )

Subroutine to allocate 1d float buffer.

LIBRARY MODE:

```
float far* matf1(i)
int i;
i: number of array elements;
```

SHORT DESCRIPTION : This function allocates an 1D float buffer of size i.
RETURN VALUE : NULL if unsuccesfull.
SEE ALSO : mfreef1( )

3.21 mfreesc2( )

Subroutine to free a 2d signed char buffer.

LIBRARY MODE:

```
void mfreesc2(m , i)
signed char far* far* m;
int i;
m: array name
i: number of rows
```

SHORT DESCRIPTION : Function to free a 2d array of type signed char.
SEE ALSO : matsc2( ).

3.22 mfreef1( )

Subroutine to free a float 1d buffer.

LIBRARY MODE:

```
void mfreef1(m)}
float far* m;
m: float array name
```

SHORT DESCRIPTION : Function to free a 1d array of type float.
SEE ALSO : matf1( ).

3.23 matf2()

Subroutine to allocate a 2d float array
LIBRARY MODE:

```
float far* far* matf2(i,j)       internal use only
unsigned int i,j;
i: number of rows
j: number of columns
```

SHORT DESCRIPTION: Function to create a 2-d matrix of size ixj and of type float. It can be used to create image buffers of dimensions different than NMAXxMMAX. However the user must be very conscious about using buffers of unequal dimensions as arguments in the subroutines of this library. They may return errors. Careful study of a subroutine is needed before its use in this case.
RETURN VALUE: It returns NULL if unsuccessful.

3.24 mfreef2()

Subroutine to free a 2d float array
LIBRARY MODE:

```
void mfreef2(m,i)        internal use only
float far* far* m;
unsigned int i;

m: array name
i: number of rows
```

SHORT DESCRIPTION: Subroutine to free a 2-d array of type float. This array must have been created by matf2().
SEE ALSO:

```
mfreeuc2(), mfreein2()
```

3.25 matin2()

Subroutine to allocate a 2d integer array
LIBRARY MODE:

```
int far* far* matin2(i,j)
unsigned int i,j;
i: number of rows
j: number of columns
```

SHORT DESCRIPTION: Function to create a 2-d array of size ixj and of type integer. It can be used to create buffers of dimensions different than NMAXxMMAX. However the user must be vary conscious about using buffers of unequal dimensions as arguments in the subroutines of this library. They may return errors. Careful study of a subroutine is needed before its use in this case. !!!!!
RETURN VALUE: It returns NULL if unsuccessful.

3.26 mfreein2()

Subroutine to build a 2d integer buffer
LIBRARY MODE:

```
void mfreein2(m,i)
int far* far* m;
unsigned int i;

m: array name
i: number of rows
```

SHORT DESCRIPTION: Subroutine to free a 2-d array of type integer. This array must have been created by matin2() or build_imatrix().
SEE ALSO: mfreeuc2(), mfreef2()

3.27 push()

Subroutine to push an element to stack
LIBRARY MODE:

```
void push(start,x)
NODE far** start;
CPIXEL x;

start: pointer to stack
x: element
```

SHORT DESCRIPTION: This subroutine pushes a stack element to stack. If a node cannot be allocated return -9.
RETURN VALUE: 0, -9.
SEE ALSO: pop

3.28 pop()

Subroutine to pop a stack element from stack.
LIBRARY MODE:

```
CPIXEL pop(start)
NODE far** start;

start: stack
```

SHORT DESCRIPTION: This subroutine pops a stack element from stack. If stack is NULL it exits with -111.
RETURN VALUE: The first stack element.
SEE ALSO: push

3.29 searchch()

Subroutine to search a chain
LIBRARY MODE:

```
CHNODE far* searchch(x, ch)
byte x;
CHAIN ch;

x: value
ch: list
```

SHORT DESCRIPTION: This subroutine searches for value "x" in chain "ch". It returns a pointer to node of list where "x" was found. It returns NULL if "x" not in "ch" .

RETURN VALUE: chain node
SEE ALSO: addchnode, getchnode

References

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

4

Display routines under DOS

4.1 build_palette()

Subroutine to build a palette buffer
LIBRARY MODE:

```
unsigned long far* build_palette()
```

```
p: palette buffer
```

SHORT DESCRIPTION: This subroutine allocates a palette buffer having 256 unsigned long elements.
RETURN VALUE: pointer, NULL
SEE ALSO: load_palette()

4.2 bindisp()

Subroutine to display a binary image
LIBRARY MODE:

```
int bindisp(a, cn, NS,MS, N1,M1,N2,M2)
```

```
image a;
int cn;
int NS,MS;
int N1,M1,N2,M2;

a: image buffer
cn: display intensity
NS,MS: screen starting point
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
bindisp a cn [NS MS N1 M1 N2 M2]
a: integer
```

SHORT DESCRIPTION: This subroutine displays a binary image stored on buffer a on CGA, EGA, VGA monitors. cn denotes the intensity of display. The image on buffer a must be binary. The subroutine checks if the image is binary and if this is not the case, it returns an error value. NS, MS denote the screen display starting point. Both the intensity and the display size depend on the display mode used.
RETURN VALUE: 0, -1, -21, -23, -60, -100
SEE ALSO: init_graphics, imdisp, overdisp, thres

4.3 clearscreen()

Subroutine to clear the entire screen
LIBRARY MODE:

```
int clearscreen()
```

INTERPRETER MODE:

```
clearscreen
```

SHORT DESCRIPTION: This subroutine clears the entire screen.
RETURN VALUE: 0
SEE ALSO: clear

4.4 coeff()

Subroutine to load filter coefficients on a buffer
LIBRARY MODE:

```
int coeff(h,N)
vector h;
int N;

h: coefficient array
N: number of coefficients
```

INTERPRETER MODE:

```
coeff h N
h: integer (0 or 1)
```

SHORT DESCRIPTION: This subroutine is used to load filter coefficients interactively on an one-dimensional array h. The number of coefficients to be loaded is N. It is used mainly in the interpreter mode.
RETURN VALUE: 0, -2, -50
SEE ALSO: l_filter

4.5 create_palette()

Subroutine to create palette for an RGB image
LIBRARY MODE:

```
int create_palette(r,g,b,p,N1,M1,N2,M2)
image r;
image g;
image b;
unsigned long far* p;
int N1,M1;
int N2,M2;

r: input R image buffer
g: input G image
b: input B image
p: palette buffer
N1,M1: upper left corner coordinates
N2,M2: lower right corner coordinates
```

SHORT DESCRIPTION: This subroutine creates palette for an RGB image.
RETURN VALUE: 0, -1, -21
SEE ALSO: default_palette()

4.6 default_palette()

Subroutine to create a palette buffer
LIBRARY MODE:

```
int default_palette(p,number)
unsigned long far * p;
int number;

p: palette buffer
number: 0 BW
        1 (2 bit red, 3 bit green, 3 bit blue)
        2 (3 bit red, 2 bit green, 3 bit blue)
        3 (3 bit red, 3 bit green, 2 bir blue)
```

SHORT DESCRIPTION: This subroutine creates 4 different default palette buffers (1 BW, 3 color buffers.
RETURN VALUE: 0, -2, -90,
SEE ALSO: build_palette(), load_palette()

4.7 extvgadisp()

Subroutine to display RGB images on a VGA card for the extended mode 640x400
LIBRARY MODE:

```
int extvgadisp(a,NS,MS, N1,M1,N2,M2,VideoCard)
image a;
int NS,MS;
int N1,M1,N2,M2;
```

```
int VideoCard;

a    : buffer
NS,MS: screen starting point (integers)
N1,M1: buffer display starting point
N2,M2: buffer display end point
Videocard: 1 (Paradise), 2 (ATI)
```

SHORT DESCRIPTION: This subroutine displays RGB images on a VGA card for the extended mode 640x400. It supports ATI and Paradise VGA cards.
RETURN VALUE: -1, -21
SEE ALSO: imdisp, vgadisp, extbindisp

4.8 extdisp()

Subroutine to display images on a VGA card for the extended mode 640x400
LIBRARY MODE:

```
int extdisp(a,NS,MS, N1,M1,N2,M2,VideoCard)
image a;
int NS,MS;
int N1,M1,N2,M2;
int VideoCard;

a    : buffer
NS,MS: screen starting point (integers)
N1,M1: buffer display starting point
N2,M2: buffer display end point
Videocard: 1 (Paradise), 2 (ATI)
```

SHORT DESCRIPTION: This subroutine display images on a VGA card for the extended mode 640x400.It supports ATI and Paradise VGA cards.
RETURN VALUE: 0, -1, -10, -21
SEE ALSO: extvgadisp, extbindisp

4.9 extbindisp()

Subroutine to display binary images on a VGA monitor for the extended mode 640x400
LIBRARY MODE:

```
int extbindisp(a,cn,NS,MS, N1,M1,N2,M2,VideoCard)
image a;
unsigned char cn;
int NS,MS;
int N1,M1,N2,M2;
int VideoCard;

cn: display intensity
a    : buffer
NS,MS: screen starting point (integers)
N1,M1: buffer display starting point
N2,M2: buffer display end point
Videocard: 1 (Paradise), 2 (ATI)
```

SHORT DESCRIPTION: This subroutine displays binary images on a VGA monitor for the extended mode 640x400 It supports ATI and Paradise VGA cards.
RETURN VALUE: 0, -1, -21, -100
SEE ALSO: extdisp

4.10 getvideomode()

Subroutine that returns the current video mode number (internal use only)
LIBRARY MODE:

```
int getvideomode()
```

SHORT DESCRIPTION: This subroutine returns the current video mode number by using the interrupt 0x10.
RETURN VALUE: regs.h.al
SEE ALSO: init_graphics, video_mode

4.11 imdisp()

Subroutine to display an image on the screen
LIBRARY MODE:

```
int imdisp(a, cn, NS,MS, N1,M1,N2,M2)
image a;
int cn;
int NS,MS;
int N1,M1,N2,M2;

a: image buffer
cn: number of gray levels (2<=cn<=64)
NS,MS: screen starting point
N1,M1: buffer display starting point
N2,M2: buffer display end point
```

INTERPRETER MODE:

```
imdisp a cn [NS MS N1 M1 N2 M2]
a: integer
```

SHORT DESCRIPTION: This subroutine displays a grayscale image stored on buffer a on CGA, EGA, VGA monitors. cn denotes the number of display intensities. NS, MS denote the screen display starting point. Both the intensities number and the display size depend on the display mode used. For more details on the allowable display dimensions see the description of the subroutine init_graphics or refer to the MICROSOFT C, RUN-TIME LIBRARY REFERENCE (subroutine _setvideomode).
RETURN VALUE: 0, -1, -21, -23, -60
SEE ALSO: init_graphics, bindisp, overdisp, thres

4.12 init_graphics()

Subroutine to initialize graphics adapter
LIBRARY MODE:

```
int init_graphics(videomode)
char videomode[];
```

videomode: video mode (string)
INTERPRETER MODE:

`init_graphics videomode`

SHORT DESCRIPTION: This subroutine initializes the display modes on CGA, EGA or VGA graphics cards and monitors. It can also be used on Hercules graphics card, if proper CGA simulation has been performed, e.g. by using MAGICKEY. A VGA adapter and monitor are recommended for most applications. The videomodes supported are the following:

```
DEFAULTMODE,    TEXTBW40,      TEXTC40,       TEXTBW80,
TEXTC80,        MRES4COLOR,    MRESNOCOLOR,   HRESBW,
TEXTMONO,       MRES16COLOR,   HRES16COLOR,   ERESCOLOR,
ERESNOCOLOR,    VRES2COLOR,    VRES16COLOR,   MRES256COLOR
```

For more information on the resolution and graylevels obtained by those modes, please refer to the MICROSOFT C MANUAL (RUN-TIME LIBRARY REFERENCE). The most usefull modes on a VGA adapter are the following:

mode	resolution (HxV)	colors
VRES16COLOR	640x480	16
MRES256COLOR	320x200	256 colors or 64 gray shades

The following modes are useful on an EGA graphics adapter:

mode	resolution (HxV)	colors
MRES16COLOR	320x200	16
HRES16COLOR	640x200	16
ERESNOCOLOR	640x350	2
ERESCOLOR	640x350	16

The following modes are useful for CGA graphics adapter:

mode	resolution (HxV)	colors
MRES4COLOR	320x200	4
MRESNOCOLOR	320x200	4
HRESBW	640x200	2

Pay attention that the resolutions 640x200 and 640x350 result in geometric distortions of the displayed image. Pay also close attention to the fact that no cursor appears on certain display modes!!! The subroutine sets the following global parameters related to the current graphics environment configuration:

```
NDMAX : number of pixels along the vertical direction
MDMAX : number of pixels along the horizontal direction
CNMAX : number of colors
NTMAX : number of text rows
MTMAX : number of text columns
```

RETURN VALUE: 0, -200, -201
SEE ALSO: reset_graphics

4.13 load_palette()

Subroutine to load palette
LIBRARY MODE:

```
int load_palette(p)
unsigned long far* p;
```

```
p: palette buffer
```

SHORT DESCRIPTION: This subroutine load a palette to the VGA card by using the _remappalette() subroutine of MICROSOFT C graphics library.
RETURN VALUE: 0, -2
SEE ALSO: load_palette(), default_palette()

4.14 overdisp()

Subroutine to overlay and display a binary image on a grayscale image
LIBRARY MODE:

```
int overdisp(a,b, cn, NS,MS, N1,M1,N2,M2)
image a,b;
int cn;
int NS,MS;
int N1,M1,N2,M2;
```

```
a: grayscale image buffer
b: binary image buffer
cn: number of gray shades
NS,MS: screen starting point
N1,M1: buffer display starting point
N2,M2: buffer display end point
```

INTERPRETER MODE:

```
overdisp a b cn NS MS [N1 M1 N2 M2]
a,b: integers
```

SHORT DESCRIPTION: This subroutine overlays a binary image stored on buffer b on a grayscale image stored on buffer a. Valid display cards are CGA, EGA, VGA. cn denotes the number of display intensities. The overlay intensity is cn (maximal intensity). The image on buffer b must be binary. The subroutine checks if the image on b is binary and if this is not the case, it returns an error value. NS, MS denote the screen display starting point. Both the intensities number and the display size depend on the display mode used. For more details on the allowable display dimensions see the description of the subroutine init_graphics or refer to the MICROSOFT C, RUN-TIME LIBRARY REFERENCE (subroutine _setvideomode).
RETURN VALUE: 0, -1, -21, -23, -60, -100
SEE ALSO: init_graphics, imdisp, bindisp, thres

4.15 reset_graphics()

Subroutine to reset graphics to hardware default video mode
LIBRARY MODE:

```
int reset_graphics()
```

INTERPRETER MODE:

```
reset_graphics
```

SHORT DESCRIPTION: This subroutine resets the graphics adapter to its hardware default mode. A new display made can be selected again by using the subroutine init_graphics.
RETURN VALUE: 0
SEE ALSO: init_graphics

4.16 rgbtovga()

Subroutine to transform an rgb image to an image that can be displayed on a VGA monitor with 256 colors
LIBRARY MODE:

```
int rgbtovga(r,g,b,vga,p,number,N1,M1,N2,M2)
image r;
image g;
image b;
image vga;
unsigned long far* p;
int number;
int N1,M1;
int N2,M2;

r,g,b: input image buffers
p: palette buffer
vga:   output image buffer
N1,M1: upper left corner coordinates
N2,M2: lower right corner coordinates
```

SHORT DESCRIPTION: This subroutine transforms an rgb image to an image that can be displayed on a VGA monitor with 256 colors. The palette to be used here must be created by using create_palette() or default_palette().
RETURN VALUE: 0, -1, -21, -90
SEE ALSO: create_palette(), default_palette(), vgadisp(), extvgadisp().

4.17 sigshow()

Subroutine to draw a signal array (float)
LIBRARY MODE:

```
int sigshow(s, M, x,y, w,h)
vector s;
```

```
int M;
int x,y;
int w,h;

s: signal buffer
M: signal size
x,y: starting display coordinates
w: x axis length
h: y axis length
```

INTERPRETER MODE:

```
sigshow s M x y w h
s: integer (0 or 1)
```

SHORT DESCRIPTION: This subroutine is used to display the signal buffer s on screen. The signal dimension M must be generally less than or equal to NSIG (maximum allowable signal dimension). The display width w and display height h depend on the display mode resolution that it is set by the subroutine init_graphics. The maximum value of x+w is MDMAX and the maximum value of y+h is NDMAX. The starting point (0,0) of the display is the coordinate (x,MDMAX-y) on the screen. For more details on the allowable display dimensions see the description of the subroutine init_graphics or refer to the MICROSOFT C, RUN-TIME LIBRARY REFERENCE (subroutine _setvideomode). Hard copies of a signal buffer can be obtained by using the subroutine dump_signal to dump its contents on a file. This file can be plotted afterwards by using suitable software. RETURN VALUE: 0, -2, -50 SEE ALSO: init_graphics, imdisp, dump_signal

---

4.18 two_d_coeff()

Subroutine to read 2-d linear filter coefficients
LIBRARY MODE:

```
int two_d_coeff(hcoe, NW,MW)
win_f hcoe;
int NW,MW;

hcoe: 2-d coefficient array
NW,MW: filter dimensions
```

INTERPRETER MODE:

```
two_d_coeff NW MW
```

SHORT DESCRIPTION: This subroutine is used in the interpreter mode to load coefficients on the 2-d coefficient buffer hcoe interactively. The buffer dimensions are NWxMW. This subroutine must be executed before subroutine conv.
RETURN VALUE: 0, -2, -24
SEE ALSO: conv

4.19 vgadisp()

Subroutine to display RGB images on VGA cards for video mode 320x200, 256 colors.
LIBRARY MODE:

```
int vgadisp(a,NS,MS, N1,M1,N2,M2)
image a;
int NS,MS;
int N1,M1,N2,M2;

a    : buffer
NS,MS: screen starting point (integers)
N1,M1: buffer display starting point
N2,M2: buffer display end point
```

SHORT DESCRIPTION: This subroutine display RGB images on VGA cards for video mode 320x200, 256 colors by using the routine movedata().
RETURN VALUE: -1, -21
SEE ALSO: imdisp, bindisp, extvgadisp

4.20 video_mode()

Subroutine to initialize extended VGA modes
LIBRARY MODE:

```
int video_mode(mode)
unsigned char mode;
mode: VGA mode number
```

SHORT DESCRIPTION: Subroutine that initializes extended VGA modes. It uses the interrupt 0x10.
RETURN VALUE: 0
SEE ALSO: init_graphics, getvideomode

4.21 view_pyramid()

Subroutine that displays the levels of a binary pyramid p
LIBRARY MODE:

```
int view_pyramid(p,color,N)
pyramid p;
int color;
int N;

p: input pyramid
color: gray level of 1s for image display
N: size of lowest level image
```

SHORT DESCRIPTION: Subroutine that displays the levels of a binary pyramid p. It uses subroutine bindisp().
RETURN VALUE: 0
SEE ALSO: bindisp

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

5

Memory allocation and image display routines for MS–Windows

5.1 alloc_logpalette()

Subroutine to allocate a LOGPALETTE structure (MS-Windows)
LIBRARY MODE:

```
LOGPALETTE *alloc_logpalette(hpal,PALETTESIZE)
HPALETTE *hpal;
unsigned long PALETTESIZE;

PALETTESIZE: number of colors that the display can handle
```

SHORT DESCRIPTION: Subroutine to allocate memory for the LOGPALETTE structure that holds palette information. On return, *hpal contains the handle to the allocated memory.

RETURN VALUE: It returns NULL if unsuccessful.

SEE ALSO: winbwimdisp(), wincolorimdisp()

5.2 computesignal

Function to compute the display values of a 1-D signal.
LIBRARY MODE:

```
int computesignal(psignal,start,end);
psignal struct SIGNAL far*;
start,end float;

psignal: Pointer to the signal structure.
start: The first point on x-axis.
end: The last point on x-axis.
```

SHORT DESCRIPTION: This function computes the display values of a 1-D signal. The user has to allocate space for the original values on x and y axis as well as for the displayed values. The fields 'valuesx' and 'valuesy' of the structure SIGNAL corresponds to the original signal. The field 'values' corresponds to the displayed signal. The constant SIGNAL_AXIS_X_SIZE represents the size of the displayed signal on x-axis. If start and end are zero then the minimum and maximum values of the original signal on x-axis are used. Finally the user must specify the number of original values on x axis. The fields 'valuesx', 'valuesy', 'start', 'end' are input arguments, and the field 'values' functions as output argument. The constants and the description of the structure SIGNAL can be found in eikona.h at the appendix.
RETURN VALUE: 0,-2,-90.
SEE ALSO: displaysignal.

5.3 create_disp_bmp()

Subroutine to initialize display bitmap (MS-Windows)
LIBRARY MODE:

```
int create_disp_bmp(hwnd,pDibInfo,p24DibInfo,PALETTESIZE)
HWND hwnd;
PBITMAPINFO * pDibInfo,* p24DibInfo;
unsigned long *PALETTESIZE;

hWnd: handle to display window
pDibInfo,p24DibInfo: double pointers to BITMAPINFO structures
*PALETTESIZE: number of colors that the display can handle
```

SHORT DESCRIPTION: Subroutine to initialize the BITMAPINFO structures pDibInfo, p24DibInfo used by winbwimdisp(), wincolorimdisp(), according to the current display mode and current image dimensions. It also returns the number of colors of the current display, on the variable PALETTESIZE.
RETURN VALUE: 0, -1
SEE ALSO: winbwimdisp(), wincolorimdisp()

5.4 displaysignal

Function to display a 1-D signal on the screen.
LIBRARY MODE:

```
int displaysignal(hDC,signal,mode);
signal struct SIGNAL;
int mode;

hDC: Handle of the device context of the window.
signal: Structure that contains the values of the signal to be displayed.
mode: 1 for normal display, 0 for inverse display.
```

SHORT DESCRIPTION: This function displays a 1-D signal on the screen. The displayed signal has already computed by the computesignal function.
RETURN VALUE: 0,-2.
SEE ALSO: computesignal.

5.5 win_build_image()

Subroutine to create image buffers (MS-Windows)
LIBRARY MODE:

WIN32s Version

```
unsigned char far* far* win_build_image(hmem,N,M)
HANDLE far * far *hmem;
```

```
int N,M;

N,M:  image dimensions
```

WIN32 Version

```
unsigned char * * win_build_image(N,M)
int N,M;

N,M : image dimensions
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and then select the "Buffers" item. A window appears, which holds information about the currently allocated buffers, for each buffer type. For image buffers the *Buffer type* combo-box should display Color Image or grayscale Image. Click the *New Buffer* button to allocate a new buffer or the *Delete Buffer* button to delete the buffer currently displayed in the thumbnail.
SHORT DESCRIPTION:
WIN32s Version: Function to create an image buffer of size NxM. It makes use of MS-Windows functions GlobalCompact, GlobalAlloc, GlobalLock to compact, allocate and lock the required memory (in pieces of 64512 bytes). hmem is a double pointer to handle which on return contains a pointer to an array of handles associated with the allocated memory pieces. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created.
NOTE: Although it is possible to use build_image(), with win_build_image() you can allocate larger blocks of memory. Thus, one should choose to allocate memory buffers with win_build_image(), when developing under the MS–Windows environment. The same is true for the win_build_matrix(), win_build_imatrix() routines.
WIN32 Version: Function to create an image buffer of size N×M. It makes use of the WIN32 function GlobalAlloc to allocate the image buffer and the associated table of pointers in one big chunk of memory thus avoiding memory fragmentation problems. The return value is a pointer to the allocated memory.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: win_build_matrix(), win_build_imatrix(), win_freeuc2(), init_eikona()

5.6 win_build_matrix()

Subroutine to create a matrix (float) buffer (MS-Windows)

LIBRARY MODE:

WIN32s Version

```
float far* far* win_build_matrix(hmem,N,M)
HANDLE far * far *hmem;
int N,M;

N,M:  float buffer dimensions
```

WIN32 Version

```
float * * win_build_matrix(N,M)
int N,M;

N,M : float buffer dimensions
```

MS–WINDOWS ENVIRONMENT: The procedure is the same as the one used to allocate buffers of type image (see build_image()), with the difference that the *Buffer type* combobox should display Float Buffer.
SHORT DESCRIPTION:
WIN32s Version: Function to create a float matrix buffer of size NxM. It makes use of MS-Windows functions GlobalCompact, GlobalAlloc, GlobalLock to compact, allocate and lock the required memory (in pieces of 64512 bytes). hmem is a double pointer to handle which on return contains a pointer to an array of handles associated with the allocated memory pieces. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created. The subroutine init_eikona() that determines the size of matrices must be used in advance.
WIN32 Version: Function to create a float matrix of dimensions N×M. It makes use of the WIN32 GlobalAlloc function to allocate both the buffer space and the associated table of pointers in one big chunk of memory thus avoiding memory fragmentation. Return value is a pointer to the buffer.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: win_build_image(), win_build_imatrix(), win_freef2(), init_eikona()

5.7 win_build_imatrix()

Subroutine to create an integer matrix buffer (MS-Windows)
LIBRARY MODE:

WIN32s Version

```
int far* far* win_build_imatrix(hmem,N,M)
HANDLE far * far *hmem;
int N,M;

N,M:  integer buffer dimensions
```

WIN32 Version

```
int * * win_build_imatrix(N,M)
int N,M;

N,M : integer buffer dimensions
```

SHORT DESCRIPTION:
WIN32s Version: Function to create an integer matrix buffer of size NxM. It makes use of MS-Windows functions GlobalCompact, GlobalAlloc, GlobalLock to compact, allocate and lock the required memory (in pieces of 64512 bytes). hmem is a double pointer to handle which on return contains a pointer to an array of handles associated with the allocated memory pieces. If N or M are less or equal to zero, a buffer of size NMAXxMMAX is created. The subroutine init_eikona() that determines the size of integer matrices must be used in advance.
WIN32 Version: Function to create an integer matrix buffer of size N×M. It makes use of the WIN32 GlobalAlloc function to allocate bothe the buffer space and the associated table of pointers in one big chunk thus minimizing memory fragmentation.
RETURN VALUE: It returns NULL if unsuccessful.
SEE ALSO: win_build_image(), win_build_matrix(), win_freei2(), init_eikona()

5.8 winbwimdisp()

Subroutine to display a black & white or binary image (MS-Windows)
LIBRARY MODE:

```
int winbwimdisp(hWnd,pPal,pDibInfo,PALETTESIZE,im, NS,MS, N1,M1,N2,M2,
                zoomfactor)
HWND hWnd;
LOGPALETTE * pPal;
PBITMAPINFO  pDibInfo;
unsigned long PALETTESIZE;
image im;
```

```
int NS,MS;
int N1,M1,N2,M2;
int zoomfactor;

hWnd: handle to display window
pPal: pointer to the logical color palette structure
      associated with the image
pDibInfo: pointer to display BITMAPINFO structure
PALETTESIZE: number of colors that the display can handle
im: input image buffer
NS,MS: display window starting point
N1,M1: buffer display starting point
N2,M2: buffer display end point
zoomfactor: image scaling factor
```

SHORT DESCRIPTION: Subroutine to display a black & white or binary image on a window (specified by hWnd), under MS-Windows. The subroutines create_disp_bmp(), alloc_logpalette() (with this execution order) must be used before displaying an image to initialize pDibInfo, PALETTESIZE, pPal. On return, pPal contains the grayscale palette. The scale of the displayed image is controlled by the parameter zoomfactor i.e. the displayed image size (both in X and Y coordinates) is the input image size divided by zoomfactor.
RETURN VALUE: 0, -1, -21
SEE ALSO: create_disp_bmp(), alloc_logpalette, wincolorimdisp()

5.9 win_freeuc2()

Subroutine to free an unsigned char buffer (MS-Windows)
LIBRARY MODE:

WIN32s Version

```
void win_freeuc2(hmem,N,M)
HANDLE far * far *hmem;
int N,M;

N,M:  unsigned char buffer dimensions
```

WIN32 Version

```
void win_freeuc2(im)
unsigned char **im;

im : Pointer to an image
```

SHORT DESCRIPTION:
WIN32s Version: Subroutine to free a 2-d array of type unsigned char (image). This array must have been created by the WIN32s win_build_image(). hmem is the pointer to the array of handles returned by the call to win_build_image() that created the image.
WIN32 Version: Function to free a 2-d array of type unsigned char (image) pointed to by 'im'. 'im' should be a pointer returned by the WIN32 version of "win_build_image".
SEE ALSO: win_freef2(), win_freei2(), win_build_image()

5.10 win_freef2()

Subroutine to free a float buffer (MS-Windows)
LIBRARY MODE:

WIN32s Version

```
void win_freef2(hmem,N,M)
HANDLE far * far *hmem;
int N,M;

N,M:  float buffer dimensions
```

WIN32 Version

```
void win_freef2(mat)
float **mat;

mat : Pointer to a float matrix
```

SHORT DESCRIPTION:
WIN32s Version: Subroutine to free a 2-d array of type float (matrix). This array must have been created by the WIN32s win_build_matrix(). hmem is the pointer to the array of handles returned by the call to win_build_matrix() that created the matrix.

WIN32 Version: Function to free a 2-d array of type float (matrix) pointed to by 'mat'. 'mat' should be a pointer returned by the WIN32 version of "win_build_matrix".
SEE ALSO: win_freeuc2(), win_freei2(), win_build_matrix()

5.11 win_freei2()

Subroutine to free an integer buffer (MS-Windows)
LIBRARY MODE:

WIN32s Version

```
void win_freei2(hmem,N,M)
HANDLE far * far *hmem;
int N,M;

N,M:  integer buffer dimensions
```

WIN32 Version

```
void win_freei2(imat)
int **imat;

imat : Pointer to an integer matrix
```

SHORT DESCRIPTION:
WIN32s Version: Subroutine to free a 2-d array of type integer. This array must have been created by win_build_imatrix(). hmem is the pointer to the array of handles returned by the call to win_build_imatrix() that created the integer buffer.
WIN32 Version: Function to free a 2-d array of type integer (imatrix) pointed to by 'imat'. 'imat' should be a pointer returned by the WIN32 version of "win_build_imatrix".
SEE ALSO: win_freef2(), win_freeuc2(), win_build_imatrix()

5.12 wincolorimdisp()

Subroutine to display a color image (MS-Windows)
LIBRARY MODE:

```
int wincolorimdisp(hWnd,pPal,pDibInfo,p24DibInfo,PALETTESIZE,
                   display,r,g,b, NS,MS, N1,M1,N2,M2,zoomfactor)
HWND hWnd;
LOGPALETTE * pPal;
PBITMAPINFO  pDibInfo,p24DibInfo;
unsigned long PALETTESIZE;
int display;
unsigned char far* far* r;
unsigned char far* far* g;
unsigned char far* far* b;
int NS,MS;
int N1,M1,N2,M2;
int zoomfact;

hWnd: handle to display window
pPal: pointer to the logical color palette structure
      associated with the image
pDibInfo,p24DibInfo: pointers to  BITMAPINFO structures
PALETTESIZE: number of colors that the display can handle
display: display mode (0: preview, 1: full display)
r: input red image buffer
g: input green image buffer
b: input blue image buffer
NS,MS: display window starting point
N1,M1: buffer display starting point
N2,M2: buffer display end point
zoomfactor: image scaling factor
```

SHORT DESCRIPTION: Subroutine to display a color image on a window (specified by hWnd) under MS-Windows. The subroutines create_disp_bmp(), alloc_logpalette() (with this execution order) must be used before displaying an image to initialize pDibInfo, p24DibInfo, PALETTESIZE, pPal. If the current number of display colors (specified by PALETTESIZE) are 256 the subroutine evaluates and loads a palette suitable for the input image. This palette is stored in pPal. Two different palette types can be selected in this case : a standard preview palette (display=0), or an image-specific palette which takes more time to be calculated (display=1). If the current number of display colors is different than 256 the subroutine uses the system palette. The scale of the displayed image is controlled by the parameter zoomfactor i.e. the displayed image size (both in X and Y coordinates) is the input image size divided by zoomfactor.

RETURN VALUE: 0, -1, -21, -90
SEE ALSO: create_disp_bmp(), alloc_logpalette, winbwimdisp()

6

Image display routines for X–Windows

6.1 disp_grayscale()

Subroutine to display a black & white or binary image (X-Windows)
LIBRARY MODE:

```
Widget disp_grayscale(a, display, parent, width, height)
image a;
Display * display;
Widget parent;
int width, int height;

a: the image to be displayed
display: a pointer to the X-Windows Display structure, which contains
information about the type of display
parent: the parent window of the display window
width, height: image dimensions
```

SHORT DESCRIPTION: This subroutine first creates a private grayscale colormap on the display server using its default visual and screen information. Then it creates the display window, adds scrollbars if needed, and displays the image in the window. The routine checks if the image to be displayed is binary or grayscale, and creates a private colormap suitable for each case.
SEE ALSO: display_color()

6.2 disp_color()

Subroutine to display a color image (X-Windows)
LIBRARY MODE:

```
Widget disp_color(a, b, c, display, parent, width, height)
image a, b, c;
Display * display;
Widget parent;
int width, int height;

a, b, c: the R, G, B components of the color image to be displayed
display: a pointer to the X-Windows Display structure, which contains
information about the type of display
parent: the parent window of the display window
width, height: image dimensions
```

SHORT DESCRIPTION: This subroutine first checks the display capabilities of the display server.
If the display server supports 24-bit color display (16 million colors), the routine uses the default TrueColor visual of the server. Then it creates the display window, adds scrollbars if needed, and displays the image in the window.
SEE ALSO: display_grayscale()

7

Input-Output routines

7.1 Acquire to Memory

Scan an image using a scanner compatible with the TWAIN interface.
MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Acquire" and then "Acquire to Memory". The Scanner dialog box provided by the scanner's TWAIN interface will appear on the screen. Select the scanning session parameters (resolution, area etc.) and click "OK". A new buffer will be created for the scanned image.
SHORT DESCRIPTION: This function allows the user to acquire (scan) an image from a scanner. The scanner should be compatible with the TWAIN interface and the corresponding TWAIN software (provided by the scanner manufacturer) should be properly installed.
SEE ALSO Select Source

7.2 dump()

Subroutine to dump the buffer contents on a file in decimal format.
LIBRARY MODE:

```
int dump(a, dumpfile, N1,M1,N2,M2)
```

```
image a;
char dumpfile[];
int N1,M1,N2,M2;

a: image buffer
dumpfile: OS file name (string)
N1,M1 buffer starting point (integers)
N2,M2 buffer end point
```

INTERPRETER MODE:

```
dump a dumpfile [N1 M1 N2 M2]
a: integer
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump". Give the filename under which the image will be dumped and select the image you want to dump. After clicking "OK", a message appears stating the dimensions of the image that was dumped.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump", then select "Dump Image". Enter the number of the image you want to dump, and the filename under which it will be saved.
SHORT DESCRIPTION: This subroutine dumps the contents of an image buffer in a text file in decimal format (integers). The dumpfile contents can be viewed and/or printed by an OS command. In the interpreter environment, they can be viewed by executing 'os type dumpfile'.
RETURN VALUE: 0, -1, -21, -300
SEE ALSO: dump_signal, printim, os, system, pixr

7.3 dumpmatrix()

Subroutine to dump the contents of a matrix on a file in decimal format. It uses the region of interest.
LIBRARY MODE:

```
int dumpmatrix(a, dumpfile, N1, M1, N2, M2)
matrix a;
char dumpfile[];
int N1,M1,N2,M2;

a: matrix buffer
```

```
dumpfile: OS file name (string)
N1,M1: upper left corner coordinates
N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump Matrix". Give the filename under which the matrix will be saved and select the matrix you want to save. After clicking "OK", a message appears stating the dimensions of the matrix that was saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump", then select "Dump Matrix". Enter the number of the matrix you want to dump, and the filename under which it will be saved.
SHORT DESCRIPTION: This subroutine dumps the contents of a matrix buffer in a text file in decimal format. The dumpfile contents can be viewed and/or printed by an OS command.
RETURN VALUE: 0, -1, -21, -300
SEE ALSO: dump, dump_matrix_hist, dump_signal

7.4 dump_signal()

Subroutine to dump the signal buffer contents on a file in decimal format
LIBRARY MODE:

```
int dump_signal(h, dumpfile)
vector h;
char dumpfile[];

h: signal buffer
dumpfile: OS file name (string)
```

INTERPRETER MODE:

```
dump_signal h dumpfile
h: integer (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump Vector". Give the filename under which the vector will be dumped. Then, enter the number of the vector you want to dump.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump", then select "Dump Vector". Enter the number of the vector you want to dump, and the filename under which it will be saved.

SHORT DESCRIPTION: This subroutine dumps the contents of an signal buffer in a text file in decimal format in the form (i, h[i]). The dumpfile contents can be viewed and/or printed by an OS command. They can also be plotted by using a software package for function plotting (e.g. TOPO). In the interpreter environment, they can viewed by executing 'os type dumpfile'.
RETURN VALUE: 0, -2, -300
SEE ALSO: dump, printim, os, system, hist, sigshow

7.5 dump_matrix_hist()

Subroutine to dump the histogram of a matrix in a file in decimal mode.
LIBRARY MODE:

```
int dump_matrix_hist(vect[], OpenName, NSIG)
vector vect[];
char* OpenName;
int NSIG;

vect: array of histogram values
OpenName: File name
NSIG: size of vector
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump Matrix Histogram". Give the filename under which it will be saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select submenu "Dump", then select "Dump Matrix Histogram". Give the filename under which it will be saved.
SHORT DESCRIPTION: This subroutine dumps the contents of the histogram of a matrix in a text file in decimal format in the form (x[i], h[i]). The dumpfile contents can be viewed and/or printed by an OS command. They can also be plotted by using a software package for function plotting (e.g. TOPO). The x[i] values are calculated by taking the minimum and maximum values of the matrix and after that we define NSIG locations for the histogram calculation.
RETURN VALUE: 0, -2, -90, -600, -602
SEE ALSO: dump, dumpmatrix, hist, matrixhist

7.6 fprint_chain()

Subroutine to store a chain to file
LIBRARY MODE:

```
int fprint_chain(fn, i, j, chain)
char *fn;
int i, j;
CHAIN chain;

fn: filename
i, j: image starting points
chain: chain
```

SHORT DESCRIPTION: This subroutine stores a chain to a disk file. The starting image coordinates (i,j) of the chain are stored first. The elements of the chain are stored sequentially. The end-of-chain marker is the number -111.
RETURN VALUE: 0, -300
SEE ALSO: fprint_list

7.7 fprint_list()

Subroutine to store a list to file
LIBRARY MODE:

```
int fprint_list(fn, i, j, list)
char *fn;
int i, j;
LIST list;

fn: filename
i, j: image starting points
list: list of pixels
```

SHORT DESCRIPTION: This subroutine stores a list to a disk file. The starting image coordinates (i,j) of the chain are stored first. The elements of the chain are stored sequentially. The end-of-list marker is the number -111.
RETURN VALUE: 0, -300
SEE ALSO: fprint_chain

7.8 fromdisk()

Subroutine to load an image from a disk file.
LIBRARY MODE:

```
int fromdisk(imfile, a, ND1,MD1, N,M)
char imfile[];
image a;
int ND1,MD1;
int N,M;

imfile: OS file name (string)
a: image array of dimensions N,M
ND1,MD1: starting point on the buffer a
N,M: image dimensions  in the file imfile
```

INTERPRETER MODE:

```
fromdisk imfile a [ND1 MD1 N M]
a: integer
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open". Use the *Open Image* common dialog box that appears to select the file to open. Any file with an extension other than JPG, GIF, TIF, TGA, BMP will be opened as a binary (raw) image. In the *Open Raw* dialog box that appears enter the dimensions of the image or use the *Guess* and *Swap* buttons to let EIKONA guess them. Press *OK* to load the image.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Import Raw". An image file can be loaded by either entering its complete name in the "Open File Name" field, including path, or by directly manipulating the elements of the "Files" list. Alternatively, the "File Name" field can also be used as a file mask, to filter in the files that show up in the "Files" list. Fill in also the destination buffer number and image dimensions fields.
SHORT DESCRIPTION: This subroutine loads an image from a disk file and stores it on an image buffer.
RETURN VALUE: 0, -1, -10, -21, -22, -300
SEE ALSO: todisk, dump

7.9 frombmp()

Subroutine to load a Windows Bitmap (BMP) image from a disk file.

LIBRARY MODE:

```
int frombmp(fname,r,g,b)
char *fname;
image r,g,b;

fname:          input image filename (string)
r,g,b:          image buffers
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open". From the *Open Image* common dialog box that appears select a BMP file and click *Open*. A new buffer will be created for the image.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open", then choose the BMP button from the selection list. Enter the filename of the BMP file to be loaded in the appropriate field. In the "Image Number" field, enter the number of the color buffer in which the file will be loaded into.
SHORT DESCRIPTION: This subroutine loads a BMP image from a disk file and stores it on an image buffer. BW images are loaded as color images (i.e. in three consecutive BW buffers) having identical color channels. Compressed BMP files are not currently supported.
RETURN VALUE: 0, -1, -31, -300
SEE ALSO: fromdisk, fromtiff, fromtga, fromgif, fromjpeg

7.10 fromjpeg()

Subroutine to load a jpeg image from a disk file.
LIBRARY MODE:

```
int fromjpeg (filename, imbw,  im1, im2,  im3, DimN, DimM)
char * filename;
image imbw;
image im1, image im2, image im3;
int *DimN, int *DimM;

filename:           input image filename (string)
imbw:               image buffer (BW JPEG image)
im1,im2,im3:        image buffers (color JPEG image)
*DimN, *DimM:       input image dimensions.
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open".From the *Open Image* common dialog box that appears select a JPEG file and click *Open*. A new buffer will be created for the image.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open", then choose the "Jpeg" button from the selection list. Enter the filename of the Jpeg file to be loaded in the appropriate field. In the "Image Number" field, enter the number of the color buffer in which the file will be loaded into. The chosen number will correspond to a color or a BW buffer depending on whether the JPEG image is a color or a BW one.
SHORT DESCRIPTION: This subroutine loads a JPEG image from a disk file and stores it on an image buffer. If the JPEG file contains a BW image, this is stored on image buffer imbw. If the JPEG file contains a color image it is stored on image buffers im1,im2,im3. On return, DimN, DimM contain the JPEG image dimensions.
RETURN VALUE: 0, -1, -29, -300
SEE ALSO: ReadJPEGHead, fromdisk, fromtiff, fromtga, frombmp

7.11 fromtga()

Subroutine to load an image from a TGA file.
LIBRARY MODE:

```
int fromtga(fname,imbw,r,g,b)
char *fname;
image imbw,r,g,b;

fname:          input image filename (string)
imbw:           image buffer (BW TGA image)
r,g,b:          image buffers  (color TGA image).
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open". From the *Open Image* common dialog box that appears select a TGA file and click *Open*. A new buffer will be created for the image.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open", then choose the "Tga" button from the selection list. Enter the filename of the Tga file to be loaded in the appropriate field. In the "Image Number" field, enter the number of the buffer in which the file will be loaded into. The chosen number will correspond to a color or a BW buffer depending on whether the TGA image is a color or a BW one.

SHORT DESCRIPTION: This subroutine loads an image from a TGA disk file. The subroutine can load color images having 16, 24, 32 bits per pixel. If the TGA file contains a BW image i.e., if it is of type 3 (TGA_Mono) or 11 (TGA_RLEMono), the image is stored on image buffer imbw. If the TGA file contains a color image this is stored on image buffers r,g,b.
RETURN VALUE: 0, -1, -30, -300
SEE ALSO: ReadTGAHead, fromdisk, fromtiff, frombmp, fromgif, fromjpeg

---

7.12 fromtiff()

Subroutine to read the image related to the first IFD of a TIFF image file
LIBRARY MODE:

```
int fromtiff(name,r,g,b)
char *name;
image r,g,b;

name  : image file
r,g,b : output image buffers
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open". From the *Open Image* common dialog box that appears select a TIFF file and click *Open*. A new buffer will be created for the image.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open", then select "TIFF" button from the selection list. Enter the required file name and the number of the image buffer that will hold the image data and press "OK". In the "Image Number" field, enter the number of the buffer in which the file will be loaded into. The chosen number will correspond to a color or a BW buffer depending on whether the Tiff image is a color or a BW one. If the Tiff image size exceeds the current buffer size the user is prompted to allocate bigger image buffers. If the user press the O.K button then the system will allocate the appropriate buffers but if he press the Cancel button the process will be stoped.
SHORT DESCRIPTION: This subroutine loads an image from a TIFF disk file and stores it on three image buffers. The size of the allocated buffers can be determined using gettiffinfo. RETURN VALUES: 0, -1, -300, [-1000] SEE ALSO: fromdisk, fromtga, gettiffinfo

7.13 gettiffinfo()

Subroutine to get size and color information about a TIFF image file.
LIBRARY MODE:

```
int gettiffinfo(name,width,height,isrgb)
char *name;
unsigned long *width,*height;
unsigned char *isrgb;

name            : image file
*width, *height : width and height of the TIFF image in pixels
*isrgb          : flag indicating if image is color or grayscale
```

SHORT DESCRIPTION: This subroutine reads appropriate tags of a TIFF image into width, height, and isrgb parameters.
RETURN VALUES: 0, -1, -300, [-1000]
SEE ALSO: fromtiff

7.14 loadmatrix()

Subroutine to load a matrix from a file.
LIBRARY MODE:

```
int loadmatrix(matfile, mat, ND1,MD1, N,M)
char matfile[];
matrix mat;
int ND1,MD1;
int N,M;

matfile : DOS file name (string)
mat : matrix array
ND1,MD1 : destination coordinates
N,M : size of the matrix stored in the file matfile
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open Matrix". Select the filename and press *Open*. In the window that appears fill-in the information of the target matrix buffer.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Open", then select button "Float matrix". Enter the filename and press "OK". In the window that appears fill-in the information (width, height) of the target matrix buffer and press "OK".
SHORT DESCRIPTION: This subroutine loads a matrix from a disk file and stores it on an image buffer. The matfile is assumed to be created by the savematrix command or any other user procedure. It must be a binary file. The matrix contents are read in a row wise fashion. The fread C command is used.
RETURN VALUE: 0, -1, -10, -21, -22, -300
SEE ALSO: savematrix

7.15 printim()

Subroutine to print a binary image on a dot matrix printer.
LIBRARY MODE:

```
int printim(a, prnfile, N1,M1,N2,M2)
image a;
char prnfile[];
int N1,M1,N2,M2;

a: image buffer
prnfile: OS file name (string)
N1,M1 start coordinates
N2,M2 end   coordinates
```

INTERPRETER MODE:

```
printim a prnfile [N1 M1 N2 M2]
a: integer
```

SHORT DESCRIPTION: This subroutine scans the contents of the image buffer a and stores them on a disk file ready to be printed. The image on buffer a must be binary, otherwise the subroutine returns an error value. If a grayscale image must be printed, it must be halftoned in advance. The prnfile can be printed by an OS command. In interpreter environment, it can be printed by executing 'os print prnfile'. Currently only STAR printers are supported.

RETURN VALUE: 0, -1, -21, -100, -300
SEE ALSO: dump, halftone, thres, imdisp, bindisp

7.16 print_hp()

Subroutine to print a binary image on a HP laser printer
LIBRARY MODE:

```
int print_hp(a, prnfile,DPI,ND, MD, N1,M1,N2,M2)
image a;
char prnfile[];
int DPI,ND,MD,N1,M1,N2,M2;

a: image buffer
prnfile: OS file name (string)
DPI: dots per inch
ND,MD: destination coordinates
N1,M1: buffer display starting point (integers)
N2,M2: buffer display end point
```

X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Print". Complete the requested fields and then press "OK".
SHORT DESCRIPTION: Subroutine to print a binary image on a HP compatible laser printer. This subroutine scans the contents of the image buffer a and stores them on a disk file ready to be printed. The image on buffer a must be binary, otherwise the subroutine returns an error value. If a grayscale image must be printed, it must be halftoned in advance. The prnfile can be printed by an OS command. In interpreter environment, it can be printed by executing 'os print prnfile'.
RETURN VALUE: 0, -1, -21, -100, -300
SEE ALSO: dump, halftone, printim, thres, imdisp, bindisp

7.17 ReadBMPHead()

Subroutine to read the header of a BMP image from disk.

LIBRARY MODE:

```
int ReadBMPHead(filename,bmp_coltype_size)
char *filename;
FORMAT_COLORTYPE_SIZE *bmp_coltype_size;

filename: input image filename
bmp_coltype_size: Structure with the size of the BMP image.
```

SHORT DESCRIPTION: This subroutine reads the header of a BMP image from a disk file and fills in the FORMAT_COLORTYPE_SIZE structure with the dimensions of the BMP image. If an error occurs or we try to read a file format different from BMP, this function returns error code -31.
RETURN VALUE: 0, -31, -300
SEE ALSO: ReadTGAHead, ReadJPEGHead, ReadGIFHead, frobmp.

7.18 ReadGIFHead()

Subroutine to read the header of a GIF image from disk.
LIBRARY MODE:

```
int ReadGIFHead(filename,gif_coltype_size)
char *filename;
FORMAT_COLORTYPE_SIZE *gif_coltype_size;

filename: input image filename
gif_coltype_size: Structure with the size of the GIF image.
```

SHORT DESCRIPTION: This subroutine reads the header of a GIF image from a disk file and fills in the FORMAT_COLORTYPE_SIZE structure with the dimensions of the GIF image. If an error occurs or we try to read a file format different from GIF, this function returns error code -32.
RETURN VALUE: 0, -32, -300
SEE ALSO: ReadTGAHead, ReadJPEGHead, ReadBMPHead, frogif.

7.19 ReadJPEGHead()

Subroutine to read the header of a JPEG image from disk.
LIBRARY MODE:

```
int ReadJPEGHead(filename,jpeg_coltype_size)
char *filename;
FORMAT_COLORTYPE_SIZE *jpeg_coltype_size;

filename: input image filename
jpeg_coltype_size: Structure with the size and color type of the JPEG image
```

SHORT DESCRIPTION: This subroutine reads the header of a JPEG image from a disk file and fills in the FORMAT_COLORTYPE_SIZE structure with the dimensions and the color type of the JPEG image. If an error occurs or we try to read a file format different from JPEG this function will return error code -29.
RETURN VALUE: 0, -29, -300
SEE ALSO: ReadTGAHead, ReadGIFHead, ReadBMPHead, fromjpeg.

7.20 ReadTGAHead()

Subroutine to read the header of a TGA image from disk.
LIBRARY MODE:

```
int ReadTGAHead(fname,tga_coltype_size)
char *fname;
FORMAT_COLORTYPE_SIZE *tga_coltype_size;

fname: input image filename
tga_coltype_size: Structure with the size and color type of the TGA image
```

SHORT DESCRIPTION: This subroutine reads the header of a TGA image from a disk file and fills in the FORMAT_COLORTYPE_SIZE structure with the dimensions and the color type of the TGA image. If an error occurs or we try to read a file format different from TGA this function will return error code -30.
RETURN VALUE: 0, -30, -300
SEE ALSO: ReadJPEGHead, ReadBMPHead, ReadGIFHead, fromtga.

7.21 savematrix()

Subroutine to store a matrix in a file
LIBRARY MODE:

```
int savematrix(mat, matfile, N1,M1,N2,M2)
matrix mat;
char matfile[];
int N1,M1,N2,M2;

mat : matrix array
matfile : DOS file name (string)
N1,M1 : upper left  corner of source matrix
N2,M2 : lower right corner of source matrix
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "Matrix file" in the *Save as Type* combo-box. Select the filename in which the matrix will be saved, then fill-in the required fields and press "OK". After clicking "OK", a message appears stating the dimensions of the matrix that was saved.
SHORT DESCRIPTION: This subroutine stores the contents of the matrix mat on the disk file matfile in binary format (4 bytes/element). The matrix contents are written in a row wise fashion. The fwrite C command is used.
RETURN VALUE: 0, -1, -10, -21, -300
SEE ALSO: todisk, dump, dump_signal, printim

7.22 Select Source

Select a TWAIN compatible source (scanner).
MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Acquire" and then "Select Source". The "Select Source" dialog box provided by the scanner's TWAIN interface will appear on the screen. Select the TWAIN source (scanner) to be used in your acquisition (scanning) sessions and click "OK".
SHORT DESCRIPTION: This function allows the user to select a TWAIN source, i.e. a TWAIN compatible scanner when more than one such devices are connected to the computer. If only one scanner is connected to the computer this is automatically selected as the default scanner.
NOTE: Imaging devices other than scanners (e.g. digital cameras) are expected to comply with the TWAIN interface standard in the near future.

SEE ALSO Acquire to Memory, Acquire to File

7.23 todisk()

Subroutine to store an image buffer in a file
LIBRARY MODE:

```
int todisk(a, imfile, N1,M1,N2,M2)
image a;
char imfile[];
int N1,M1,N2,M2;

a: buffer
imfile: OS file name (string)
N1,M1: start coordinates
N2,M2: end coordinates
```

INTERPRETER MODE:

```
todisk a imfile [N1 M1 N2 M2]
a:integer
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "Bin file" in the *SAve as type* combobox and specify the filename, then the buffer to be saved. After clicking "Save", a message appears stating the dimensions of the image that was saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then select the "Raw" button from the selection list. An image can be saved to disk by either entering its complete name in the "File Name" field, including path, or by directly manipulating the elements of the "Files" list. Alternatively, the "File Name" field can also be used as a file mask, to filter in the files that show up in the "Files" list. Fill in also the number of the image buffer to be saved.
SHORT DESCRIPTION: This subroutine stores an image on a disk file in binary format (1 byte/pixel).
RETURN VALUE: 0, -1, -10, -21, -300
SEE ALSO: fromdisk, dump, dump_signal, printim

7.24 totga()

Subroutine to store an image on a TGA image file
LIBRARY MODE:

```
int totga(r,g,b,imfile, N1,M1, N2,M2,depth)
char imfile[];
image r;
image g;
image b;
int N1,M1;
int N2,M2;
unsigned char depth;

r,g,b: input image buffers
imfile: output image file name
N1, M1: upper left corner coordinates
N2, M2: lower right corner coordinates
depth: number of bits per pixel stored at the output image file
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "TGA file" from the *Save as type* combobox and specify the filename. Then, select the type of the Targa file that is going to be saved and specify the color type of the image buffer. Complete the required fields and press "OK". After clicking "OK", a message appears stating the dimensions of the image that was saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then choose the "Tga" button from the selection list. Enter the filename of the Tga file to be loaded in the appropriate field. In the "Image Number" field, enter the number of the color buffer in which the file will be loaded into.
SHORT DESCRIPTION: This subroutine stores an image to a TGA disk file from three image buffers. The subroutine can store color images having 16, 24, 32 bits per pixel. It can also store BW images having 8 bits per pixel.
RETURN VALUE: 0, -1, -10, -21, -30, -90, -300
SEE ALSO: todisk, dump

7.25 topostscript()

Subroutine to store an image in the Adobe Postscript format
LIBRARY MODE:

```
int topostscript(a,b,c,filename,paper_size,color_type,N1,M1,N2,M2)
image q;
image b;
image c;
char filename[];
int paper_size;
int color_type;
int N1,M1;
int N2,M2;

a,b,c: input image buffers
filename: output image file name
paper_size: paper size
color_type: color flag indicating output color type
N1, M1: upper left corner coordinates
N2, M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "Postscript" from the *Save as type* combobox. Select the output file name then, the image buffer to be saved. Then select the color type of the input image depending on the value of the color check box, the color type of the postscript file and the paper size, and press "OK". After clicking "OK", a message appears stating the dimensions of the image that was saved.

X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then select "Postscript" from the selection list. Fill in the output file name and the image buffer to be saved, select the postscript color type and the paper size, and press "OK".

SHORT DESCRIPTION: This subroutine stores an image to a postscript file. A color input image can be saved either as a color or as a grayscale postscript file. In the second case the luminance of the color image is computed. The postscript file can then be sent to any postscript printer, or viewed with any postscript viewing program. The paper size depends on the printing media, and can be one of the following: A4, A3, US legal or 35mm slide. The printed image is automatically centered in the selected paper. The DPI resolution of the printed image depends on its dimensions and on the paper size selected. A grayscale image does not have to be dithered, as dithering is done by the postscript printer.

RETURN VALUE: 0, -1, -21, -90, -300

SEE ALSO: todisk

7.26 thumbnail()

Subroutine to create a thumbnail image.
LIBRARY MODE:

```
int thumbnail(im1,im2,im3,im4,im5,im6,col_bw,thumbx,thumby,
              roitop,roileft,roibottom,roiright)
image im1,im2,im3,im4,im5,im6;
int col_bw;
int thumbx,thumby;
int roitop,roileft,roibottom,roiright;

im1,im2,im3:         input buffers
im4,im5,im6:         intermediate storage buffers
col_bw:              color/BW indicator (1=color, 0=BW)
thumbx,thumby:       thumbnail size
roitop,roileft       upper left corner coordinates
roibottom,roiright   lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then "Thumbnail" in the *Save as type* combobox.. Select the output filename. Fill in the thumbnail image dimensions or, alternatively, check the "Use Default Size" box if you want to save a thumbnail with the default thumbnail dimensions stored in the EIKONA.PAR file. Give the buffer number that is to be stored as a thumbnail and specify its color type. The thumbnail operation destroys the last Color/BW image buffer. The thumbnail image is saved in JPEG format with a quality factor equal to 75.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then select "Thumbnail" from the selection list, and either "BW" or "Color". Fill in the thumbnail image dimensions or, alternatively, check the "Use Default Size" box if you want to save a thumbnail with the default thumbnail dimensions stored in the EIKONA.PAR file. Provide also the output image filename as well as the buffer number that is to be stored as a thumbnail.
SHORT DESCRIPTION: Subroutine to create a thumbnail and save it to disk in JPEG format. The input image can be either BW or color, depending on the value of the integer variable col_bw. If the input image is chosen to be a BW one it should reside in image buffer im1. This operation requires one BW (im4) or Color (im4, im5, im6) buffer for storage of intermediate results. The input image is scaled by an integer scaling factor so that its dimensions become smaller or equal to the specified thumbnail dimensions.
RETURN VALUE: 0, -1, -21, -90, return values of decim
SEE ALSO: decim

7.27 tojpeg()

Subroutine to store an image on a JPEG image file
LIBRARY MODE:

```
int tojpeg (char * filename, image im1, image im2, image im3,
            int quality,int col_bw ,int subsampling, int roitop, int roileft,
            int roibottom,int roiright )
char * filename;
image im1, image im2, image im3;
int quality;
int col_bw;
int roitop, roileft, roibottom, roiright;

filename:              output file name (string)
im1,im2,im3:           input image buffers
quality:               JPEG quality factor. Range: 0..100
col_bw:                color/BW indicator (1=color output,0=BW output)
subsampling            subsampling indicator  (0: no subsampling,
      1: 2 to 1 horizontal and vertical subsampling in Cb, Cr
      color components (YCbCr colorspace)
roitop, roileft:       upper left corner coordinates
roibottom, roiright:   lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "Jpeg File" in the *Save as type* combobox. Select the output filename. Then, specify the color type of the buffer to be saved. Fill in the quality factor and specify whether or not chromaticity subsampling is desired (in Color Jpeg only) and press "OK". After clicking "OK", a message appears stating the dimensions of the image that was saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then select "Jpeg" from the selection list, and either "BW" or "Color". Fill in the output file name, the image buffer to be saved, the quality factor and specify whether or not chromaticity subsampling is desired (in Color Jpeg only) and press "OK".
SHORT DESCRIPTION: This subroutine stores an image to a JPEG disk file. The image to be saved can be a BW or a color one, depending on the value of the integer variable col_bw. If the output image is chosen to be a BW one it should reside in image buffer im1. The quality factor controls the quality of the JPEG compressed image and, consequently, the compression ratio. It can obtain integer values within the range 0 (very poor quality, big compression) - 100 (perfect quality, very little compression).
RETURN VALUE: 0, -1, -29, -90, -300
SEE ALSO: todisk, totga

7.28 totiff()

Subroutine to store three (or one) image buffers as a TIFF file.
LIBRARY MODE:

```
int totiff(r,g,b,name,rowsperstrip,compression,config,photometric,N1,M1,N2,M2)
image r,g,b;
char *name;
unsigned long rowsperstrip;
unsigned short compression,config,photometric;
int N1, M1, N2, M2;

r,g,b        : input image buffers
name         : image file
rowsperstrip : number of rows in every strip that is stored
compression  : type of compression used:     1 -> None
                                             5 -> LZW
                                         32773 -> PackBits
config       : type of planar configuration of samples: 1 -> planar
                                                        2 -> separate
photometric  : type of photometric interpretation: 0 -> MinIsWhite
                                                   1 -> MinIsBlack
                                                   2 -> RGB
N1,M1        : upper left corner coordinates
N2,M2        : lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save". Select "TIFF file" in the *Save as type* combobox. Select the output filename and the buffer, specify the color type of the buffer to be saved, select the "Compression" and "Photometric" radio buttons and press "OK". After clicking "OK", a message appears stating the dimensions of the image that was saved.
X–WINDOWS ENVIRONMENT: Open the "File" menu and select "Save", then select "TIFF" button from the selection list. Complete the "Buffer Number", "Output File Name" fields, and select the "Compression" and "Photometric" radio buttons. Then press "OK".
SHORT DESCRIPTION: This subroutine stores three (in case of a color image) or one (in case of a grayscale image) image buffers into a TIFF disk file.
RETURN VALUES: 0, -1, -300, [-1000,-1117] SEE ALSO: todisk, totga, dump

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

8

Basic image processing routines

8.1 add()

Subroutine to add two images
LIBRARY MODE:

```
int add(a,b,c, N1,M1,N2,M2)
image a,b,c;
int N1,M1, N2, M2;

a,b: source image buffers
c: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
add a b c [N1 M1 N2 M2]
a,b,c: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Add". Fill-in the required fields, then click "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Add". Fill-in the required fields, then click "OK".
SHORT DESCRIPTION: This subroutine adds two images stored on buffers a, b and stores the result on buffer c. When an output pixel value exceeds 255, it is truncated to 255.

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: addc, diff, multc

8.2 addc()

Subroutine to add a constant to an image
LIBRARY MODE:

```
int addc(a,b, t, N1,M1,N2,M2)
image a,b;
int t;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
t: constant
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
addc a b t [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Addc". Fill-in the required fields, then click "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Addc". Fill-in the required fields, then click "OK".
SHORT DESCRIPTION: This subroutine adds a constant t to the image from buffer a and stores the result on buffer b. When an output pixel value exceeds 255, it is truncated to 255. It is also truncated to 0 if its value is less than 0.
RETURN VALUE: 0, -1, -21
SEE ALSO: add, multc, diff

8.3 and()

Subroutine to perform and on two images
LIBRARY MODE:

```
int and(a,b,c, N1,M1,N2,M2)
image a,b,c;
int N1,M1,N2,M2;

a,b: source image buffers
c: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
and a b c [N1 M1 N2 M2]
a,b,c: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "And". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "And". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine performs bitwise and between two images stored on buffers a,b and stores the result on buffer c.
RETURN VALUE: 0, -1, -21
SEE ALSO: or, not, xor, bindisp

8.4 bin()

Function to check if an image is binary
LIBRARY MODE:

```
int bin(a, N1,M1,N2,M2)
image a;
int N1,M1,N2,M2;

a: image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
bin a [N1 M1 M2 M2]
a: integer
```

SHORT DESCRIPTION: This function checks if an image is binary. If it is binary, it returns 0, otherwise the function returns 1. In interpreter mode the function types the corresponding messages.
RETURN VALUE: 0, 1, -1, -21
SEE ALSO: bindisp, overdisp, thres

8.5 clear()

Subroutine to clear a buffer
LIBRARY MODE:

```
int clear(a, t, N1,M1,N2,M2)
image a;
int t;
int N1,M1,N2,M2;

a: image buffer
t: new pixel value in the cleared buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
clear a t [N1 M1 N2 M2]
a: integer
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Clear". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Clear". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine clears buffer a. After this command each pixel on the buffer inside the region of interest has value t.
RETURN VALUE: 0, -1, -21
SEE ALSO: clearscreen, clearmatrix

8.6 clearmatrix()

Subroutine to fill a matrix with a value
INTERPRETER MODE:

```
int clearmatrix(matrix mat, float value, int N1, int M1, int N2, int M2)
matrix mat;
float value;
int N1,M1,N2,M2;

mat : matrix array
value : new value in the cleared matrix
N1,M1 : upper left  corner
N2,M2 : lower right corner
```

SHORT DESCRIPTION: This subroutine clears matrix mat. After this command each pixel on the buffer inside the region of interest has value "value".
RETURN VALUE: 0, -1, -21
SEE ALSO: clear

8.7 clip()

Subroutine to clip an image
LIBRARY MODE:

```
int clip(a,b, cmin,cmax, N1,M1,N2,M2)
image a,b;
int cmin,cmax;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
cmin,cmax: clip constants
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
clip a b cmin cmax [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Clip". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Clip". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine clips the pixel intensities of an image stored on buffer a and stores the result on buffer b. If a pixel value is below cmin or above cmax it becomes cmin and cmax respectively.
RETURN VALUE: 0, -1, -21, -41
SEE ALSO: thres, segm

8.8 copy()

Subroutine to copy buffers
LIBRARY MODE:

```
int copy(a,b, ND1,MD1, N1,M1,N2,M2)
image a,b;
int ND1,MD1;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
ND1,MD1: destination coordinates
N1,M1: upper left  corner coordinates of source buffer
N2,M2: lower right corner coordinates of source buffer
```

INTERPRETER MODE:

```
copy a b [ND1 MD1 N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Copy". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Copy". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutines copies the contents of buffer a on buffer b.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: copymatrix

8.9 copymatrix()

Subroutine to copy one matrix buffer to another.
INTERPRETER MODE:

```
int copymatrix(mats, matd, ND1,MD1, N1,M1,N2,M2)
matrix mats,matd;
int ND1,MD1;
int N1,M1,N2, M2;

mats : source matrix
matd : destination matrix
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrix
N2,M2 : lower right corner of source matrix
```

SHORT DESCRIPTION: This subroutines copies the contents of matrix mats on buffer matd.
RETURN VALUE: 0, -1, -21, -22, -36
SEE ALSO: copy

8.10 diff()

Subroutine to subtract two images a-b
LIBRARY MODE:

```
int diff(a,b,c, N1,M1,N2,M2)
image a,b,c;
int N1,M1 N2,M2;

a,b: source image buffers
c: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
diff a b c [N1 M1 N2 M2]
a,b,c: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Diff". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Diff". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine computes the difference a-b of two images stored on buffers a, b and stores the result on buffer c. When an output pixel value is less than 0, it is truncated to 0.
RETURN VALUE: 0, -1, -21
SEE ALSO: addc, add, multc

8.11 image2matrix()

Subroutine to copy an image to a matrix
INTERPRETER MODE:

```
int image2matrix(im, mat, ND1,MD1, N1,M1,N2,M2)
image im;
matrix mat;
int ND1,MD1;
int N1,M1,N2,M2;

im : source image
mat : destination matrix
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source image
N2,M2 : lower right corner of source image
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Image to Matrix". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutines copies the contents of image im on matrix mat.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: copy

8.12 matrix2image()

Subroutine to copy a matrix to an image
INTERPRETER MODE:

```
int matrix2image(mat, im, ND1,MD1, N1,M1,N2,M2)
matrix mat;
image im;
int ND1,MD1;
int N1,M1,N2,M2;

mat : source matrix
im : destination image
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrix
N2,M2 : lower right corner of source matrix
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Matrix to Image(trun)". Fill-in the required fields, then press "OK". If the user selects "Matrix to Image(norm)", then the values of the matrix will be normalized in the range 0 to 255.
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Matrix to Image". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutines copies the contents of matrix mat on image im. When the value of an element of the matrix exceeds 255, it is truncated to 255 at the destination image. If its value is less than 0, it is truncated to 0 at the destination image.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: copy

8.13 mix()

Subroutine to mix two images
LIBRARY MODE:

```
int mix(a,b,c, c1,c2, N1,M1,N2,M2)
image a,b,c;
float c1,c2;
int N1,M1,N2,M2;
```

```
a,b: source image buffer
c: destination image buffer
c1,c2: mixing coefficients
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
mix a b c c1 c2 N1 N2 M1 M2
a,b,c: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Mix". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Mix". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine is used to mix two images stored on buffers a, b and stores the result on buffer c. The percentage of the contribution of images a, b is c1% and c2% respectively. It is advisable that c1, c2 must lie in the range [0.0 .. 1.0] and their sum to be equal to unity. If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21
SEE ALSO: add, overmix

8.14 multc()

Subroutine to multiply an image by a constant
LIBRARY MODE:

```
int multc(a,b, f, N1,M1,N2,M2)
image a,b;
float f;
int N1,M1,N2,M2;
```

```
a: source image buffer
b: destination image buffer
f: constant (float)
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
multc a b f [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Multc". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Multc". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine multiplies each pixel value of an image stored on buffer a by a constant factor f and stores the result on buffer b. If an output pixel value exceeds 255, it is truncated to 255.
RETURN VALUE: 0, -1, -21, -90
SEE ALSO: add, addc, mix

8.15 neg()

Subroutine to get image negative
LIBRARY MODE:

```
int neg(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
neg a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Processing" and "Neg". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Processing" and "Neg". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine calculates the negative of a grayscale image stored on buffer a and stores the result on buffer b by assigning b[i][j]=255-a[i][j]. In case of binary images, similar operation is performed by the subroutine not.

RETURN VALUE: 0, -1, -21
SEE ALSO: not

8.16 nltran()

Subroutine for nonlinear point-wise image transformation
LIBRARY MODE:

```
int nltran(a,b, h, N1,M1,N2,M2)
image a,b;
 vector h;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
   h: transformation function
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
nltran a b h [N1 M1 N2 M2]
a,b: integers
h: integer (0 or 1)
```

SHORT DESCRIPTION: This subroutine performs nonlinear image transformation of an image stored on buffer a and stores the result on buffer b. The transformation function is stored on the signal buffer h. The size of this buffer must be 256 [0..255]. It must be loaded with the transformation function, before nltran is used.
RETURN VALUE: 0, -1, -2, -21
SEE ALSO: histeq, thres, clip

8.17 not()

Subroutine to perform logical not on a binary image

LIBRARY MODE:

```
int not(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
not a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Not". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Not". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine performs logical not operation on an image stored on buffer a and stores the result on buffer b. If the image is binary, it produces its negative. If the image is grayscale, its negative must be computed by subroutine neg!!!
RETURN VALUE: 0, -1, -21
SEE ALSO: neg, and, or, xor

8.18 or()

Subroutine to perform or on two images
LIBRARY MODE:

```
int or(a,b,c, N1,M1,N2,M2)
image a,b,c;
int N1,M1,N2,M2;

a,b: source image buffers
c: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
or a b c [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Or". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Or". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine performs bitwise or operation between two images stored on buffers a, b and stores the result on buffer c.
RETURN VALUE: 0, -1, -21
SEE ALSO: and, not, xor

8.19 overmix()

Subroutine to mix two images on the basis of a binary image
LIBRARY MODE:

```
int overmix(a,b,c,d, N1,M1,N2,M2)
image a,b,c,d;
int N1,M1,N2,M2;

a,b: source image buffers
c: source binary image buffer
d: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
overmix a b c d [N1 M1 N2 M2]
a,b,c,d: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "OverMix". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "OverMix". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine sends to the destination buffer d the pixels of the image stored on buffer a in the areas where c[i][j]=0 and the pixels of the image stored on buffer b in the areas where c[i][j]=1. Therefore, it mixes two

images on the basis of a binary image c. The image on buffer c must be binary. The subroutine checks if the image on c is binary and if this is not the case, it returns an error value. The main use of this subroutine is in videoart applications.
RETURN VALUE: 0, -1, -21, -100
SEE ALSO: mix, copy, thres

8.20 pixr()

Function to read a pixel value from a buffer
LIBRARY MODE:

```
int pixr(a, i,j)
image a;
int i,j;

a: image buffer
i,j: pixel coordinates
```

INTERPRETER MODE:

```
pixr a i j
a: integer
```

X–WINDOWS ENVIRONMENT: To have EIKONA report the value of a certain pixel, the item "Display Coord.", in the "File" menu, must be checked, and at least one image buffer should be visible. If it is checked, then in the modeless window that appears, the field "Level" displays the grayscale value of the point that the mouse pointer is on if the displayed buffer is grayscale, and fields R, G and B display the three color component values if the image is color. In this case, the color components can be transformed to other color spaces, by right-clicking on the coordinates window.
SHORT DESCRIPTION: This function reads the pixel value of buffer a, positioned at the i-th row, j-th column (a[i][j]) and returns it. In interpreter mode, it is displayed on the screen.
RETURN VALUE: -1, -44, pixel value (>=0)
SEE ALSO: pixw, dump, imdisp, bindisp, overdisp

8.21 pixw()

Subroutine to write a pixel value in a buffer
LIBRARY MODE:

```
int pixw(a, i,j, v)
image a;
int i,j;
int v;

a: image buffer
i,j: pixel coordinates
v: new pixel value
```

INTERPRETER MODE:

```
pixw a i j v
a: integer
```

MS–WINDOWS ENVIRONMENT: Open the "File" menu and then select "Write pixel".
SHORT DESCRIPTION: This subroutine sets the pixel value of buffer a, positioned at the i-th row, j-th column (a[i][j]) to the value v.
RETURN VALUE: 0, -1, -44
SEE ALSO: pixr, copy, clear

8.22 rotim()

Subroutine to rotate the image around (Xkm, Ykm) for rotation angle theta
LIBRARY MODE:

```
int rotim(a,b, theta, IC, JC, ND1,MD1, N1,M1,N2,M2)
image a,b;
float theta;
int IC, JC;
int ND1,MD1;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
```

```
theta: rotation angle (degrees)
IC,JC: rotation center coordinates
ND1,MD1: destination starting coordinates
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
rotim a b theta Xkm Ykm [ND1 MD1 N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Rotate". Fill-in the required fields, then press "OK".
–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Rotate". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine rotates the image of buffer a around the pixel (IC,JC) by angle theta. The resulting image is stored on buffer b. The new image center is at the position (IC+ND1, JC+MD1). The rotation operation is generally a time consuming operation. The rotated image contain isolated spikes that can be eliminated by using a median filter or a mode filter after the rotation. The destination buffer b must be different than the source buffer a in general.
RETURN VALUE: 0, -1, -21 SEE ALSO: copy, zoom, decim

8.23 thres()

Subroutine to threshold an image
LIBRARY MODE:

```
int thres(a,b, t, N1,M1,N2,M2)
image a,b;
int t;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
t: threshold
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
   thres a b t [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Threshold". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Threshold". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine thresholds an image stored on buffer a and produces a binary output that is stored on buffer b. Each pixel b[i][j] takes value 1 if the pixel a[i][j] is greater than or equal to threshold t, otherwise it takes value 0.
RETURN VALUE: 0, -1, -21
SEE ALSO: segm, bindisp, clip

8.24 xor()

Subroutine to perform xor on two images
LIBRARY MODE:

```
int xor(a,b,c, N1,M1,N2,M2)
image a,b,c;
int N1,M1,N2,M2;

a,b: source image buffers
c: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
xor a b c [N1 M1 N2 M2]
a,b,c: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Xor". Fill-in the required fields, then press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then select "Basic" and "Xor". Fill-in the required fields, then press "OK".
SHORT DESCRIPTION: This subroutine is used to perform bitwise xor between two images stored on buffers a and b. The result is stored on buffer c.

RETURN VALUE: 0, -1, -21
SEE ALSO: and, not, or

8.25 Test Programs

In order to test the subroutines thres() and not() from this chapter, along with some other from previous chapters, needed for the input – output and visualization of the results, we can use the program presented below:

```
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(argc,argv)
int argc;
char *argv[];

{  int N,M;
   image xin,xout;


   N=atoi(argv[1]);
   M=atoi(argv[2]);

   init_eikona(N,M,-1,-1,-1);
   xin=build_image(N,M);
   xout=build_image(N,M);
   fromdisk(argv[3],xin,0,0,N,M);
   thres(xin,xin,200,0,0,N,M);
   not(xin,xout,0,0,N,M);
   init_graphics("VRES16COLOR");
   bindisp(xin, 16,0,0,0,0,N,M);
   bindisp(xout, 16,0,N,0,0,N,M);
```

```
   getch();
   reset_graphics();
   printf("OK");
   dump(xin,"im1",0,0,N,M);
   dump(xout,"im2",0,0,N,M);
   return(0);
}
```

For this case, as well as for all the other test programs appearing in following chapters, we can obtain an executable code of the program by building an MS-DOS application using Microsoft Visual C++ 1.51. In order to do so we choose MS-DOS application (.EXE) as Project Type in the Project submenu of the Options menu, then choose Release for the Build Mode. The CPU for Code Generation in the Compiler submenu should be set to 80386 and the Disable Stack Checking option should not be set. We choose None in the Debug Information option as well as in the Browser Information option. The Memory Model should be set to Medium and the Optimizations to Default. Finally, the Distinguish Letter Case check box in the Input category of the Linker submenu should not be set. In the Project menu we choose New, set project type to MS-DOS application (.EXE), and then we choose a name for our project file. In the Edit window we add the source file, the eikdos.lib library, and the graphics.lib system library. To compile the project file, we choose Build in the Project menu. The test program can then be run from the command line. The syntax of the command line is:

`<program name> <Rows> <Columns> <Input file>`

After the program exits, the files: `im1`, `im2` will contain the dumped values of `<Input file>` image and its negative, respectively.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

9

Color transforms

9.1 cieRGB_to_XYZ()

Subroutine to perform cieRGB to XYZ transform
LIBRARY MODE:

```
int cieRGB_to_XYZ(r,g,b,x,y,z,N1,M1,N2,M2)
image r,g,b,x,y,z;
int N1,M1,N2,M2;
r,g,b: input image buffers
  x,y,z: transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE RGB". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE RGB". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.490 & 0.310 & 0.200 \\ 0.177 & 0.813 & 0.011 \\ 0.000 & 0.010 & 0.990 \end{bmatrix} \cdot \begin{bmatrix} R_{CIE} \\ G_{CIE} \\ B_{CIE} \end{bmatrix}$$

RETURN VALUE: 0, -1, -21

SEE ALSO: XYZ_to_cieRGB

9.2 ciexyY_to_XYZ()

Subroutine to perform CIE xyY to XYZ transform
LIBRARY MODE:

```
int ciexyY_to_XYZ(xi,yi,Y,x,y,z,N1,M1,N2,M2)
matrix xi,yi,Y;
image x,y,z;
int N1,M1;
int N2,M2;
xi,yi,Y: xyY transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE xyY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE xyY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs CIE xyY to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_ciexyY

9.3 cieuvY_to_XYZ()

Subroutine to perform CIE uvY to XYZ transform
LIBRARY MODE:

```
int cieuvY_to_XYZ(u,v,Yi,x,y,z,N1,M1,N2,M2)
matrix u,v,Yi;
```

```
image x,y,z;
int N1,M1;
int N2,M2;
u,v,Yi: uvY transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE uvY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE uvY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine perform CIE uvY to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_cieuvY

9.4 cieUVW_to_XYZ ()

Subroutine to perform CIE UVW to XYZ transform
LIBRARY MODE:

```
int cieUVW_to_XYZ(u,v,w,x,y,z,N1,M1,N2,M2)
matrix u,v,w;
image x,y,z;
int N1,M1;
int N2,M2;
u,v,w: UVW transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE UVW". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE UVW". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine performs CIE UVW to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_cieUVW

9.5 cieUsVsWs_to_XYZ()

Subroutine to perform CIE $U^*V^*W^*$ to XYZ transform
LIBRARY MODE:

```
int cieUsVsWs_to_XYZ(u,v,w,x,y,z,N1,M1,N2,M2)
matrix u,v,w;
image x,y,z;
int N1,M1;
int N2,M2;
u,v,w: U V W transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE U* V* W*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "CIE U* V* W*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs CIE $U^*V^*W^*$ to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_cieUsVsWs

9.6 CMY_to_RGB()

Subroutine to perform CMY to RGB transform
LIBRARY MODE:

```
int CMY_to_RGB(c,m,y,r,g,b,N1,M1,N2,M2)
image c,m,y,r,g,b;
int N1,M1;
int N2,M2;
c,m,y: transform buffers
  r,g,b: input image buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "CMY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "CMY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs CMY to RGB transform.
RETURN VALUE: 0, -1, -21
SEE ALSO: RGB_to_CMY

9.7 CMYK_to_RGB()

Subroutine to perform CMYK to RGB transform
LIBRARY MODE:

```
int CMYK_to_RGB(c,m,y,k,r,g,b,N1,M1,N2,M2)
image r,g,b,c,m,y,k;
int N1,M1;
int N2,M2;
c,m,y,k: transform buffers
  r,g,b: input image buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

SHORT DESCRIPTION: This subroutine performs CMYK to RGB transform.
RETURN VALUE: 0, -1, -21
SEE ALSO: RGB_to_CMYK

9.8 HLS_to_RGB()

Subroutine to perform HLS to RGB transform
LIBRARY MODE:

```
int HLS_to_RGB(h,l,s,r,g,b,N1,M1,N2,M2)
matrix h,l,s;
image r,g,b;
int N1,M1;
int N2,M2;
h,l,s: HLS transform buffers
  r,g,b: RGB buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HLS". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HLS". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs HLS to RGB transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: RGB_to_HLS

9.9 HSV_to_RGB()

Subroutine to perform HSV to RGB transform
LIBRARY MODE:

```
int HSV_to_RGB(h,s,v,r,g,b,N1,M1,N2,M2)
matrix h,s,v;
image r,g,b;
int N1,M1;
int N2,M2;
h,s,v: HSV transform buffers
  r,g,b: RGB buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HSV". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HSV". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs HSV to RGB transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: RGB_to_HSV

9.10 HSI_to_RGB()

Subroutine to perform HSI to RGB transform
LIBRARY MODE:

```
int HSI_to_RGB(h,s,I,r,g,b,N1,M1,N2,M2)
matrix h,s,I;
image r,g,b;
int N1,M1;
int N2,M2;
h,s,I: HSI transform buffers
  r,g,b: output buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HSI". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To RGB from.." and "HSI". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$V_1 = S \cos H$$

$$V_2 = S \sin H$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} \sqrt{3}/3 & 0 & 2/\sqrt{6} \\ \sqrt{3}/3 & 1/\sqrt{2} & -1/\sqrt{6} \\ \sqrt{3}/3 & -1/\sqrt{2} & -1/\sqrt{6} \end{bmatrix} \begin{bmatrix} I \\ V_1 \\ V_2 \end{bmatrix}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: RGB_to_HSI

9.11 Lsasbs_to_XYZ()

Subroutine to perform $L^*a^*b^*$ to XYZ transform
LIBRARY MODE:

```
int Lsasbs_to_XYZ(l,a,b,x,y,z,N1,M1,N2,M2)
matrix l,a,b;
image x,y,z;
int N1,M1;
int N2,M2;
l,a,b: L a b transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "L* a* b*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "L* a* b*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs $L^*a^*b^*$ to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_Lsasbs

9.12 Lsusvs_to_XYZ()

Subroutine to perform $L^*u^*v^*$ to XYZ transform
LIBRARY MODE:

```
int Lsusvs_to_XYZ(l,u,v,x,y,z,N1,M1,N2,M2)
matrix l,u,v;
image x,y,z;
int N1,M1;
int N2,M2;
l,u,v: L u v transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "L* u* v*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "L* u* v*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs $L^*u^*v^*$ to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_Lsusvs

9.13 ntscRGB_to_YIQ()

Subroutine to perform ntsc RGB to YIQ transform
LIBRARY MODE:

```
int ntscRGB_to_YIQ(r,g,b,y,ii,q,N1,M1,N2,M2)
image r,g,b;
matrix y,ii,q;
int N1,M1;
int N2,M2;
r,g,b: input image buffers
  y,ii,q: transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr." and "NTSC RGB to YIQ". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr." and "NTSC RGB to YIQ". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: YIQ_to_ntscRGB

9.14 ntscRGB_to_XYZ()

Subroutine to perform ntscRGB to XYZ transform
LIBRARY MODE:

```
int ntscRGB_to_XYZ(r,g,b,x,y,z,N1,M1,N2,M2)
image r,g,b,x,y,z;
int N1,M1,N2,M2;
r,g,b: input image buffers
  x,y,z: transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "NTSC RGB". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "NTSC RGB". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.607 & 0.174 & 0.201 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.117 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RETURN VALUE: 0, -1, -21
SEE ALSO: XYZ_to_ntscRGB

9.15 Points_cieRGB_to_XYZ()

Subroutine to perform cieRGB to XYZ transform for a pixel value.
LIBRARY MODE:

```
int Points_cieRGB_to_XYZ(R,G,B,X,Y,Z)
double R,G,B;
double *X,*Y,*Z;
R,G,B: input pixel values
X,Y,Z: output pixel values
```

MS–WINDOWS ENVIRONMENT: In the control window, click the *XYZ* button . Then the pixels values will be displayed at XYZ color space.
X–WINDOWS ENVIRONMENT: Right-click on the display coordinates window, and choose the XYZ choice. Then the pixels values will be displayed at XYZ color space.
SHORT DESCRIPTION: This subroutine calculates the cieRGB to XYZ transform for a pixel.
RETURN VALUE: 0
SEE ALSO: cieRGB_to_XYZ

9.16 Points_cieRGB_to_Lab()

Subroutine to perform cieRGB to Lab transform for a pixel value.
LIBRARY MODE:

```
int Points_cieRGB_to_Lab(R,G,B,L,a,b)
double R,G,B;
double *L,*a,*b;
R,G,B: input pixel values
L,a,b: output pixel values
```

MS–WINDOWS ENVIRONMENT: In the control window, click the *Lab* button . Then the pixels values will be displayed at Lab color space.
X–WINDOWS ENVIRONMENT: Right-click on the display coordinates window, and choose the Lab choice. Then the pixels values will be displayed at Lab color space.
SHORT DESCRIPTION: This subroutine calculates the cieRGB to Lab transform for a pixel.
RETURN VALUE: 0
SEE ALSO: cieRGB_to_Lsasbs

9.17 Points_cieRGB_to_HLS()

Subroutine to perform cieRGB to HLS transform for a pixel value.
LIBRARY MODE:

```
int Points_cieRGB_to_HLS(R,G,B,H,L,S)
double R,G,B;
double *H,*L,*S;
R,G,B: input pixel values
H,L,S: output pixel values
```

MS–WINDOWS ENVIRONMENT: In the control window, click the *HLS* button . Then the pixels values will be displayed at HLS color space.
X–WINDOWS ENVIRONMENT: Right-click on the display coordinates window, and choose the HLS choice. Then the pixels values will be displayed at HLS color space.
SHORT DESCRIPTION: This subroutine calculates the cieRGB to HLS transform for a pixel.
RETURN VALUE: 0
SEE ALSO: cieRGB_to_HLS

9.18 RGB_to_CMY()

Subroutine to perform RGB to CMY transform

LIBRARY MODE:

```
int RGB_to_CMY(r,g,b,c,m,y,N1,M1,N2,M2)
image r,g,b,c,m,y;
int N1,M1;
int N2,M2;
r,g,b: input image buffers
  c,m,y: transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "CMY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "CMY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$C = 1 - R$$

$$M = 1 - G$$

$$Y = 1 - B$$

RETURN VALUE: 0, -1, -21
SEE ALSO: CMY_to_RGB

9.19 RGB_to_CMYK()

Subroutine to perform RGB to CMYK transform
LIBRARY MODE:

```
int RGB_to_CMYK(r,g,b,c,m,y,k,N1,M1,N2,M2)
image r,g,b,c,m,y,k;
int N1,M1;
int N2,M2;
r,g,b: input image buffers
```

```
  c,m,y,k: transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$K = \min(C, M, Y)$$

$$C = C - K$$

$$M = M - K$$

$$Y = Y - K$$

RETURN VALUE: 0, -1, -21
SEE ALSO: CMYK_to_RGB

9.20 RGB_to_HSI()

Subroutine to perform RGB to HSI transform
LIBRARY MODE:

```
int RGB_to_HSI(r,g,b,h,s,I,N1,M1,N2,M2)
image r,g,b;
matrix h,s,I;
int N1,M1;
int N2,M2;
r,g,b: input buffers
  h,s,I: HSI transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HSI". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HSI". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{bmatrix} I \\ V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The rectangular coordinates (V_1, V_2) can be transformed to polar coordinates in the second step:

$$H = tan^{-1}(\frac{V_2}{V_1})$$

$$S = \sqrt{V_1^2 + V_2^2}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: HSI_to_RGB

9.21 RGB_to_HLS()

Subroutine to perform RGB to HLS transform
LIBRARY MODE:

```
int RGB_to_HLS(r,g,b,h,l,s,N1,M1,N2,M2)
image r,g,b;
matrix h,l,s;
int N1,M1;
int N2,M2;
r,g,b: input buffers
  h,l,s: HLS transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HLS". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HLS". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine performs RGB to HLS transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: HLS_to_RGB

9.22 RGB_to_HSV()

Subroutine to perform RGB to HSV transform
LIBRARY MODE:

```
int RGB_to_HSV(r,g,b,h,s,v,N1,M1,N2,M2)
image r,g,b;
matrix h,s,v;
int N1,M1;
int N2,M2;
r,g,b: input buffers
  h,s,v: HSV transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HSV". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From RGB to.." and "HSV". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs RGB to HSV transform
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: HSV_to_RGB

9.23 RGB_to_YUV()

Subroutine to perform RGB to YUV transform
LIBRARY MODE:

```
int RGB_to_YUV(in1, in2, in3, out1, out2, out3, iv , ih )
image in1,in2,in3,out1,out2,out3;
unsigned int ih,iv;\medskip

in1,in2,in3: Input RGB image buffers
out1,out2,out3: Output YUV image buffers
ih, iv : horizontal and vertical size of image.
```

SHORT DESCRIPTION : These functions transform an RGB image to an YUV image system.
RETURN VALUE: 0, -1, -21
SEE ALSO : RGB_to_Y( ).

9.24 RGB_to_Y( )

Subroutine to calculate the Luminance component Y from an RGB image.

LIBRARY MODE:

```
int RGB_to_Y(in1, in2, in3, out1, iv , ih )
image in1,in2,in3,out1;
unsigned int ih,iv;

in1,in2,in3: Input image buffers (RGB)
out1: output buffer containing the luminance
ih, iv : horizontal and vertical size of image.
```

SHORT DESCRIPTION : This function calculates the luminance component from an RGB color image and place it in a buffer.
RETURN VALUE: 0, -1, -21
SEE ALSO : RGB_to_YUV( ), YUV_to_RGB( ).

9.25 SthetaWs_to_XYZ()

Subroutine to perform $S\theta W$ to XYZ transform
LIBRARY MODE:

```
int SthetaWs_to_XYZ(s,t,w,x,y,z,N1,M1,N2,M2)
matrix s,t,w;
image x,y,z;
int N1,M1;
int N2,M2;
s,t,w: S theta W transform buffers
  x,y,z: XYZ transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "S theta W*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "To XYZ from.." and "S theta W*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs $S\theta W$ to XYZ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: XYZ_to_SthetaWs

9.26 UserDefinedTrans()

Subroutine to perform a user defined transform
LIBRARY MODE:

```
int UserDefinedTrans(in1,in2,in3,out1,out2,out3,f_mask,maskYsize,maskXsize,
ND1,MD1,N1,M1,N2,M2)
matrix in1,in2,in3;
matrix out1,out2,out3;
matrix f_mask;
int maskYsize,maskXsize;
int ND1,MD1;
int N1,M1;
int N2,M2;
in1,in2,in3: input matrix buffers
out1,out2,out3: output matrix buffers
f_mask: The tranformation matrix(user defined)
maskYsize,maskXsize: The dimensions of f_mask
ND1,MD1: destination coordinates in output matrix
```

```
N1,M1: upper left corner coordinates
N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu, then select "User Defined Transformation". Fill-in the required fields and press "OK". The user specifies the coefficients of the tranformation matrix in the win_f.par file.
SHORT DESCRIPTION: This subroutine calculates a User Defined Color Transformation.
RETURN VALUE: 0, -1, -21, -90.
SEE ALSO: cieRGB_to_XYZ

9.27 XYZ_to_cieRGB()

Subroutine to perform XYZ to cieRGB transform
LIBRARY MODE:

```
int XYZ_to_cieRGB(x,y,z,r,g,b,N1,M1,N2,M2)
image x,y,z,r,g,b;
int N1,M1;
int N2,M2;
x,y,z: transform buffers
  r,g,b: input image buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE RGB". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE RGB". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs the inverse XYZ to cieRGB transform.
RETURN VALUE: 0, -1, -21
SEE ALSO: cieRGB_to_XYZ

9.28 XYZ_to_ciexyY()

Subroutine to perform XYZ to CIE uvY transform
LIBRARY MODE:

```
int XYZ_to_ciexyY(x,y,z,xi,yi,Y,N1,M1,N2,M2)
image x,y,z;
matrix xi,yi,Y;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  xi,yi,Y: xyY transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE xyY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE xyY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs XYZ to CIE xyY transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: ciexyY_to_XYZ

9.29 XYZ_to_cieuvY()

Subroutine to perform XYZ to CIE uvY transform
LIBRARY MODE:

```
int XYZ_to_cieuvY(x,y,z,u,v,Y,N1,M1,N2,M2)
image x,y,z;
matrix u,v,Y;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  u,v,Y: uvY transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE uvY". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE uvY". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$u = \frac{4X}{X + 15Y + 3Z} = \frac{4x}{-2x + 12y + 3}$$

$$v = \frac{6Y}{X + 15Y + 3Z} = \frac{6y}{-2x + 12y + 3}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: cieuvY_to_XYZ

9.30 XYZ_to_cieUVW()

Subroutine to perform XYZ to CIE UVW transform
LIBRARY MODE:

```
int XYZ_to_cieUVW(x,y,z,u,v,w,N1,M1,N2,M2)
image x,y,z;
matrix u,v,w;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  u,v,w: UVW transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE UVW". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE UVW". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$U = \frac{2X}{3}$$

$$V = Y$$

$$W = \frac{-X + 3Y + Z}{2}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: cieUVW_to_XYZ

9.31 XYZ_to_cieUsVsWs()

Subroutine to perform XYZ to CIE $U^*V^*W^*$ transform
LIBRARY MODE:

```
int XYZ_to_cieUsVsWs(x,y,z,u,v,w,N1,M1,N2,M2)
image x,y,z;
matrix u,v,w;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  u,v,w: U V W transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE U* V* W*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "CIE U* V* W*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$W = 25(100Y)^{1/3} - 17 \qquad 1 \leq 100Y \leq 100$$

$$U = 13W(u - u_0)$$

$$V = 13W(v - v_0)$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: cieUsVsWs_to_XYZ

9.32 XYZ_to_Lsasbs()

Subroutine to perform XYZ to $L^*a^*b^*$ transform
LIBRARY MODE:

```
int XYZ_to_Lsasbs(x,y,z,l,a,b,N1,M1,N2,M2)
image x,y,z;
matrix l,a,b;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  l,a,b: L a b transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "L* a* b*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{aligned} L^* &= 116.04 f(Y/Y_0) - 16 \\ a^* &= 500[f(X/X_0) - f(Y/Y_0)] \\ b^* &= 200[f(Y/Y_0) - f(Z/Z_0)] \end{aligned}$$

where :

$$f(x) = \begin{cases} x^{\frac{1}{3}}, & x \geq 0.008856 \\ 7.787x + \frac{16}{116}, & x \leq 0.008856 \end{cases}$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: Lsasbs_to_XYZ

9.33 XYZ_to_Lsusvs()

Subroutine to perform XYZ to $L^*u^*v^*$ transform
LIBRARY MODE:

```
int XYZ_to_Lsusvs(x,y,z,l,u,v,N1,M1,N2,M2)
image x,y,z;
matrix l,u,v;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  l,u,v: L u v transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "L* u* v*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "L* u* v*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the following color transform:

$$\begin{aligned} L^* &= 116.04 f(Y/Y_0) - 16 \\ u^* &= 13L^*(u - u_0) \\ v^* &= 13L^*(v - v_0) \end{aligned}$$

where :

$$f(x) = \begin{cases} x^{\frac{1}{3}}, & x \geq 0.008856 \\ 7.787x + \frac{16}{116}, & x \leq 0.008856 \end{cases}$$

This subroutine calculates the following color transform:

$$L^* = 25(100Y/Y_0)^{1/3} - 16$$

$$u^* = 13L^*(u - u_0)$$

$$v^* = 13L^*(1.5v - v_0)$$

RETURN VALUE: 0, -1, -21, -22
SEE ALSO: Lsusvs_to_XYZ

9.34 XYZ_to_SthetaWs()

Subroutine to perform XYZ to $S\theta W$ transform
LIBRARY MODE:

```
int XYZ_to_SthetaWs(x,y,z,s,t,w,N1,M1,N2,M2)
image x,y,z;
matrix s,t,w;
int N1,M1;
int N2,M2;
x,y,z: XYZ transform buffers
  s,t,w: S theta W transform buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "S theta W*". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "S theta W*". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs XYZ to $S\theta W$ transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: SthetaWs_to_XYZ

9.35 XYZ_to_ntscRGB()

Subroutine to perform XYZ to ntscRGB transform
LIBRARY MODE:

```
int XYZ_to_ntscRGB(x,y,z,r,g,b,N1,M1,N2,M2)
image x,y,z,r,g,b;
int N1,M1;
int N2,M2;
x,y,z: transform buffers
  r,g,b: input image buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "NTSC RGB". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr.", "From XYZ to.." and "NTSC RGB". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs XYZ to ntscRGB transform.
RETURN VALUE: 0, -1, -21
SEE ALSO: ntscRGB_to_XYZ

9.36 YUV_to_RGB()

Subroutine to perform YUV to RGB transform
LIBRARY MODE:

```
int YUV_to_RGB(in1, in2, in3, out1, out2, out3, iv , ih )
image in1,in2,in3,out1,out2,out3;
unsigned int ih,iv;

in1,in2,in3: Input YUV image buffers
out1,out2,out3: Output RGB image buffers
ih, iv : horizontal and vertical size of image.
```

SHORT DESCRIPTION : These functions transform a YUV image to an RGB image system.
RETURN VALUE: 0, -1, -21
SEE ALSO : RGB_to_Y( ).

9.37 YIQ_to_ntscRGB()

Subroutine to perform YIQ to ntsc RGB transform
LIBRARY MODE:

```
int YIQ_to_ntscRGB(y,ii,q,r,g,b,N1,M1,N2,M2)
```

```
matrix y,ii,q;
image r,g,b;
int N1,M1;
int N2,M2;
y,ii,q: input buffers
  r,g,b: output image buffers
  N1,M1: upper left corner coordinates
  N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr." and "YIQ to NTSC RGB". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Color" menu and then, consecutively, select "Color Repr." and "YIQ to NTSC RGB". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs YIQ to ntsc RGB transform.
RETURN VALUE: 0, -1, -21, -22
SEE ALSO: ntscRGB_to_YIQ

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[WYZ67] G.W.Wyzecki, W.S.Stiles, *Color science*, Wiley, 1967.

[JAI89] A.K.Jain, *Fundamentals of digital image processing*, Prentice Hall, 1989.

[FOL90] J.D.Foley, A.van Dam, S.K.Feiner, J.F.Hughes, *Computers graphics: Principles and practice*, Addison-Wesley, 1990.

[HAR87] S.Harrington, *Computer graphics: A programming approach*, McGraw-Hill, 1987.

10

Image transform routines

10.1 arpsd()

Subroutine for 2-d AR modeling Power Spectral Density (PSD) estimation.
LIBRARY MODE:

```
int arpsd(ima,psd,N,M,K,LL,LR)
image ima;
matrix psd;
int N,M,K,LL,LR;

ima: input image buffer
psd: output PSD matrix
N,M: size of the image (end coordinates)
K,LL,LR: size of the support region of the AR model
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "AR PSD". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "AR PSD". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine computes the Power Spectral Density (PSD) by using 2d AR image modeling. It uses the subroutine arcoefficients() to calculate the 2d AR model coefficients. Finally, it uses fft2d() to calculate the power spectrum.

RETURN VALUE: 0, -1, -10, -21, -90, return of fft2d
SEE ALSO: arcoefficients, fft2d

10.2 blackman_tukey_psd()

Subroutine for Blackman-Tukey Method Power Spectral Density (PSD) estimation.
LIBRARY MODE:

```
int blackman_tukey_psd(ima,psd,WN,WM,N,M)
image ima;
matrix psd;
int WN,WM,N,M;

ima: input image buffer
psd: output PSD matrix
WN,WM: attenuation coefficients, the points in the N and M axes
          at which the window becomes zero
N,M: size of the image (end coordinates)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Blackman Tukey PSD". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Blackman Tukey PSD". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: Subroutine for Blackman-Tukey Method Power Spectral Density (PSD) estimation. It calculates the autocorrelation of the image. It performs windowing on it. Filally, it calculates the DFT of the windowed autocorrererelation. It must be noted that the matrix size must be double the size of the image.
RETURN VALUE: 0,-10,-21,-24,-33,return of correlation,windowcorrel,fft2d().
SEE ALSO: correlation, windowcorrel, fft2d, arpsd

10.3 correlation()

Subroutine to compute the correlation of two signals
LIBRARY MODE:

```
int correlation(mats1,mats2, matd, ND1,MD1, N1,M1,N2,M2)
matrix mats1,mats2,matd;
int ND1,MD1;
int N1,M1,N2,M2;

mats1,mats2 : source matrices
matd : destination matrix (correlation)
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Correlation". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Correlation". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine computes the correlation of two two-dimensional signals stored in matrices mats1 and mats2, and stores the result in matrix matd. The correlation is performed by using an efficient Fast Fourier Transform (fft2d). The size of all matrices is NMAXxMMAX. When the size of the region of interest is NxM, the size of the resulting correlation is 2Nx2M, where N=N2-N1 and M=M2-M1. This means that the maximum usable signal size is (NMAX/2)x(MMAX/2). For the calculation of the correlation, it is assumed that the input signals are periodic functions with periods N and M along the axes. In the resulting correlation matrix matd, a period of the periodic correlation function is stored, which has periods 2Nx2M along the axes. Taking into account that the value of the correlation at the lag (0,0) is stored at the element [ND1][MD1] of the matd matrix, and that the correlation function is periodic, it is easy to locate the element which holds the value of the correlation at every lag, e.g. the value at the lag (4,-5) is stored at matd[ND1+4][MD1+2*(M2-M1)-5]. Note that the subroutine correlation makes use of the subroutine fft2d and consequently the lengths N2-N1 and M2-M1 must be powers of 2. When the arguments mats1 and mats2 are identical, the subroutine computes the autocorrelation of the signal.
RETURN VALUE: 0, -10, -21, -22, return values of fft2d()
SEE ALSO: fft2d, convolution

10.4 dct()

Subroutine for the computation of the 2d DCT
LIBRARY MODE:

```
int dct(mat,COS,SIN,N,M)
matrix mat;
vector COS, SIN;
int N,M;

mat    : input, output matrix
COS,SIN: vectors containing cos(), sin() tables
N,M: image size
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "DCT". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "DCT". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine computes the 2d Discrete Cosine Transform. The computation is performed by using the 2d DFT. It uses the subroutine fft2d() to calculate the 2d FFT. The vectors COS,SIN contain the entries $cos(2 * \pi * i/4 * N)$ and $sin(2 * \pi * i/4 * N)$.
RETURN VALUE: 0,-1,-2,-21,-33,return of clearmatrix,fft2d.
SEE ALSO: fft2d, idct

10.5 fft()

Subroutine to compute the 1-Dimensional Discrete Fourier Transform
LIBRARY MODE:

```
int fft(xr,xi,n,wr,wi,mode)
float xr[],xi[];
int n;
float wr[],wi[];
int mode;

xr: real part
xi: imaginary part
```

```
n: size of signal
wr,wi: cos, sin coefficients
mode: 0 for forward FFT, 1 for inverse FFT
```

SHORT DESCRIPTION: Subroutine to compute the 1-Dimensional Discrete Fourier Transform. It performs input scrambling and calls fft1d() to perform the DFT calculation.
RETURN VALUE: 0
SEE ALSO: fft1d()

10.6 fft1d()

Subroutine to compute the 1d FFT
LIBRARY MODE:

```
int fft1d(RE,IM,N,WR,WI)
float RE[],IM[];
int N;
float WR[],WI[];
RE: real part
IM: imaginary part
N: size of signal
WR,WI: cos, sin coefficients
```

SHORT DESCRIPTION: This subroutine computes the 1d FFT. Input unscrambling must be performed externally before this subroutine is called. This routine is for internal use only (within rcfft).
RETURN VALUE: 0
SEE ALSO: rcfft

10.7 fft2d()

Subroutine to compute the 2-dimensional Discrete Fourier Transform
LIBRARY MODE:

```
int fft2d(matsre,matsim, matdre,matdim, mode, ND1,MD1,
          N1,M1,N2,M2)
matrix matsre,matsim,matdre,matdim;
int mode;
int ND1,MD1;
int N1,M1,N2,M2;

matsre,matsim : source matrices (real and imaginary
                components)
matdre,matdim : destination matrices (real and imaginary
                components)
mode : 0 forward / 1 inverse Transform
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and either "2-D FFT", or "Inverse 2-D FFT", to perform a forward or inverse 2-dimensional FFT, respectively. Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and either "2-D FFT", or "Inverse 2-D FFT", to perform a forward or inverse 2-dimensional FFT, respectively. Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine computes the Discrete Fourier Transform (DFT) of a signal having size NxM when mode is equal to 0. NxM is the size of the signal (N=N2-N1, M=M2-M1). When mode is equal to 1, the subroutine computes the inverse Discrete Fourier Transform of size NxM. The input matrices are always the matsre and matsim which correspond to the real and imaginary components. The results are always stored in matrices matdre and matdim either in forward or in inverse Fourier Transform. The algorithm used is the 2-d Row-Column Fast Fourier Transform (FFT). When an input signal is real, its imaginary component matrix must be filled by 0 (see clearmatrix command).

The lengths N2-N1 and M2-M1 must be powers of 2. If (N2-N1) == (M2-M1) fft2d() subroutine uses Vector Radix FFT vrfft(), else Row-Column FFT rcfft() is used.

RETURN VALUE: 0,-21,-22,-33,return values of rcfft, vrfft.

SEE ALSO: rcfft, vrfft, fft2image, reim2magnphase, matrix2image, image2matrix

10.8 fft2image()

Subroutine to compute the image of the magnitude of DFT
LIBRARY MODE:

```
int fft2image(matre,matim, immagn, ND1,MD1, N1,M1,N2,M2)
matrix matre,matim;
image immagn;
int ND1,MD1;
int N1,M1,N2,M2;

matre,matim : source matrices (real and imaginary
              components of DFT)
immagn : destination image
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

SHORT DESCRIPTION: This subroutine has as input (matre and matim) the real and imaginary components of the Discrete Fourier Transform of a signal, and computes an image (immagn) which represents a visualization of the magnitude of the transformed signal:
$im(i,j) = 255[log(1+|F(u,v)|^2)]/[log(1+max|F(u,v)|^2)]$ where $max|F(u,v)|^2$ is the maximum magnitude of the transformed signal over the Fourier domain. The output image provides a visualization of the signal frequency contents. Note that in the output image, the frequency (0,0) has been shifted at the center of the region of interest. Furthermore, the region of interest must cover the entire region of support of the transformed signal.
RETURN VALUE: 0, -1, -10, -21, -22
SEE ALSO: fft2d, reim2magnphase, matrix2image

10.9 fft_IO_mc()

Subroutine to perform Row - Column FFT for images stored on disk files.
LIBRARY MODE:

```
int fft_IO_mc(fr,fi,N1,N2,k,mode)
FILE *fr,*fi;
int N1,N2,k,mode;
fr,fi: stream files containing the real and imaginary part of
```

```
2-d input signal respectively. FFT output
is stored in these files at exit.
N1,N2 : image sizes. They MUST be powers of 2.
k     : number of lines to be read as a block in memory.
mode : 0 for forward, 1 for inverse DFT.
```

SHORT DESCRIPTION: Subroutine to perform Row - Column FFT for images stored on disk files. The stream files contain the real and imaginary part of 2-d input signal respectively. FFT output is stored in these files at exit. These files must be open before the subroutine is called. The image sizes must be powers of 2. The data is stored in the files in a row-wise manner: x(0,0) x(0,1) ... x(0,N2-1), x(1,0) .. etc. The number of lines to be read as a block in memory must be a power of 2. It computes the DFT when mode is equal to 0. When mode is equal to 1, the subroutine computes the inverse Discrete Fourier Transform. It uses the subroutine fft() to compute the 1d FFTs along rows and columns.
RETURN VALUE: 0,-10,-33.
SEE ALSO: fft(), fft_IO_trans()

10.10 fft_IO_trans()

Subroutine to perform Row - Column FFT of signal stored on disk files by fast matrix transposition.
LIBRARY MODE:

```
int fft_IO_trans(fr,fi,N,mode)
FILE *fr,*fi;
int N,mode;
Input matrices MUST be square!!!.
fr,fi: stream files containing the real and imaginary
part of 2-d input signal respectively. FFT output
is stored in these files at exit.
N : input signal size (square matrix).
mode  : 0 for forward, 1 for inverse DFT.
```

SHORT DESCRIPTION: Subroutine to perform Row - Column FFT of signal stored on disk files by fast matrix transposition. The input matrices MUST be square. The stream files must contain the real and imaginary part of 2-d input signal respectively. FFT output is stored in these files at exit. The files must be open before the call of the subroutine. Output is TRANSPOSED with respect to the input signal. The input signal size must be a power of 2. It computes the

DFT when mode is equal to 0. When mode is equal to 1, the subroutine computes the inverse Discrete Fourier Transform. It uses the subroutine fft() to compute the 1d FFTs along rows and columns.
RETURN VALUE: 0, -10, -21, -33
SEE ALSO: ffft, fft_IO_mc

10.11 fftc()

Subroutine to perform complex 1d FFT and IFFT (in place).
LIBRARY MODE:

```
int fftc(xr,xi,n,mode)
vector xr,xi;
int n;
int mode;

xr, xi: real and imaginary parts of input and output data
n: vector length
mode: 1 for fft, -1 for ifft
```

SHORT DESCRIPTION: This subroutine performs complex 1d FFT and IFFT (in place). Sine and cosine tables are computed internally.
RETURN VALUE: 0, -10, -90
SEE ALSO: fft, fft1d

10.12 init()

Subroutine to compute cos() and sin() which are used in dct().
LIBRARY MODE:

```
int init(COS,SIN,N)
vector COS,SIN;
int N;
```

```
COS,SIN: vectors to store cosine and sine functions
N: dimension of vectors COS,SIN.
```

SHORT DESCRIPTION: Subroutine to compute $cos(2 * \pi * i/4 * N)$ and $sin(2 * \pi * i/4 * N)$ which are used in dct().
RETURN VALUE: 0, -2, -90
SEE ALSO: dct, idct

10.13 idct()

Subroutine to compute the 2d Inverse DCT
LIBRARY MODE:

```
int idct(mat,COS,SIN,N,M)
matrix mat;
vector COS,SIN;
int N,M;

mat: input, output matrix
COS,SIN: matrices containing cos(), sin() tables
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Inverse DCT". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Inverse DCT". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: Subroutine to compute the Inverse Discrete Cosine Transform. The computation is performed by using the inverse 2d DFT. It uses the subroutine fft2d() to calculate the inverse 2d FFT. The vectors COS,SIN contain the entries $cos(2 * \pi * i/4 * N)$ and $sin(2 * \pi * i/4 * N)$.
RETURN VALUE: 0, -1, -2, -21, -33, return values of fft2d()
SEE ALSO: fft2d

10.14 magnphase2reim()

Subroutine to convert the magnitude-phase representation of a signal to the real-imaginary one.
LIBRARY MODE:

```
int magnphase2reim(matmagn,matphase, matre,matim,
                   ND1,MD1,N1,M1,N2,M2)
matrix matmagn,matphase,matre,matim;
int ND1,MD1;
int N1,M1,N2,M2;

matmagn,matphase : source matrices (magnitude and phase
                        components)
matre,matim : destination matrices (real and imaginary
                        components)
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

SHORT DESCRIPTION: This subroutine computes the real matre and imaginary matim components of a signal in the cartesian coordinates system, given the magnitude matmagn and phase matphase components of the signal in the polar coordinates system.
RETURN VALUE: 0, -1, -21, -22, -36
SEE ALSO: reim2magnphase, fft2d

10.15 ptfft()

Subroutine to perform polynomial transform FFT
LIBRARY MODE:

```
int ptfft(RE,IM,mode,rs,cs)
matrix RE;
matrix IM;
int mode,rs,cs;

RE: real matrix
IM: imaginary matrix
mode : 0 forward / 1 inverse Transform
```

```
rs,cs: number of rows and columns
```

SHORT DESCRIPTION: This subroutine performs the calculation of the 2d FFT by using the polynomial transform. It uses the subroutine pt() to perform the polynomial transform. It also uses the subroutine fft1d() to calculate the 1d FFTs that are needed in the algorithm. The bit reversion needed as well as the calculation of the cos() and sin() function tables is performed within ptfft().
RETURN VALUE: 0, -1, -10, -21, -33.
SEE ALSO: rcfft, vrfft, pt, fft1d

10.16 pt()

Subroutine to perform polynomial transform
LIBRARY MODE:

```
int pt(RE,IM,rs,cs)
matrix RE;
matrix IM;
int rs,cs;

RE: real matrix
IM: imaginary matrix
rs,cs: number of rows and columns
```

SHORT DESCRIPTION: This subroutine performs the polynomial transform. It is used in the subroutine ptfft() for the calculation of the 2d DFT.
RETURN VALUE: 0, -10
SEE ALSO: ptfft

10.17 rcfft()

Subroutine to compute the 2-dimensional row-column FFT
LIBRARY MODE:

```
int rcfft(matsre,matsim, matdre,matdim,
```

```
          mode, ND1,MD1, N1,M1,N2,M2)
matrix matsre,matsim,matdre,matdim;
int mode;
int ND1,MD1;
int N1,M1,N2,M2;

matsre,matsim : source matrices
                (real and imaginary components)
matdre,matdim : destination matrices
                (real and imaginary components)
mode : 0 forward / 1 inverse Transform
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

SHORT DESCRIPTION: This subroutine computes the 2-dimensional Discrete Fourier Transform by using the row-column decomposition method in conjuction with the 1-dimensional FFT. It computes the DFT of a signal having size NxM when mode is equal to 0. NxM is the size of the signal (N=N2-N1, M=M2-M1). When mode is equal to 1, the subroutine computes the inverse Discrete Fourier Transform of size NxM. The input matrices are always the matsre and matsim which correspond to the real and imaginary components. The results are always stored in matrices matdre and matdim either in forward or in inverse Fourier Transform. The algorithm used is the 2-d Row-Column Fast Fourier Transform (FFT). This subroutine uses the routine fft1d() to perform 1d FFTs. Row and column unscrambling is performed in rcfft() rather than within fft1d(). When an input signal is real, its imaginary component matrix must be filled by 0 (see clearmatrix command).
The lengths N2-N1 and M2-M1 must be powers of 2.
RETURN VALUE: 0, -1, -10, -21, -22, -33, -36,
SEE ALSO: fft2d, vrfft, ptfft, fft1d

10.18 real_2d_fft()

Subroutine to perform Row - Column FFT of a real two-dimensional signal.
LIBRARY MODE:

```
int real_2d_fft(x,N1,N2)
matrix x;
```

```
int N1,N2;

x : input and output image matrix.
N1,N2 : image size.
```

SHORT DESCRIPTION: Subroutine to perform Row - Column FFT of a real two-dimensional signal. The 2d signal size must be a power of 2. Furthermore, its size must be greater than 4x4. It uses subroutine fft() to perform row and column FFTs. Subroutine refft2images() can be used to unscramble the output of real_2d_fft().
RETURN VALUE: 0, -10, -21, -33.
SEE ALSO: fft2d, refft2images

10.19 refft2images()

Subroutine to unscramble the output of real_2d_fft().
LIBRARY MODE:

```
int refft2images(mat1,mat2,N1,N2)
matrix mat1,mat2;
int N1,N2;

mat1: input matrix. It contains the real 2-D DFT part at exit.
mat2: output matrix containing the imaginary 2-D DFT part.
N1,N2: image size
```

SHORT DESCRIPTION: This subroutine unscrambles the output of real_2d_fft().
RETURN VALUE: 0
SEE ALSO: real_2d_fft()

10.20 vrfft()

Subroutine to compute the 2-dimensional vector-radix 2x2 FFT
LIBRARY MODE:

```
int vrfft(matsre,matsim, matdre,matdim,
          mode, ND1,MD1, N1,M1,N2,M2)
matrix matsre,matsim,matdre,matdim;
int mode;
int ND1,MD1;
int N1,M1,N2,M2;

matsre,matsim : source matrices
                (real and imaginary components)
matdre,matdim : destination matrices
                (real and imaginary components)
mode : 0 forward / 1 inverse Transform
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

SHORT DESCRIPTION: This subroutine computes the 2-dimensional Discrete Fourier Transform by using the vector-radix 2x2 method. It computes the DFT of a signal having size NxM when mode is equal to 0. NxM is the size of the signal (N=N2-N1, M=M2-M1). When mode is equal to 1, the subroutine computes the inverse Discrete Fourier Transform of size NxM. The input matrices are always the matsre and matsim which correspond to the real and imaginary components. The results are always stored in matrices matdre and matdim either in forward or in inverse Fourier Transform. The algorithm used is the 2-d radix 2x2 FFT. The algorithm can operate only on square matrices (N=M).
RETURN VALUE: 0, -1, -10, -21, -22, -33, -34, -36
SEE ALSO: rcfft, ptfft

10.21 windowcorrel()

Subroutine to apply a 2-D window (pyramid) on a correlation matrix
LIBRARY MODE:

```
int windowcorrel(matin,matout,WN,WM,N,M)
matrix matin,matout;
int WN,WM,N,M;

matin: input correlation matrix
matout: output windowed correlation matrix
```

```
WN,WM: attenuation coefficients, the points in the N and M axes
       at which the window becomes zero
N,M: size of the correlation matrix
```

SHORT DESCRIPTION: This subroutine applies a 2-D window on a correlation matrix. This 2d window has theshape of a pyramid having base of size WNxWM.
RETURN VALUE: 0, -1, -24
SEE ALSO: blackman_tukey_psd

10.22 Test Programs

Subroutines from this chapter can be tested using the program below:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

/*------------------------------------------------------------------------*/
main(argc,argv)
int argc;
char *argv[];
{ int x,ret,choice,NN,MM;
  matrix mat1,mat2,mat3,mat4;
  image im1,im2,im3;
  FILE *fpr, *fpi;
  vector cs, ss;

/* initialize EIKONA, maximum image and matrix size */
  NN=atoi(argv[1]);
  MM=atoi(argv[2]);
  init_eikona(NN,MM,15,15,256);

/* build matrices of size NNxMM (of type float) */
  mat1=build_matrix(NN,MM);
  mat2=build_matrix(NN,MM);
```

```
  mat3=build_matrix(NN,MM);
  mat4=build_matrix(NN,MM);

/* build image buffers of size NNxMM (of type unsigned char) */
  im1=build_image(NN,MM);
  im2=build_image(NN,MM);
  im3=build_image(NN,MM);

/* check whether memory allocation is O.K. */
  if (mat1==NULL || mat2==NULL || mat3==NULL || mat4==NULL
      || im1==NULL || im2==NULL)
     { printf("build : Unable to allocate memory\n"); exit(1); }

  clearmatrix(mat1,0,0,0,NN,MM);
  clear(im1,0,0,0,NN,MM);
  clear(im2,0,0,0,NN,MM);
/* load an image of size NN/2 MM/2 */
  ret=fromdisk(argv[4],im1,0,0,NN/2,MM/2);
  if (ret!=0) { printf("fromdisk : error %d\n",ret); exit(2); }

/* copy the image buffer im1 to the matrix buffer mat1 */
  image2matrix(im1,mat1,0,0,0,0,NN/2,MM/2);

  choice=atoi(argv[3]);
  if(choice==1)
    {
   /* compute the auto-correlation and Power Spectrum of an image */
     ret=correlation(mat1,mat1,mat2,0,0,0,0,NN/2,MM/2);
     if(ret!=0) { printf("correlation : error %d\n",ret); exit(2); }
     clearmatrix(mat3,0,0,0,NN,MM);

     fft2d(mat2,mat3,mat2,mat3,0,0,0,0,0,NN,MM);
     blackman_tukey_psd(im1,mat1,NN/2,MM/2,NN/2,MM/2);
     arpsd(im1,mat4,NN,MM,16,8,7);
     clearmatrix(mat3,0,0,0,NN,MM);

     fft2image(mat2,mat3,im1,0,0,0,0,NN,MM);
     fft2image(mat1,mat3,im2,0,0,0,0,NN,MM);
     fft2image(mat4,mat3,im3,0,0,0,0,NN,MM);
     init_graphics("VRES16COLOR");
     imdisp(im1, 16,0,0,0,0,NN,MM);
     imdisp(im2, 16,0,MM,0,0,NN,MM);
     imdisp(im3, 16,NN,MM,0,0,NN,MM);
     getch();
```

```
    reset_graphics();
   }

 if(choice==2)
   {
  /* compute convolution */
  /* clear the matrix mat2 with 0 */
    clearmatrix(mat2,0,0,0,NN,MM);

  /* blur function : motion blur of length 6 */
    for(x=0; x<=2; x++)
      mat2[0][x]=(float)(1.0/5.0);
    for(x=62; x<=63; x++)
      mat2[0][x]=(float)(1.0/5.0);

  /* compute the convolution with the blur function stored on matrix mat2 */
    ret=convolution(mat1,mat2,mat1,0,0,0,0,NN/2,MM/2);
    if(ret!=0) { printf("convolution : error %d\n",ret); exit(3); }

  /* copy the convolution stored on buffer mat1 to the image buffer im2 */
    matrix2image(mat1,im2,0,0,0,0,NN,MM);
    init_graphics("VRES16COLOR");
    imdisp(im1, 16,0,0,0,0,NN,MM);
    imdisp(im2, 16,0,MM,0,0,NN,MM);
    getch();
    reset_graphics();

  /* store the blurred image to a file */
    todisk(im2,argv[5],0,0,NN,MM);
   }

 if(choice==3)
   {
  /* compute RCFFT */
    clearmatrix(mat2,0,0,0,NN,MM);
    rcfft(mat1,mat2,mat1,mat2,0,0,0,0,0,NN,MM);
    fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

    init_graphics("VRES16COLOR");
    imdisp(im1, 16,0,0,0,0,NN,MM);
    imdisp(im2, 16,0,MM,0,0,NN,MM);
    getch();
    reset_graphics();
```

```
  }
if(choice==4)
  {
/* compute RCFFT */
   real_2d_fft(mat1,NN,MM);
   refft2images(mat1,mat2,NN,MM);
   fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();
  }
if(choice==5)
  {
     /* compute VRFFT */
   clearmatrix(mat2,0,0,0,NN,MM);
   vrfft(mat1,mat2,mat1,mat2,0,0,0,0,0,NN,MM);
   fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();
  }
if(choice==6)
  {
     /* compute PTFFT */
   clearmatrix(mat2,0,0,0,NN,MM);
   ptfft(mat1,mat2,0,NN,MM);
   fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();
  }
if(choice==7)
  {
     /* compute 2D FFT under memory constraints */
   clearmatrix(mat2,0,0,0,NN,MM);
```

```
    savematrix(mat1, "tempr.dat", 0,0,NN,MM);
    savematrix(mat2, "tempi.dat", 0,0,NN,MM);
    if((fpr=fopen("tempr.dat","r+b"))==NULL) exit(-3);
    if((fpi=fopen("tempi.dat","r+b"))==NULL) exit(-3);
    fft_IO_mc(fpr,fpi,NN,MM,2,0);
    loadmatrix("tempr.dat", mat1, 0,0,NN,MM);
    loadmatrix("tempi.dat", mat2, 0,0,NN,MM);
    fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

    init_graphics("VRES16COLOR");
    imdisp(im1, 16,0,0,0,0,NN,MM);
    imdisp(im2, 16,0,MM,0,0,NN,MM);
    getch();
    reset_graphics();
   }
  if(choice==8)
   {
     /* compute 2D FFT by matrix transposition */
    clearmatrix(mat2,0,0,0,NN,MM);
    savematrix(mat1, "tempr.dat", 0,0,NN,MM);
    savematrix(mat2, "tempi.dat", 0,0,NN,MM);
    if((fpr=fopen("tempr.dat","r+b"))==NULL) exit(-3);
    if((fpi=fopen("tempi.dat","r+b"))==NULL) exit(-3);
    fft_IO_trans(fpr,fpi,NN,0);
    loadmatrix("tempr.dat", mat1, 0,0,NN,MM);
    loadmatrix("tempi.dat", mat2, 0,0,NN,MM);
    fft2image(mat1,mat2,im2,0,0,0,0,NN,MM);

    init_graphics("VRES16COLOR");
    imdisp(im1, 16,0,0,0,0,NN,MM);
    imdisp(im2, 16,0,MM,0,0,NN,MM);
    getch();
    reset_graphics();
   }
  if(choice==9)
   {
     /* compute DCT, IDCT */
    cs=build_vector(NN);
    ss=build_vector(NN);
    init(cs,ss,NN);
    DCT(mat1,cs,ss,NN,MM);
    IDCT(mat1,cs,ss,NN,MM);
    matrix2image(mat1,im2,0,0,0,0,NN,MM);
    init_graphics("VRES16COLOR");
```

```
     imdisp(im1, 16,0,0,0,0,NN,MM);
     imdisp(im2, 16,0,MM,0,0,NN,MM);
     getch();
     reset_graphics();
    }
    return(0);
}
```

The syntax of the command line for this program is:

```
<Program Name> <Rows>x2 <Columns>x2 <Choice> <Input File> [<Output File>]
```

And, depending on the value of `<Choice>`, these are the possible outcomes:
1: Periodogram, Blackman Tukey method Power Spectral Density and AR Power Spectral Density.
2: Blurring of an image using the convolution() subroutine. The output image is stored in `<Output File>`.
3: Row – Column FFT.
4: Row – Column FFT for real 2-dimensional signal.
5: Vector Radix FFT.
6: Polynomial Transform FFT.
7: Row – Column FFT for images stored on disk files.
8: Row –Column FFT by fast matrix transposition.
9: Discrete Cosine Transform and Inverse Discrete Cosine Transform.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[JAI89] A.K.Jain, *Fundamentals of digital image processing*, Prentice Hall, 1989.

11

Digital image filtering and enhancement

11.1 cdfhist()

Subroutine to calculate the cdf of a histogram
LIBRARY MODE:

```
int cdfhist(h,t)
vector h,t;

h: histogram input buffer
t: cdf output buffer
```

INTERPRETER MODE:

```
cdfhist h t
h,t: integers (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Cdf Histogram". Fill-in the required field and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Cdf Histogram". Fill-in the required field and press "OK".
SHORT DESCRIPTION: This subroutine calculates the cumulative density function of a histogram stored on signal buffer h and stores the result on signal buffer t. Both buffers have size 256 [0..255] and are of float type. Subroutine hist must be executed before cdfhist.

RETURN VALUE: 0, -2
SEE ALSO: hist, histeq, sigshow

11.2 conv()

Subroutine for 2-d convolution
LIBRARY MODE:

```
int conv(a,b, hcoe, NW,MW, N1,M1,N2,M2)
image a,b;
win_f hcoe;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
hcoe: 2-d filter coefficient buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
conv a b NW MW [N1 M1 N2 M2]
a,b: integers
```

SHORT DESCRIPTION: This subroutine calculates the convolution of an image stored on buffer a with the filter coefficients stored on the 2-d buffer hcoe having dimensions NWxMW. The result is stored on buffer b. The destination buffer b must be different from the input buffer a. When an output pixel value exceeds 255 it is truncated to 255. It is also truncated to 0, if its value is less than 0. The subroutine two_d_coeff must be used to load the coefficients on hcoe.
RETURN VALUE: 0, -1, -2, -21, -24
SEE ALSO: two_d_coeff, movav, median, l_filter, maxi, mini

11.3 convolution()

Subroutine to compute the convolution of two signals
LIBRARY MODE:

```
int convolution(mats1,mats2, matd, ND1,MD1, N1,M1,N2,M2)
matrix mats1,mats2,matd;
int ND1,MD1;
int N1,M1,N2,M2;

mats1,mats2 : source matrices
matd : destination matrix (convolution)
ND1,MD1 : destination coordinates
N1,M1 : upper left  corner of source matrices
N2,M2 : lower right corner of source matrices
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Convolution". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Transforms" and "Convolution". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine computes the convolution of two two-dimensional signals stored in matrices mats1 and mats2, and stores the result in matrix matd. The convolution is performed by using an efficient Fast Fourier Transform (fft2d). The size of all matrices is NMAXxMMAX. When the size of the region of interest is NxM, the size of the resulting convolution is 2Nx2M, where N=N2-N1 and M=M2-M1. This means that the maximum usable signal size is (NMAX/2)x(MMAX/2). For the calculation of the convolution, it is assumed that the input signals are periodic functions with periods N and M along the axes. In the resulting convolution matrix matd, a period of the periodic convolution function is stored, which has periods 2Nx2M along the axes. Taking into account that the value of the convolution at the point (0,0) is stored at the element [ND1][MD1] of the matd matrix, and that the convolution function is periodic, it is easy to locate the element which holds the value of the convolution at every point, e.g. the value at the point (4,-5) is stored at matd[ND1+4][MD1+2*(M2-M1)-5]. Note that the subroutine convolution makes use of the subroutine fft2d and consequently the lengths N2-N1 and M2-M1 must be powers of 2.
RETURN VALUE: 0, -1, -21, -22, -33, return values of fft2d()
SEE ALSO: fft2d, correlation

11.4 decim()

Subroutine to decimate an image
LIBRARY MODE:

```
int decim(a,b, t, ND1,MD1, N1,M1,N2,M2)
image a,b;
int t;
int ND1,MD1;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
t: decimation factor
ND1,MD1: destination coordinates
N1,M1: upper left  corner coordinates of source buffer
N2,M2: lower right corner coordinates of source buffer
```

INTERPRETER MODE:

```
   decim a b t [ND1 MD1 N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: For B/W images, open the "Black and White" menu, then consecutively select "Processing" and "Decim". For color images, open the "Color" menu, then consecutively select "Basic" and "Decim". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: For B/W images, open the "Black and White" menu, then consecutively select "Processing" and "Decim". For color images, open the "Color" menu, then consecutively select "Basic" and "Decim". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine decimates (shortens) the image stored on buffer a by a factor t. The result is stored on buffer b. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21, -90
SEE ALSO: zoom

11.5 dither()

Subroutine to perform grayscale image dithering.
LIBRARY MODE:

```
int dither(a,b,size,N1,M1,N2,M2)
image a,b;
int size;
int N1,M1,N2,M2;

a: input grayscale image buffer
b: output binary image buffer
size: size of dither matrix
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine performs dithering by using a dither matrix of size "size". First the dither matrix is constructed and its elements are scaled. Then the dither matrix is applied to the grayscale image to produce the binary output. The size of the dither matrix must be greater than 1.
RETURN VALUE: 0, -1, -10, -21, -90
SEE ALSO: halftone

11.6 halftone()

Subroutine to halftone an image
LIBRARY MODE:

```
int halftone(a,b, cn, N1,M1,N2,M2)
image a,b;
int cn;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
cn: number of halftones (4, 9, 16)
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
halftone a b cn [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Basic" and "Halftone". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Basic" and "Halftone". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to produce the halftone of an image stored on buffer a. The new image is stored on buffer b. The destination buffer b must be different from the source buffer a. The produced image is binary and its dimensions are the same with the dimensions of the original grayscale image. Therefore it is, in a sense, a decimated version of the original image. The number of halftones cn determines the density of the dots in the halftoned image and must be power of an integer (4, 9, 16). The higher the number cn is the lower the resolution of the halftoned image is. Halftoned images can be used to produce hard copies of grayscale images on a dot printer or a laser printer. The halftoned image can be viewed on screen by the command bindisp.
RETURN VALUE: 0, -1, -21, -62
SEE ALSO: printim, bindisp.

11.7 hist()

Subroutine to calculate the histogram of an image
LIBRARY MODE:

```
int hist(a, h, N1,M1,N2,M2)
image a;
vector h;
int N1,M1,N2,M2;

a: image buffer
h: histogram buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
hist a h [N1 M1 N2 M2]
a: integer
h: integer (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Histogram". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Histogram". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the histogram (i.e. the probability density function) of the image intensities stored on buffer a. The histogram produced is a function stored on signal buffer h of float type. Its dimension is 256 [0..255] since 256 are the allowable gray levels. Its values lie in the range [0.0 .. 1.0]. The histogram can be plotted on screen by using the command sigshow. It can be stored on a disk file by using the command dump_signal.
RETURN VALUE: 0, -1, -2, -10, -21
SEE ALSO: sigshow, dump_signal, cdfhist, histeq, segm, thres

11.8 histeq()

Subroutine for histogram equalization
LIBRARY MODE:

```
int histeq(a,b, h, N1,M1,N2,M2)
image a,b;
vector h;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
h: histogram cdf function
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
histeq a b h [N1 M1 N2 M2]
a,b: integers
h: integer (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Histeq". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Histeq". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine performs histogram equalization of an image stored on buffer a and stores the result on buffer b. The produced output is an image whose histogram is uniform. Therefore it enhances the contrast in an image. The commands hist and cdfhist must be executed before histeq!!!
RETURN VALUE: 0, -1, -2, -10, -21
SEE ALSO: hist, cdfhist, sharp

11.9 interpolation()

Subroutine to interpolate an image
LIBRARY MODE:

```
int interpolation(a,b,t,order,ND1,MD1,N1,M1,N2,M2);
image a,b;
float t;
int order;
int ND1,MD1;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
t: zoom factor
order: interpolation mode
ND1,MD1: destination coordinates
N1,M1: upper left  corner of source buffer
N2,M2: lower right corner of source buffer
```

SHORT DESCRIPTION: This subroutine is used to interpolate an image stored on buffer a and stores the result on buffer b. The zooming factor is t ($t >= 1$) and can be float. The order of interpolation is controled by the variable "order". "Order" can take the integer values 0,1,2,3 and zero-order, linear, bell-shaped and cubic B-spline interpolation is correspondingly applied. If $t < 1$, order<0 or > 3, or the resulting image exceeds the dimensions of buffer b, the subroutine returns an error value. Image shortening can be performed by subroutine decim.
RETURN VALUE: 0, -1, -21, -22, -90
SEE ALSO: decim, zoom

11.10 L2_error_norm()

Subroutine for L2 error norm calculation
LIBRARY MODE:

```
int L2_error_norm(a,b,s,N1,M1,N2,M2)
image a,b;
float *s;
int N1,M1,N2,M2;

a,b: image buffers
s: L2 error
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "L2 Error Norm". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "L2 Error Norm". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the L2 error norm between two images a,b:

$$s = \frac{\sum_{k=N_1}^{k=N_2} \sum_{l=M_1}^{l=M_2} (a[k][l] - b[k][l])^2}{(N2 - N1)(M2 - M1)}$$

It is used for the evaluation of image filtering and coding algorithms. RETURN VALUE: 0,-1, -21
SEE ALSO: L2_error_ratio, snr

11.11 L2_error_ratio()

Subroutine for L2 error ratio calculation in dB
LIBRARY MODE:

```
int L2_error_ratio(a,b,c,d,s,N1,M1,N2,M2)
image a,b,c,d;
float *s;
int N1,M1,N2,M2;
```

```
a,b,c,d: image buffers
s: L2 error ratio
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "L2 Error Ratio". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "L2 Error Ratio". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates L2 error ratio (in dB) between two pairs of images a,b and c,d:

$$s = \log \frac{\sum_{k=N_1}^{k=N_2} \sum_{l=M_1}^{l=M_2} (a[k][l] - b[k][l])^2}{\sum_{k=N_1}^{k=N_2} \sum_{l=M_1}^{l=M_2} (c[k][l] - d[k][l])^2}$$

It is used for the evaluation of image filtering and coding algorithms.
RETURN VALUE: 0, -1, -21, -87
SEE ALSO: L2_error_norm, snr

11.12 l_filter()

Subroutine for 2-d l-filtering
LIBRARY MODE:

```
int l_filter(a,b, h, NW,MW, N1,M1,N2,M2)
image a,b;
vector h;
int NW,MW;
int N1,M1,N2,M2;
```

```
a: source image buffer
b: destination image buffer
h: buffer of filter coefficients
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
l_filter a b h NW MW [N1 M1 N2 M2]
a,b: integers
h: integer (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "L Filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "L Filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates L-filter output of an image stored on buffer a and stores the result on buffer b. The filter is of dimensions NWxMW. The filter coefficients are stored on the one-dimensional buffer h. Their number is NWxMW. According to the choice of the filter coefficients, it can perform filtering or edge detection. The coefficient set [0.5 0 0 0 0 0 0 0 0.5] corresponds to a midpoint filter of size 3x3. The coefficient set [-0.125 -0.125 -0.125 -0.125 0 0.125 0.125 0.125 0.125] corresponds to the dispersion edge detector. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -2, -10, -21, -24
SEE ALSO: maxi, mini, order, range

11.13 matrixcdfhist()

Subroutine to compute the cdf histogram of a matrix.
LIBRARY MODE:

```
int matrixcdfhist(vector vect, int NSIG)
vector vect;
int NSIG;

vect: array of histogram values (input and output)
NSIG: size of vector
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Matrix Cdf Histogram". Fill in the required field and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Matrix Cdf Histogram". Fill in the required field and press "OK".

SHORT DESCRIPTION: This subroutine calculates the cumulative density function of a matrix histogram stored on signal buffer vect and stores the result on the same buffer. The buffer have size NSIG [0..NSIG] and is of float type.
RETURN VALUE: 0, -2, -90
SEE ALSO: matrixhist, hist, histeq

11.14 matrixhist()

Subroutine to calculate the histogram of a matrix
LIBRARY MODE:

```
int matrixhist(mat, vect[], NSIG, N1, M1, N2, M2)
matrix mat;
vector vect[];
int NSIG;
int N1,M1,N2,M2

mat: pointer to matrix (input)
vect: array of histogram values (output)
NSIG: size of vector
N1,M1: upper left corner coordinates
N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Matrix Histogram". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Matrix Histogram". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the histogram (i.e. the probability density function) of the matrix values stored on matrix mat. The histogram produced is a function stored on signal buffer vect[1] of float type. Its dimension is NSIG [0..NSIG]. Its values lie in the range [0.0 .. 1.0] due to normalization in order to give pdf. It can be stored on a disk file by using the command dump_matrix_hist. In vect[0] we store the x[i] values. In order to calculate these values we take the minimum and maximum matrix values and we divide this space in NSIG intervals.
RETURN VALUE: 0, -1, -2, -21, -90
SEE ALSO: matrixcdfhist, dump_matrix_hist, dump_signal, cdfhist

11.15 maxi()

Subroutine for 2-d max filtering
LIBRARY MODE:

```
int maxi(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
maxi a b NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Maxi". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Maxi". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to produce the local maximum filtering (dilation) of an image stored on buffer a and stores the result on buffer b. The filter size is NWxMW. The destination buffer b must be different from the source buffer a. This subroutine is a special case of the subroutines order, imdilate and l_filter. It is faster than these subroutines.
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: mini, imdilate, imerode, l_filter, order, range

11.16 median()

Subroutine for 2-d median filtering
LIBRARY MODE:

```
int median(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
median a b NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Median". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to perform median filtering of an image stored on buffer a and stores the result on buffer b. The filter size is NWxMW. The destination buffer b must be different from the source buffer a. This subroutine is a special case of the subroutines order and l_filter. It is faster than these subroutines. This subroutine is very useful for impulsive noise removal and for edge preservation purposes.
RETURN VALUE: 0, -1, -10, -21, -24
SEE ALSO: l_filter, order, movav

11.17 mini()

Subroutine for 2-d min filtering
LIBRARY MODE:

```
int mini(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
mini a b NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Mini". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Mini". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to produce the local minimum filtering (erosion) of an image stored on buffer a and stores the result on buffer b. The filter size is NWxMW. The destination buffer b must be different from the source buffer a. This subroutine is a special case of the subroutines order, imerode and l_filter. It is faster than these subroutines.
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: maxi, imdilate, imerode, l_filter, order, range

11.18 movav()

Subroutine for 2-d moving average filtering
LIBRARY MODE:

```
int movav(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;
```

```
a: source image buffer
b: destination image buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
movav a b NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Movav". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Movav". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to perform moving average filtering of an image stored on buffer a and stores the result on buffer b. The filter size is NWxMW. The destination buffer b must be different from the source buffer a. This subroutine is a special case of the subroutines conv and l_filter. This subroutine performs well in additive Gaussian noise filtering.
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: l_filter, order, median, conv

11.19 noise_add_gauss()

Subroutine to add Gaussian noise to an image
LIBRARY MODE:

```
int noise_add_gauss(a,b, sigma, N1,M1,N2,M2)
image a,b;
float sigma;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
sigma: noise standard deviation
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_add_gauss a b sigma [N1 M1 N2 M2]
a,b:integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Gauss". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Gauss". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine adds additive Gaussian noise of standard deviation sigma on an image stored on buffer a and stores the result on buffer b. The main use of this subroutine is for simulation purposes. This subroutine uses the random number generator rand(). If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_laplace, noise_add_uni, noise_imp, noise_mult_gauss, noise_mult_uni

---

11.20 noise_add_laplace()

Subroutine to add Laplacian noise to an image
LIBRARY MODE:

```
int noise_add_laplace(a,b, v, N1,M1,N2,M2)
image a,b;
float v;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
v: distribution coefficient
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_add_laplace a b v [N1 M1 N2 M2]
a,b:integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Laplace". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Laplace". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine adds additive Laplacian noise on an image stored on buffer a and stores the result on buffer b. The Laplacian noise distribution has coefficient v. The noise variance is equal to v*sqrt(2). The main use of this subroutine is for simulation purposes. This subroutine uses the random number generator rand(). If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_gauss, noise_add_uni, noise_imp, noise_mult_gauss, noise_mult_uni

---

11.21 noise_add_uni()

Subroutine to add uniform noise to an image
LIBRARY MODE:

```
int noise_add_uni(a,b, v, N1,M1,N2,M2)
image a,b;
float v;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
v: noise range [-v/2, v/2]
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_add_uni a b v [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Uni". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Add Uni". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine adds additive uniform noise on an image stored on buffer a and stores the result on buffer b. The uniform noise distribution is in the range [-v/2, v/2]. The main use of this subroutine is for simulation purposes. This subroutine uses the random number generator rand(). If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_gauss, noise_add_laplace, noise_imp, noise_mult_gauss, noise_mult_uni

---

11.22 noise_imp()

Subroutine to add impulsive noise to an image
LIBRARY MODE:

```
int noise_imp(a,b, p, imin,imax, N1,M1,N2,M2)
image a,b;
float p;
int imin,imax;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
p: noise probability
imin: minimum impulse value
imax: maximum impulse value
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_imp a b p imin imax [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Impulsive". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Impulsive". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine adds impulsive salt and pepper noise on an image stored on buffer a and stores the result on buffer b. The probability of noise occurence is p and the probability of positive and negative spikes is equal p/2. The value of the negative and positive spikes is imin and imax respectively. They must lie in the range [0..255]. This subroutine uses the random number generator rand(). The main purpose of this subroutine is noise simulation.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_gauss, noise_add_laplace, noise_add_uni, noise_mult_gauss, noise_mult_uni

---

11.23 noise_mult_gauss()

Subroutine to add multiplicative Gaussian noise to an image
LIBRARY MODE:

```
int noise_mult_gauss(a,b, v, N1,M1,N2,M2)
image a,b;
float v;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
v: standard deviation of a zero mean Gaussian random variable
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_mult_gauss a b v [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Mult Gauss". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Mult Gauss". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine adds Gaussian multiplicative noise on an image stored on buffer a and stores the result on buffer b. The noisy output b[i][j] is equal to the input a[i][j] multiplied by a zero mean Gaussian random variable having standard deviation v. The main use of this subroutine is for simulation purposes. This subroutine uses the random number generator rand(). If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_gauss, noise_add_laplace, noise_add_uni, noise_imp, noise_mult_uni

---

11.24 noise_mult_uni()

Subroutine to add multiplicative uniform noise to an image
LIBRARY MODE:

```
int noise_mult_uni(a,b, v, N1,M1,N2,M2)
image a,b;
float v;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
v: uniform random variable range [-v/2, v/2]
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
noise_mult_uni a b v [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Mult Uni". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Mult Uni". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine adds multiplicative uniform noise on an image stored on buffer a and stores the result on buffer b. The noisy output

b[i][j] is equal to the input a[i][j] multiplied by a uniform random variable in the range [-v/2, v/2]. The main use of this subroutine is for simulation purposes. This subroutine uses the random number generator rand(). If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively.
RETURN VALUE: 0, -1, -21, -80
SEE ALSO: noise_add_gauss, noise_add_laplace, noise_add_uni, noise_imp, noise_mult_gauss

---

11.25 order()

Subroutine for 2-d order statistics filtering
LIBRARY MODE:

```
int order(a,b, ord, NW,MW, N1,M1,N2,M2)
image a,b;
int ord;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
ord: order statistic (1<=ord<=NW*MW)
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
order a b ord NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Order". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering" and "Order". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to perform order statistics filtering on an image stored on buffer a and stores the result on buffer b. The filter size is NWxMW. The destination buffer b must be different from the source

buffer a. This subroutine is a special case of the subroutine l_filter. Special subroutines for median, max and min filtering exist.
RETURN VALUE: 0, -1, -10, -21, -24, -25
SEE ALSO: median, maxi, mini, l_filter

11.26 overlap_add()

Subroutine to perform overlap-add convolution.
LIBRARY MODE:

```
overlap_add(x,h,y,NL,ML,N1,M1,N2,M2,Nh,Mh)
matrix x,h,y;
int NL,ML,N1,M1,N2,M2,Nh,Mh;

x: input image matrix
h: filter impulse response matrix
y: output image matrix
NL ,ML: block size
N1, M1: start coordinates
N2 ,M2: end coordinates
Nh ,Mh: size of impulse response matrix h
```

SHORT DESCRIPTION: Subroutine to perform overlap-add convolution between the signal x and the impulse response h. The subroutine uses rcfft() for the calculation of the convolution among the various blocks of x and h. The 2d FFT size is power of 2 larger or equal to (Nh+NL-1,Mh+ML-1). It returns -90 if (N2-N1)%NL!=0 or (M2-M1)%ML!=0.
RETURN VALUE: 0, -10, -21, -90
SEE ALSO: conv, convolution, overlap_add_c, overlap_save

11.27 overlap_add_c()

Subroutine to perform overlap-add convolution of complex signals.
LIBRARY MODE:

```
overlap_add_c(xr,xi,hr,hi,yr,yi,NL,ML,N1,M1,N2,M2,Nh,Mh)
matrix xr,xi,hr,hi,yr,yi;
int NL,ML,N1,M1,N2,M2,Nh,Mh;

xr,xi: input matrices
hr,hi: filter impulse response matrices
yr,yi: output image matrices
N1, M1: start coordinates
N2 ,M2: end coordinates
Nh ,Mh: size of impulse response matrix h
NL ,ML: block size
```

SHORT DESCRIPTION: Subroutine to perform overlap-add convolution between the complex signal (xr, xi) and the complex impulse response (hr,hi). The subroutine uses rcfft() for the calculation of the convolution among the various blocks of x and h. The 2d FFT size is power of 2 larger or equal to (Nh+NL-1,Mh+ML-1). It returns -90 if (N2-N1)%NL!=0 or (M2-M1)%ML!=0.
RETURN VALUE: 0, -10, -21, -90
SEE ALSO: conv, convolution, overlap_add, overlap_save

11.28 overlap_save()

Subroutine to perform overlap-save convolution.
LIBRARY MODE:

```
overlap_save(x,h,y,NL,ML,N1,M1,N2,M2,Nh,Mh)
matrix x,h,y;
int NL,ML,N1,M1,N2,M2,Nh,Mh;

x: input image matrix
h: filter impulse response matrix
y: output image matrix
N1, M1: start coordinates
N2 ,M2: end coordinates
Nh ,Mh: size of impulse response matrix h
NL ,ML: block size
```

SHORT DESCRIPTION: Subroutine to perform overlap-save convolution between the signal x and the impulse response h. The subroutine uses rcfft() for the calculation of the convolution among the various blocks of x and h. The 2d

FFT size is equal to (Nh+NL-1,Mh+ML-1) and must be power of 2. Otherwise, it returns -33.
RETURN VALUE: 0, -1, -10, -21, -33, -90
SEE ALSO: conv, convolution, overlap_add_c, overlap_add

11.29 overlap_save_c()

Subroutine to perform overlap-save convolution of complex signals.
LIBRARY MODE:

```
overlap_save_c(xr,xi,hr,hi,yr,yi,NL,ML,N1,M1,N2,M2,Nh,Mh)
matrix xr,xi,hr,hi,yr,yi;
int NL,ML,N1,M1,N2,M2,Nh,Mh;

xr,xi : two-dimensional complex signal
hr,hi : two-dimensional impulse response
yr,yi : two-dimensional response
NL , ML : y[ij] block size.
N1, M1: start coordinates
N2 ,M2: end coordinates
Nh , Mh : define region of support for h as [0,Nh)x[0,Mh)
```

SHORT DESCRIPTION: Subroutine to perform overlap-save convolution between the complex signal (xr,xi) and the complex impulse response (hr,hi). The subroutine uses rcfft() for the calculation of the convolution among the various blocks of x and h. The 2d FFT size is equal to (Nh+NL-1,Mh+ML-1) and must be power of 2. Otherwise, it returns -33.
RETURN VALUE: 0, -1, -10, -21, -33, -90
SEE ALSO: conv, convolution, overlap_add_c, overlap_save, overlap_add

---

11.30 searchmathistval()

Subroutine to find the X location of a matrix value. It uses a variation of binary search.(INTERNAL USE ONLY)

LIBRARY MODE:

```
int searchmathistval(matvalue, vect[], start, end)
float matvalue;
vector vect[];
int start;
int end;

matvalue: the value of the matrix
vect: array of histogram values
start,end: start and end points for binary search
```

SHORT DESCRIPTION: This subroutine is used to calculate the X location of a matrix value. It uses a variation of binary search and is recursive. It is used by matrixhist.
RETURN VALUE: The X location of matvalue on vect[0].
SEE ALSO: matrixhist, matrixcdfhist, dump_matrix_hist

11.31 sort()

Subroutine to sort an integer array
LIBRARY MODE:

```
int sort(t, NM)
int far* t;
int NM;
t: integer array
NM: array size
```

SHORT DESCRIPTION: This subroutine sorts an integer array. It uses bubble sorting. It is internally used in a number of nonlinear filtering operations e.g. median() and order(). NM must be greater than 2.
RETURN VALUE: 0, -2, -90
SEE ALSO: median, order

11.32 sharp()

Subroutine for 2-d image sharpening
LIBRARY MODE:

```
int sharp(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
sharp a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Sharp". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Processing" and "Sharp". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to perform local image sharpening (contrast enhancement) of an image stored on buffer a and stores the result on buffer b. This subroutine is much faster than histogram equalization. If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively. The subroutine uses a 3x3 window. The destination buffer b must be different from the input buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: histeq

11.33 snr()

Subroutine for snr calculation in dB
LIBRARY MODE:

```
int snr(a,b,s,N1,M1,N2,M2)
```

```
image a,b;
float *s;
int N1,M1,N2,M2;

a,b: input and output buffers
s: L2 error ratio
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "SNR". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Filtering", "Metrics" and "SNR". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates SNR in dB between two images a,b:

$$s = \log \frac{\sum_{k=N_1}^{k=N_2} \sum_{l=M_1}^{l=M_2} (a[k][l] - b[k][l])^2}{\sum_{k=N_1}^{k=N_2} \sum_{l=M_1}^{l=M_2} a[k][l]^2}$$

It is used for the evaluation of image filtering and coding algorithms.
RETURN VALUE: 0, -1, -21
SEE ALSO: L2_error_ratio

11.34 wienerfrequency()

Subroutine to calculate the Wiener filter in the frequency domain
LIBRARY MODE:

```
int wienerfrequency(Fre,Fim,Pss,Hre,Him,Pw,N,M)
matrix Fre,Fim;
matrix Pss,Hre,Him;
float Pw;
int N,M;

fre: real part of the Filter (output)
fim: imaginary part of the Filter (output)
Pss: Power Spectral Density of the original image
Hre: real part of the Fourier Transform of the blur function
```

```
Him: imaginary part of the Fourier Transform of the blur function
Pw: Variance of the additive Gaussian noise on the blurred image
N,M: size of the matrices
```

SHORT DESCRIPTION: This subroutine calculates the Wiener filter in the frequency domain. The image formation model used is: $X = HS + W$. The filter output is given by: $Y = FX$, where F is the Wiener Filter:
$F = (H * Pss)/(|H|^2 Pss + Pw)$
RETURN VALUE: 0, -1, -21
SEE ALSO: convolution

11.35 zoom()

Subroutine to zoom an image
LIBRARY MODE:

```
int zoom(a,b,t,ND1,MD1,N1,M1,N2,M2);
image a,b;
int t;
int order;
int ND1,MD1;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
t: zoom factor
ND1,MD1: destination coordinates
N1,M1: upper left  corner of source buffer
N2,M2: lower right corner of source buffer
```

INTERPRETER MODE:

```
zoom a b t[ND1 MD1 N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: For a B/W image, open the "Black and White" menu, then consecutively select "Processing" and "Zoom". For a color image, open the "Color" menu, then consecutively select "Basic" and "Zoom". Fill-in the required fields, choose the interpolation mode and press "OK".

X–WINDOWS ENVIRONMENT: For a B/W image, open the "Black and White" menu, then consecutively select "Processing" and "Zoom". For a color image, open the "Color" menu, then consecutively select "Basic" and "Zoom". Fill-in the required fields, choose the interpolation mode and press "OK".
In the MS-Windows or X-Windows environments when the zoom factor is integer, routine zoom() is called. When zoom factor is float, routine interpolation() is called.
SHORT DESCRIPTION: This subroutine is used to zoom an image stored on buffer a and stores the result on buffer b. The zooming factor is t ($t >= 1$) and it is integer. If the resulting image exceeds the dimensions of buffer b, the subroutine returns an error value. Image shortening can be performed by subroutine decim.
RETURN VALUE: 0, -1, -21, -22, -90
SEE ALSO: decim

11.36 Test Programs

Subroutines from this chapter can be tested using the program below:

```
/* Program to test convolution routines                */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

/*---------------------------------------------------------------------*/
main(argc,argv)
int argc;
char *argv[];
{ int x,y,ret,choice,NN,MM;
  matrix mat1,mat2,mat3,mat4;
  image im1,im2,im3;

/* initialize EIKONA, maximum image and matrix size */
  NN=atoi(argv[1]);
  MM=atoi(argv[2]);
  init_eikona(NN,MM,15,15,256);

/* build two matrices of size NNxMM (of type float) */
  mat1=build_matrix(NN,MM);
  mat2=build_matrix(NN,MM);
  mat3=build_matrix(NN,MM);
  mat4=build_matrix(NN,MM);

/* build an image buffer of size NNxMM (of type unsigned char) */
  im1=build_image(NN,MM);
  im2=build_image(NN,MM);
  im3=build_image(NN,MM);

/* check whether memory allocation is O.K. */
  if (mat1==NULL || mat2==NULL || mat3==NULL || mat4==NULL
      || im1==NULL || im2==NULL)
     { printf("build : Unable to allocate memory\n"); exit(1); }

  clearmatrix(mat1,0,0,0,NN,MM);
  clear(im1,0,0,0,NN,MM);
```

```
  clear(im2,0,0,0,NN,MM);
/* load an image */
  ret=fromdisk(argv[4],im1,0,0,NN/2,MM/2);
     if(ret!=0) { printf("fromdisk : error %d\n",ret); exit(2); }

/* copy the image buffer im1 to the matrix buffer mat1 */
  image2matrix(im1,mat1,0,0,0,0,NN/2,MM/2);
/* clear the matrix mat2 with 0 */
  clearmatrix(mat2,0,0,0,NN,MM);

/* blur function : blur of size 3x3 */
  for(x=0; x<2; x++)
    mat2[0][x]=(float)(1.0/9.0);
  for(x=0; x<2; x++)
   for(y=0; y<2; y++)
    mat2[NN-1-y][x]=(float)(1.0/9.0);
  for(y=0; y<2; y++)
    mat2[y][NN-1]=(float)(1.0/9.0);
    mat2[0][MM-1]=(float)(1.0/9.0);

  choice=atoi(argv[3]);
  if(choice==1)
    {
   /* compute convolution by FFT*/
     ret=convolution(mat1,mat2,mat1,0,0,0,0,NN/2,MM/2);
     if(ret!=0) { printf("convolution : error %d\n",ret); exit(3); }

   /* copy the convolution stored on buffer mat1 to the image buffer im1 */
     matrix2image(mat1,im2,0,0,0,0,NN,MM);
     init_graphics("VRES16COLOR");
     imdisp(im1, 16,0,0,0,0,NN,MM);
     imdisp(im2, 16,0,MM,0,0,NN,MM);
     getch();
     reset_graphics();

   /* store the blurred image to a file */
     printf("OK, saving...");
     todisk(im2,argv[5],0,0,NN,MM);
    }
  if(choice==2)
    {
   /* compute convolution by direct computation*/
     ret=conv(im1,im2,mat2,3,3,0,0,NN/2,MM/2);
     if(ret!=0) { printf("convolution : error %d\n",ret); exit(3); }
```

```
 /* copy the convolution stored on buffer mat1 to the image buffer im1 */
   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();

 /* store the blurred image to a file */
   printf("OK, saving...");
   todisk(im2,argv[5],0,0,NN,MM);
  }
if(choice==3)
  {
 /* compute convolution by overlap-add*/
   ret=overlap_add(mat1,mat2,mat3,8,8,0,0,NN/2,MM/2,3,3);
   if(ret!=0) { printf("convolution : error %d\n",ret); exit(3); }

 /* copy the convolution stored on buffer mat1 to the image buffer im1 */
   matrix2image(mat3,im2,0,0,0,0,NN,MM);
   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();

 /* store the blurred image to a file */
   printf("OK, saving...");
   todisk(im2,argv[5],0,0,NN,MM);
  }
if(choice==4)
  {
 /* compute convolution by overlap-save*/
   ret=overlap_save(mat1,mat2,mat3,6,6,0,0,NN/2,MM/2,3,3);
   if(ret!=0) { printf("convolution : error %d\n",ret); exit(3); }

 /* copy the convolution stored on buffer mat1 to the image buffer im1 */
   matrix2image(mat3,im2,0,0,0,0,NN,MM);
   init_graphics("VRES16COLOR");
   imdisp(im1, 16,0,0,0,0,NN,MM);
   imdisp(im2, 16,0,MM,0,0,NN,MM);
   getch();
   reset_graphics();
```

```
   /* store the blurred image to a file */
     printf("OK, saving...");
     todisk(im2,argv[5],0,0,NN,MM);
    }
    return(0);
}
```

The syntax of the command line for this program is:

```
<Program Name> <Rows>x2 <Columns>x2 <Choice> <Input File> <Output File>
```

And depending on the value of `<Choice>` these are the possible outcomes:
1: Blurring of the input image through direct convolution (subroutine convolution()).
2: Blurring of the input image through filtering (subroutine conv()).
3: Blurring of the input image using overlap–add convolution.
4: Blurring of the input image using overlap–save convolution.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[PRA91] W.K.Pratt, *Digital image processing*, Wiley, 1991.

[JAI89] A.K.Jain, *Fundamentals of digital image processing*, Prentice Hall, 1989.

12

Nonlinear digital image filtering

12.1 adapt_l_filter()

Subroutine for adaptive L-filter
LIBRARY MODE:

```
adapt_l_filter(a,b, r, h, coe, NW,MW, N1,M1,N2,M2)
image a,b,r;
vector h;
float coe;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
r: reference image buffer
h: buffer of filter coefficients
coe: adaptation step size
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "Adapt. L-filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "Adapt. L-filter". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine calculates the adaptive LMS L-filter output and the filter coefficients after adaptation. The initial filter coefficients can be chosen arbitrarily. The use of the moving average or the median filter coefficients is recommended. The adaptation speed depends on the magnitude of the step size coe. Small step size leads to slow adaptation rate. Large step size may lead to instability.
REFERENCES: [PIT90], pp. 295-298
RETURN VALUE: 0, -1, -2, -10, -21, -24
SEE ALSO: a_trimmed_filt, l_filter

12.2 adapt_wei_median()

Subroutine for weighted 2-d median filtering
LIBRARY MODE:

```
adapt_wei_median(a,b,w,ct, NW,MW,N1,M1,N2,M2)
image a,b;
int w;
float ct;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
w: filter mid coefficient
ct: filter coefficient
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Adapt. Weighted Median". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Adapt. Weighted Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: Adaptive weighted median filter is designed to be used for speckle noise removal. The filter coefficients are controlled by the coefficient ct. If it is equal to zero all filter coefficients are equal to w. If it is greater than 0, the filter coefficients take triangular form along the two dimensions. The central

pixel has the largest weight. The slope of coefficient reduction depends on ct and on the ratio of local variance and local mean. If the ratio is small (homogeneous speckle image regions) the slope is small. If the ratio is large (close to edges) only the central pixel is heavily weighted and almost no filtering is performed. Thus we have adaptation both to edges and to the local speckle statistics.
REFERENCES:
T. Loupas, W.N. McDicken, P.L. Allan, "An adaptive weighted median filter for speckle suppression in medical ultrasonic imaging", IEEE Transactions on Circuits and Systems, vol. CAS-36, no. 1, pp. 129-135, Jan. 1989.
RETURN VALUE: 0, -1, -10, -21, -24, -84, -85, -633
SEE ALSO: rec_median, wei_median, sep_median, median

12.3 a_trimmed_filt()

Subroutine for 2-d alpha-trimmed mean filtering
LIBRARY MODE:

```
a_trimmed_filt(a,b, NE, NW,MW, N1,M1,N2,M2)
image a,b;
int NE,NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
NE: number of large pixel values and number of small
    pixel values to be trimmed (2*NE pixels in total)
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "a-trimmed filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "a-trimmed filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The alpha-trimmed mean filter trims the smallest NE and the largest NE pixels in the filter window and calculates the arithmetic mean of the rest of them. If NE is zero, the filter is equivalent to the moving average filter. If NE is equal to NW*MW/2 the filter is equivalent to the median filter.

REFERENCES: [PIT90], pp. 131-132.
RETURN VALUE: 0, -1, -10, -21, -24, -90
SEE ALSO: mnn, mtm, median, movav

12.4 bclose()

Subroutine for 2-d binary closing
LIBRARY MODE:

```
bclose(a,b, c, NW,MW, N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Close". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Close". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates binary closing by using a rectangular filter window. Since it is a two-step operation, it requires an image buffer for the storage of the intermediate results. The output image is evaluated in the region [N1+NW-1, N2-NW]x[M1+MW-1, M2-MW]. It uses the subroutines maxi(), mini().
REFERENCES: [PIT90], pp. 167-170
RETURN VALUE: 0, -1, -21, -24, return values of maxi(), mini()
SEE ALSO: bopen, imclose, imopen

12.5 bdilate()

Subroutine for 2-d 3x3 binary dilation
LIBRARY MODE:

```
bdilate(a,b, ki, N1,M1,N2,M2)
image a,b;
int ki;
int N1,M1,N2,M2;

a,b: input and output buffers
ki: parameter denoting the structuring element
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Dilate". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Dilate". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates binary dilation. The parameter ki selects structuring element. For ki=1 the structuring element is 3x3 square. For ki=2 the structuring element is a cross of size 3x3. For ki=3 the structuring element is a diagonal cross of size 3x3. For ki=0,45,90,135,180,225,270,315 the structuring element consists of only two pixels, namely the central pixel and the pixel that forms the direction described by ki. Dilation by larger and more complex structuring elements can be decomposed in a series of dilations using these fundamental windows. The dilation by a CIRCLE can be decomposed in two dilations by using ki=1 and ki=2 respectively.
REFERENCES: [PIT90], pp. 138-167
RETURN VALUE: 0, -1, -21, -24, -90
SEE ALSO: bclose, berode, bopen, imclose, imopen

12.6 berode()

Subroutine for 2-d 3x3 binary erosion
LIBRARY MODE:

```
berode(a,b, ki, N1,M1,N2,M2)
image a,b;
int ki;
int N1,M1,N2,M2;

a,b: input and output buffers
ki: parameter denoting the structuring element
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Erode". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Erode". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates binary erosion. The parameter ki selects structuring element. For ki=1 the structuring element is 3x3 square. For ki=2 the structuring element is a cross of size 3x3. For ki=3 the structuring element is a diagonal cross of size 3x3. For ki=0,45,90,135,180,225,270,315 the structuring element consists of only two pixels, namely the central pixel and the pixel that forms the direction described by ki. Erosion by larger and more complex structuring elements can be decomposed in a series of erosions using these fundamental windows. The erosion by a CIRCLE can be decomposed in two erosions by using k=1 and ki=2 respectively.
REFERENCES: [PIT90], pp. 138-167
RETURN VALUE: 0, -1, -21, -24, -90
SEE ALSO: bclose, bdilate, bopen, imclose, imopen

12.7 bopen()

Subroutine for 2-d binary opening
LIBRARY MODE:

```
bopen(a,b, c, NW,MW, N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;
```

```
a,b: input and output buffers
c: buffer for intermediate results
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Open". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Binary Open". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates binary opening by using a rectangular filter window. Since it is a two-step operation, it requires an image buffer for the storage of the intermediate results. The output image is evaluated in the region [N1+NW-1, N2-NW]x[M1+MW-1, M2-MW]. It uses the subroutines maxi(), mini().
REFERENCES: [PIT90], pp. 167-170
RETURN VALUE: 0, -1, -21, -24, return values of maxi(), mini()
SEE ALSO: bclose, imclose, imopen

12.8 dw_mtm()

Subroutine for 2-d double window modified trimmed mean filtering
LIBRARY MODE:

```
dw_mtm(a,b, q, NW,MW, NWL,MWL,N1,M1,N2,M2)
image a,b;
int q,NW,MW,NWL,MWL;
int N1,M1,N2,M2;

a,b: input and output buffers
q: threshold
NW,MW: small window size
NWL,MWL: large window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: The double window modified trimmed mean filter uses two filter windows. It calculates the median in the small window and then calculates the arithmetic mean of the pixels lying in the large window and having value in the range [-q,q] from the median.
REFERENCES: [PIT90], p.133
RETURN VALUE: 0, -1, -10, -21, -24, -82, -90
SEE ALSO: mnn, mtm

12.9 harm_filt()

Subroutine for 2-d harmonic filtering
LIBRARY MODE:

```
harm_filt(a,b, hcoe, NW,MW, N1,M1,N2,M2)
image a,b;
matrix hcoe;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
hcoe: filter coefficients
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Harmonic Filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Harmonic Filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The harmonic mean filter is a special case of homomorphic filter having the nonlinear point-wise function 1/x in the front-end of linear FIR filter. If a pixel value is equal to zero, it is transformed to 1 to prevent divide-by-zero error.
REFERENCES: [PIT90], p.230
RETURN VALUE: 0, -1, -2, -21, -24, -86
SEE ALSO: homom_filt, Lp_filt

12.10 homom_filt()

Subroutine for 2-d homomorphic filtering (exponential)
LIBRARY MODE:

```
homom_filt(a,b, hcoe, NW,MW, N1,M1,N2,M2)
image a,b;
matrix hcoe;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
hcoe: filter coefficients
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Homomorphic Filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Homomorphic Filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The logarithmic homomorphic filter is a special case of homomorphic filter having the nonlinear point-wise function ln(x) and exp(x) in the front-end of a linear FIR filter. If a pixel value is equal to zero, it is transformed to 1 to prevent evaluation of logarithm at zero (run-time error).
REFERENCES: [PIT90], pp. 218-220
RETURN VALUE: 0, -1, -2, -21, -24, -86
SEE ALSO: harm_filt, Lp_filt

12.11 imclose()

Subroutine for 2-d grayscale opening
LIBRARY MODE:

```
imclose(a,b,c, g, NW,MW, N1,M1,N2,M2)
image a,b,c;
image g;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
g: structuring function
NW,MW: window size
N1,M1: starting coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Close". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Close". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates grayscale closing. Since it is a two-step operation, it requires an image buffer for the storage of the intermediate results. The output image is evaluated in the region [N1+NW-1, N2-NW]x[M1+MW-1, M2-MW]. It uses the subroutines imdilate(), imerode().
REFERENCES: [PIT90], pp. 178-181
RETURN VALUE: 0, -1, -2, -21, -24, return values of imdilate(), imerode()
SEE ALSO:bopen, bclose, imopen

12.12 imdilate()

Subroutine for 2-d grayscale dilation
LIBRARY MODE:

```
int imdilate(a,b, g, NW,MW, N1,M1,N2,M2)
image a,b;
image g;
int NW,MW;
int N1,M1,N2,M2;

   a,b: input and output buffers
```

```
   g:   structuring function
   NW,MW: window size
   N1,M1: starting coordinates
   N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Dilate". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Dilate". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates grayscale dilation.
REFERENCES: [PIT90], pp. 178-181
RETURN VALUE: 0, -1, -2, -10, -21, -24
SEE ALSO:bopen, bclose, imopen

12.13 imerode()

Subroutine for 2-d grayscale erosion
LIBRARY MODE:

```
int imerode(a,b, g, NW,MW, N1,M1,N2,M2)
image a,b;
image g;
int NW,MW;
int N1,M1,N2,M2;

   a,b: input and output buffers
   g:   structuring function
   NW,MW: window size
   N1,M1: starting coordinates
   N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Erode". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Grayscale Erode". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates grayscale erosion.

REFERENCES: [PIT90], pp. 178-181
RETURN VALUE: 0,-1, -2, -10, -21, -24
SEE ALSO: bopen, bclose, imopen

12.14 imopen()

Subroutine for 2-d grayscale opening
LIBRARY MODE:

```
imopen(a,b,c, g, NW,MW, N1,M1,N2,M2)
image a,b,c;
image g;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
g: structuring function
NW,MW: window size
N1,M1: starting coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Local Adapt. filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Local Adapt. filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates grayscale opening. Since it is a two-step operation, it requires an image buffer for the storage of the intermediate results. The output image is evaluated in the region [N1+NW-1, N2-NW]x[M1+MW-1, M2-MW]. It uses the subroutines imdilate(), imerode().
REFERENCES: [PIT90], pp. 178-181
RETURN VALUE: 0, -1, -2, -21, -24, return values of imdilate(), imerode()
SEE ALSO:bopen, bclose, imclose

12.15 Lp_filt()

Subroutine for 2-d Lp filtering
LIBRARY MODE:

```
Lp_filt(a,b, p, hcoe, NW,MW, N1,M1,N2,M2)
image a,b;
float p;
matrix hcoe;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
p: filter parameter
hcoe: filter coefficients
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine calculates the Lp filter output, which is a special case of homomorphic filtering. p may take positive or negative values. When p is a large positive number, the filter output tends to the local maximum. When p is a large negative number, the filter output tends to the local minimum.
REFERENCES: [PIT90], pp. 230-234.
RETURN VALUE: 0, -1, -2, -21, -24, -86, -88
SEE ALSO: harm_filt, homom_filt

12.16 local_adapt_filt()

Subroutine for 2-d adaptive filtering based on local statistics
LIBRARY MODE:

```
local_adapt_filt(a,b, sn, NW,MW, N1,M1,N2,M2)
image a,b;
float sn;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
sn: noise variance
```

```
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Local Adapt. filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "Local Adapt. filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the local arithmetic mean and the local signal variance. If the signal variance is equal to noise variance sn (homogeneous image regions) the filter gives the arithmetic mean as output. If the signal variance is much larger than noise variance sn no filtering is performed at all. Thus noise reduction at homogeneous image regions is combined with edge preservation. The noise variance sn must be estimated in advance.
REFERENCES: [PIT90], p. 280.
RETURN VALUE: 0, -1, -21, -24, -81
SEE ALSO: mult_adapt_filt, sam

12.17 max_median()

Subroutine for 2-d max median filtering
LIBRARY MODE:

```
max_median(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Max Median". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Max Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates max/median filter output. The filter preserves edges. However, the filter output tends to be biased towards large filter outputs. It produces interesting video effects. The filter must have square window.
REFERENCES: [PIT90], pp.117-122.
RETURN VALUE: 0, -1, -10, -21, -24, -72
SEE ALSO: medhybr, multistage_median

12.18 medhybr()

Subroutine for 2-d median hybrid filtering (2LM+)
LIBRARY MODE:

```
medhybr(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Hybrid Median". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Hybrid Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The median hybrid filter combines noise reduction and edge and line preservation. It calculates linear FIR subfilter outputs along vertical, horizontal and diagonal directions and then calculates their median in two subsequent stages. The filter window must be square.
REFERENCES: [PIT90], pp. 122-127.
RETURN VALUE: 0, -1, -21, -24, -72
SEE ALSO: multistage_median

12.19 mnn()

Subroutine for 2-d modified nearest neighbor filtering
LIBRARY MODE:

```
mnn(a,b, q,NW,MW, N1,M1,N2,M2)
image a,b;
int q,NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
q: threshold
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine calculates the arithmetic mean of the pixels that lie in the filter window and in the range [-q,q] from the central pixel. Therefore it has good edge preservation properties if q is relatively small.
REFERENCES: [PIT90], p.133
RETURN VALUE: 0, -1, -21, -24, -90
SEE ALSO: mtm, dw_mtm

12.20 msd()

Subroutine for the morphological representation of binary images
LIBRARY MODE:

```
int msd(a,b,c,d, ki, N,N1,M1,N2,M2)
image a,b,c,d;
int ki,N;
int N1,M1,N2,M2;

   a,d: input and output buffers
   b,c: buffer for intermediate results
```

```
ki: parameter denoting the filter window
N: maximum number of components
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine computes the morphological decomposition of binary images. It decomposes a binary shape in simple primitive shapes (e.g. squares). The primitive shape depends on the parameter ki. For ki=1 the structuring element is 3x3 square. For ki=2 the structuring element is a cross of size 3x3. For ki=3 the structuring element is a diagonal cross of size 3x3. For ki=0,45,90,135,180,225,270,315 the structuring element consists of only two pixels, namely the central pixel and the pixel that forms the direction described by ki. It uses the subroutines bdilate(), berode(), or(), xor().
REFERENCES: [PIT90]
RETURN VALUE: 0, -1, -21, -24, -90, -100
SEE ALSO: bdilate, berode

12.21 mtm()

Subroutine for 2-d modified trimmed mean filtering
LIBRARY MODE:

```
mtm(a,b, q, NW,MW, N1,M1,N2,M2)
image a,b;
int q,NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
q: threshold
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This filter operates in two steps. In the first step calculates the arithmetic mean in the filter window. In the second step calculates the filter output which is the arithmetic mean of the pixels that lie in the filter window and in the range [-q,q] from the arithmetic mean found in the first step.
REFERENCES: [PIT90], p.133
RETURN VALUE: 0, -1, -21, -24, -90
SEE ALSO: dw_mtm, mnn

12.22 mult_adapt_filt()

Subroutine for 2-d adaptive filtering based on local statistics for multiplicative noise
LIBRARY MODE:

```
mult_adapt_filt(a,b, sn, NW,MW, N1,M1,N2,M2)
image a,b;
float sn;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
sn: noise variance
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine calculates the local arithmetic mean and the local signal variance. It uses them for filter adaptation in the case of multiplicative noise. Therefore, it can be used in speckle noise removal. It has been proven to have very good performance in the case of SAR image filtering.
REFERENCES:
J.S. Lee, "Speckle analysis and smoothing for of synthetic aperture radar images", Computer Graphics, Image Processing, vol. 17, pp. 24-32, 1981.
RETURN VALUE: 0, -1, -21, -24, -81
SEE ALSO: local_adapt_filt

12.23 multistage_median()

Subroutine for 2-d multistage median filtering
LIBRARY MODE:

```
multistage_median(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
```

```
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Multistage Median". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Multistage Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The multistage median filter calculates its output in two stages. In the first stage it calculates the medians along the horizontal, vertical and the two diagonal directions. In the second stage it forms of vector having as elements the four medians and the central pixel and calculates the median of this vector. It preserves well edges, lines and image details.
REFERENCES: [PIT90], p. 120
RETURN VALUE: 0, -1, -10, -21, -24, -72
SEE ALSO: max/median, medhybr

12.24 optimal_L_filter()

Subroutine for optimal MSE 2-d L-filter design
LIBRARY MODE:

```
optimal_L_filter(a,h,NW,MW,N1,M1,N2,M2)
image a;
vector h;
int NW,MW;
int N1,M1,N2,M2;

a,b: input image buffer
h: optimal L-filter coefficients
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "Optimal L-filter". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Order Statistics Filters" and "Optimal L-filter". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the optimal L-filter coefficients in the MSE sense. Its input is the noisy image a and its output is the filter coefficient vector h. The subroutine works well in the cases of images that have slowly varying content (large homogeneous regions) and are corrupted by additive noise. The filter coefficients obtained by this algorithm can be used by the l_filter routine to perform filtering.
REFERENCES: [PIT90], pp. 135-140
RETURN VALUE: 0, -1, -2, -10, -21, -24, -71, -72, -89
SEE ALSO: l_filter

12.25 rec_median()

Subroutine for recursive 2-d median filtering
LIBRARY MODE:

```
rec_median(a,b,NW,MW,N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine calculates the 2-d recursive median filter output in a row-wise manner. The previously calculated filter outputs within the filter window are used to calculate the new filter output. At the borders of the image no previously calculated pixels exist. In this case we use the original border image data instead. Recursive median filters tend to produce strongly correlated output and streaks.
REFERENCES: [PIT90], pp.108-111
RETURN VALUE: 0, -1, -10, -21, -24

SEE ALSO: median

12.26 run_max()

Subroutine for fast 2-d max filtering
LIBRARY MODE:

```
run_max(a,b,c,NW,MW,N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine performs fast 2-d max filtering for filter window NWxMW. It exploits the running nature of 2-d max filter and therefore it is much faster than the subroutine maxi, especially for large filter windows. Its output is identical to that of the subroutine maxi.
REFERENCES: [PIT90], p. 358
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: maxi, run_median, run_min

12.27 run_median()

Subroutine for fast 2-d median filtering
LIBRARY MODE:

```
run_median(a,b,NW,MW,N1,M1,N2,M2)
image a,b;
int NW,MW;
```

```
int N1,M1,N2,M2;

a,b: input and output buffers
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine performs fast 2-d median filtering for filter window NWxMW. It exploits the running nature of 2-d median filter (Huang's algorithm) and therefore it is much faster than the subroutine median. Its output is identical to that of the subroutine median.
REFERENCES: [PIT90], p. 346
RETURN VALUE: 0, -1, -10, -21, -24,
SEE ALSO: run_max, run_min, median

12.28 run_min()

Subroutine for fast 2-d min filtering
LIBRARY MODE:

```
run_min(a,b,c,NW,MW,N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine performs fast 2-d min filtering for filter window NWxMW. It exploits the running nature of 2-d min filter and therefore it is much faster than the subroutine mini, especially for large filter windows. Its output is identical to that of the subroutine mini.
REFERENCES: [PIT90], p. 358
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: mini, run_median, run_max

12.29 sam()

Subroutine for modified signal adaptive median filtering
LIBRARY MODE:

```
int sam(a,b,sn,ca,cb,cc,bt,smin,smax,
        pn,pp,NW,MW,NWMA,MWMA,NWMI,MWMI,N1,M1,N2,M2)
image a,b;
float sn,ca,cb,cc,bt,pp,pn;
int smax,smin;
int NW,MW,NWMA,MWMA,NWMI,MWMI;
int N1,M1,N2,M2;

a,b: input and output image buffers
sn: noise variance
ca,cb,cc,bt: filter parameters
smax,smin: maximum and minimum signal values
pp,pn:percentage of positive (smax) and negative (smin) constant impulses
NW,MW: initial filter window dimensions         (3 x 3)
BNWMA,MWMA: maximal filter window dimensions   (11 x 11)
NWMI,MWMI: minimal filter window dimensions     (3 x 3)
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS-WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "S.A.M. filter". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering" and "S.A.M. filter". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: SAM filter consists of a median filter whose size is varying. It becomes large in homogeneous image regions and small close to edges. The filter suppresses high-pass image components in homogeneous image regions and allows them pass close to edges. It removes impulses. It has very good performance as it has been demonstrated in several simulations.

REFERENCES: [PIT90], pp. 290-293

RETURN VALUE: 0, -1, -10, -21, -24, -72, -80, -81, -89, -90

SEE ALSO: local_adapt_filter

12.30 sep_median()

Subroutine for separable 2-d median filtering
LIBRARY MODE:

```
sep_median(a,b,c, NW,MW, N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
c: buffer for intermediate results
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Separable Median". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Separable Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs separable median filtering along horizontal and vertical directions sequentially. It is generally faster than the 2-d standard median filter and has similar properties in noise filtering and edge preservation. However, standard median filtering has superior noise filtering properties.
REFERENCES: [PIT90], pp. 105-108
RETURN VALUE: 0, -1, -10, -21, -24
SEE ALSO: median

12.31 skeleton()

Subroutine for morphological skeletonization
LIBRARY MODE:

```
skeleton(a,b,c,d, ki, N1,M1,N2,M2)
image a,b,c,d;
int ki;
int N1,M1,N2,M2;
```

```
a,d: input and output buffers
b,c: buffers for intermediate results
ki: parameter denoting the structuring element
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Skeleton". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Skeleton". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the morphological skeleton of a binary image. The parameter ki selects structuring element used in skeletonization. For ki=1 the structuring element is 3x3 square. For ki=2 the structuring element is a cross of size 3x3. For ki=3 the structuring element is a diagonal cross of size 3x3. For ki=0,45,90,135,180,225,270,315 the structuring element consists of only two pixels, namely the central pixel and the pixel that forms the direction described by ki. The method used for the calculation of the morphological skeleton is described in [PIT90], p. 191. It uses the subroutines berode() and xor(). The input image must be binary. The resulting binary output image is the skeleton. No information is produced about the skeleton function [PIT90], p.189. Therefore the output produced cannot be used for object reconstruction. It can be used for object recognition purposes.
REFERENCES: [PIT90], pp. 188-192.
RETURN VALUE: 0, -1, -21, -24, -90, -100
SEE ALSO: bdilate, berode

12.32 top_hat()

indextop_hat()
Subroutine for top-hat transformation
LIBRARY MODE:

```
top_hat(a,b,c, NW,MW, N1,M1,N2,M2)
image a,b,c;
int NW,MW;
int N1,M1,N2,M2;
```

```
a,b: input and output buffers
c: intermediate buffer
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Top Hat". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Morphology" and "Top Hat". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: Top-hat transformation is the difference of an image minus its opening by a structuring element B. It is a high pass filter that eliminates background and preserves lines.It uses the subroutine bopen().
REFERENCES: [PIT90], p. 187
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: bopen

12.33 wei_median()

Subroutine for weighted 2-d median filtering
LIBRARY MODE:

```
wei_median(a,b,g, NW,MW,N1,M1,N2,M2)
image a,b;
image g;
int NW,MW;
int N1,M1,N2,M2;

a,b: input and output buffers
g: filter coefficients
NW,MW: window size
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Weighted Median". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Non-linear filtering", "Median Filters" and "Weighted Median". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: The weighted median filter is a special case of median filter. Its linear counterpart is the FIR filter having coefficient set g. Its coefficients are positive integers. The sum of its weights must not be greater then W*LNWMAX*LMWMAX. W is chosen to be equal to 3 in this implementation. It can become adaptive.
REFERENCES: [PIT90], pp. 111-112
RETURN VALUE: 0, -1, -2, -10, -21, -24, -84
SEE ALSO: adapt_wei_filt

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

13

Digital image coding

13.1 AddToTable()

Subroutine that adds entries to table at the position
LIBRARY MODE:

```
int AddToTable(T,pos,c,Tlength,Plength,tablepointer,
               codesize,maxcode,Nc,llinks)
int far*T,pos,c,far*Tlength,Plength,*tablepointer,
    *codesize,*maxcode,Nc;
LPOINTER *llinks;

T : code table
pos : position of string to be written to T
c : ASCII code of character to be appended to S(pos)
tablepointer : position of last entry to T
codesize : current length (in bytes) of code that is read
maxcode : 2^(codesize+1)-1
Nc : number of code table symbols
llinks : array of the "children" of each code table string
```

SHORT DESCRIPTION: This subroutine that adds S(pos)+c to table at the position indicated by tablepointer. It returns -340 if ClearCode is missing from input file while decoding or must be outputed while encoding.
RETURN VALUE: 0, -10, -340
SEE ALSO: lzw

13.2 AddEntry()

Subroutine that add entries to tablepointer
LIBRARY MODE:

```
int AddEntry(T,pos,c,tablepointer,codesize,maxcode,Nc)
int far* far* T,pos,c,*tablepointer,*codesize,*maxcode,Nc;

T : code table
pos : position of string to be written to T
c : ASCII code of character to be appended to S(pos)
tablepointer : position of last entry to T
codesize : current length (in bytes) of code that is read
maxcode : 2^(codesize+1)-1
Nc : number of code table symbols
```

SHORT DESCRIPTION: This subroutine adds S(pos)+c to tablepointer. It is used in invlzw. It returns -340 if ClearCode is missing from input file.
RETURN VALUE: 0, , -10, -340
SEE ALSO: invlzw

13.3 arcoefficients()

Subroutine for 2-d AR model Coefficients
LIBRARY MODE:

```
int arcoefficients(ima,N,M,K,LL,LR,A,Pu)
image ima;
int N,M,K,LL,LR;
float A[];
float *Pu;

ima: input image buffer
N,M: size of the image (end coordinates)
K,LL,LR: size of the support region of the AR model
```

```
A: AR model Coefficients (output)
Pu: variance of the driving white noise proccess (output)
```

SHORT DESCRIPTION: This subroutine calculates the 2-d AR model coefficients. The model can be non-symmetric half-plane (NSHP) or quarter plane. It uses the subroutine corsample() to calculate image correlations and subroutine arsolve to solve the resulting system of equations.
RETURN VALUE: 0, -1, -2, -10, -21, -90
SEE ALSO: arsolve, corsample, arpsd

13.4 arsolve()

Subroutine to solve a system of AR equations
LIBRARY MODE:

```
arsolve(sa,sb,ss,s1,sn)
float far* far* sa;
float *sb;
float *ss;
int s1,sn;

sa: square matrix of size (sn x sn)
sb,ss: vectors of size (sn)
s1: (s1,s1) is the upper left corner in the matrix sa from where the
contex of the matrix will be taken into account
```

SHORT DESCRIPTION: This subroutine solves the linear system of equations to calculate the 2d AR model coefficients. Internal use only, called by arcoefficients().
RETURN VALUE: 0
SEE ALSO: arcoefficients

13.5 corsample()

Subroutine that calculates the correlation between two images

LIBRARY MODE:

```
float corsample(im1,im2,N,M,k,l)
image im1;
image im2;
int N,M;
int k,l;
```

SHORT DESCRIPTION: This subroutine calculates the correlation between two images im1 and im2 of size (NxM) at lag (k,l). Called by arcoefficients().
RETURN VALUE: 0
SEE ALSO: arcoefficients, correlation

13.6 decode_huffman()

Subroutine to perform Huffman decoding
LIBRARY MODE:

```
int decode_huffman(fr,im,root,leaf,N,M)
FILE *fr;
image im;
SPOINTER root;
struct symbol far*leaf;
int N,M;

fr: input file containing the compressed image
im: output image
root: root of decoding tree
leaf: decoding tree leaves
N,M: image size
```

SHORT DESCRIPTION: This subroutine performs Huffman decoding. The coded data are read from file fr by using buffer. 4096 bytes are read each time. If coded data are not enough, it returns unsuccessfully. The coded data are used to traverse the decoding tree and to produce the pixel value that is stored on the output image. It returns -320 if a decoding error occurs.
RETURN VALUE: 0, -320
SEE ALSO: hufcod

13.7 find2min()

Subroutine that finds the two elements pointed to by 'nodes' that have the minimal probability.
LIBRARY MODE:

```
int find2min(nodes,min,NUMOFSYMBOLS)
SPOINTER *nodes;
int min[2];
int NUMOFSYMBOLS;

nodes: array of elements of type SPOINTER.
min: output table containing elements pointing the position of two
     elements of nodes having minimal probability.
NUMOFSYMBOLS : number of image symbols
```

SHORT DESCRIPTION: This subroutine that finds the two elements pointed to by 'nodes' that have the minimal probability. It is used in huftree() for the construction of the Huffman tree. It returns -310 if an error occurs in the construction of decoding tree.
RETURN VALUE: 0, -310
SEE ALSO: huftree

13.8 GetNextCode()

Subroutine that reads bits from input buffer.
LIBRARY MODE:

```
int GetNextCode(code,codesize,fr,buf,bufsize,
                BUFMAXSIZE,bufpointer,bytepointer,imask,cmask)
int *code,codesize;
FILE *fr;
unsigned char *buf;
int *bufsize,BUFMAXSIZE,*bufpointer,*bytepointer;
unsigned int *imask;
unsigned char *cmask;

code : integer buffer (2 bytes) where next code will be stored
codesize : current length (in bits) of code to be read
fr : file that contains compressed image
```

```
buf : integer buffer (BUFMAXSIZE*2 bytes)
bufpointer : byte in buf that is currently read
bytepointer : bit in bufpointer byte that is currently read
imask : unsigned integer mask
cmask : unsigned character mask
```

SHORT DESCRIPTION: This subroutine reads codesize bits from input buffer and puts them in code. It is used in invlzw. It returns -330 input file too small.
RETURN VALUE: 0, -330
SEE ALSO: invlzw

13.9 hufcod()

Subroutine that encodes a grayscale image by using Huffman coding.
LIBRARY MODE:

```
int hufcod(fc,im,fw,N,M,NUMOFSYMBOLS)
FILE *fc;
image im;
FILE *fw;
int N,M, NUMOFSYMBOLS;

fc: file containing the codewords of the symbols
im : image buffer
fw: compressed file
N,M: image dimensions
NUMOFSYMBOLS: number of gray levels
```

SHORT DESCRIPTION: Subroutine that encodes a grayscale image by using Huffman coding. It reads first the codetable. Then it reads the image pixels one by one and sends the corresponding codes to the output file. Both files fc, fw must be open before this subroutine is called.
RETURN VALUE: 0, -10, -21
SEE ALSO: sendToOutput, decode_huffman

13.10 huftree()

Subroutine for constructing a Huffman decoding tree
LIBRARY MODE:

```
int huftree(pdf,rpnr,lpnr,NUMOFSYMBOLS)
vector pdf;
SPOINTER *rpnr;
struct symbol far**lpnr;
int NUMOFSYMBOLS;

pdf : vector of grayscale probabilities
rpnr: pointer to root of decoding tree
lpnr: pointer to array of decoding tree leaves
NUMOFSYMBOLS : number of image symbols
```

SHORT DESCRIPTION: This subroutine constructs a Huffman decoding tree. It returns -310 if an error occurs in decoding tree construction. It returns -320 if an error occurs in decoding.
RETURN VALUE: 0, -10, -310, -320
SEE ALSO: decode_huffman

13.11 initialize()

Subroutine for initialization of "leaf" and "nodes" elements
LIBRARY MODE:

```
int initialize(nodes,lpnr,pdf,NUMOFSYMBOLS)
SPOINTER *nodes;
struct symbol far**lpnr;
vector pdf;
int NUMOFSYMBOLS;

nodes: array of elements of type SPOINTER. One of them
       will point to the Huffman tree root after tree construction.
lpnr: pointer to array of elements of type symbol that correspond
      to Huffman tree leaves.
pdf: grayscale pdf
NUMOFSYMBOLS : number of image symbols
```

SHORT DESCRIPTION: Subroutine for initialization of "leaf" and "nodes" elements. It is used in huftree() for the construction of the Huffman tree.
RETURN VALUE: 0
SEE ALSO: huftree

13.12 init_lzw()

Subroutine that initializes codetable and current prefix
LIBRARY MODE:

```
init_lzw(Ns,Nc,tablepointer,T,Tlength,Plength,maxcode,codesize,llinks)
int Ns,Nc,*tablepointer,far*T,far*Tlength,*Plength,*maxcode,*codesize;
LPOINTER *llinks;

Ns : number of image symbols
Nc : number of code table symbols
tablepointer : position of last entry to code table
T : code table
Tlength : table of code table string length
Plength : length of current prefix P
maxcode : maximum code that can be represented with codesize bits
codesize : current length (in bits) of the code that is read
llinks : array of the "children" of each code table string
```

SHORT DESCRIPTION: This subroutine initializes codetable and current prefix. It is used in lzw().
RETURN VALUE: 0
SEE ALSO:

13.13 inT()

Subroutine that searches code table
LIBRARY MODE:

```
int inT(T,Tlength,Plength,symbol,old,llinks)
```

```
int far* T,far* Tlength,Plength;
unsigned char symbol;
int old;
LPOINTER *llinks;
T: code table
P: current prefix
symbol: input symbol
old: position of current prefix in codetable
llinks : array of the "children" of each code table string
```

SHORT DESCRIPTION: This subroutine searches the code table to check if it includes current prefix appended with "symbol". It returns the position (0) of curprefix+symbol in codetable. It returns -310 if curprefix+symbol does not exist in codetable.
RETURN VALUE: 0, -310, positive integers
SEE ALSO: lzw

13.14 invlzw()

Subroutine that decodes an LZW-encoded image
LIBRARY MODE:

```
int invlzw(fr,im,Ns,Nc,N,M)
FILE *fr;
image im;
int Ns,Nc,N,M;

fr : input file
im : decoded image buffer
Ns : number of image symbols
Nc : number of code symbols
N,M: image size
```

SHORT DESCRIPTION: This subroutine decodes an LZW-encoded image from file fr and stores it on buffer im. It uses init_invlzw for initializations. It returns -330 if the input file does not contain enough information for decoding. It returns -340 if ClearCode missing from input file.
RETURN VALUE: 0, -10, -330, -340
SEE ALSO: lzw

13.15 init_invlzw()

Subroutine that initializes codetable and current prefix
LIBRARY MODE:

```
init_invlzw(Ns,Nc,tablepointer,T,maxcode,codesize)
int Ns,Nc,*tablepointer,far*far*T,*maxcode,*codesize;

Ns : number of image symbols
Nc : number of code table symbols
tablepointer : position of last entry to code table
T : code table
maxcode : maximum code that can be represented with codesize bits
codesize : current length (in bits) of the code that is read
```

SHORT DESCRIPTION: This subroutine initializes codetable and current prefix. It is used in invlzw.
RETURN VALUE: 0, -10
SEE ALSO: invlzw

13.16 lzw()

Subroutine for the coding of an image according to LZW algorithm
LIBRARY MODE:

```
int lzw(im,fw,Ns,Nc,N,M)
image im;
FILE *fw;
int Ns,Nc,N,M;

im : image to be encoded
fw: output file
Ns : number of image symbols (usually 256)
Nc : number of code symbols  (usually 4096)
N,M: image size
```

SHORT DESCRIPTION: This subroutine codes of an image im according to LZW algorithm. It stores the output on a file fw. Initializations are performed by init_lzw(). Output to fw is performed by send2output().
RETURN VALUE: 0, -10
SEE ALSO: init_lzw(), send2output, invlzw

13.17 merge()

Subroutine to merge two elements of nodes.
LIBRARY MODE:

```
SPOINTER merge(nodes,min)
SPOINTER *nodes;
int min[2];

nodes: array of elements of type SPOINTER.
min: output table containing elements pointing the position of two
     elements of nodes having minimal probability.
```

SHORT DESCRIPTION: Subroutine to merge two elements of nodes. It is used in huftree() for the construction of the Huffman tree.
RETURN VALUE: 0
SEE ALSO: huftree

13.18 reconstr_tree()

Subroutine used for the reconstruction of the Huffman tree from the code table.
LIBRARY MODE:

```
int reconstr_tree(fc,lpnr,rpnr,NUMOFSYMBOLS)
FILE *fc;
struct symbol far**lpnr;
SPOINTER *rpnr;
int NUMOFSYMBOLS;
```

```
fc: file that contains the code table
lpnr: pointer to array of tree leaves
rpnr: pointer to root of the tree
NUMOFSYMBOLS: number of grayscale levels in image (symbols)
```

SHORT DESCRIPTION: Subroutine used for the reconstruction of the Huffman tree from the code table. It is used during Huffman decoding.
RETURN VALUE: 0, -10
SEE ALSO: decode_huffman

13.19 send2output()

Subroutine that sends packed bits to output file.
LIBRARY MODE:

```
int send2output(fw,code,csize,packpointer,pack,cmask,pmask)
FILE *fw;
int code,csize;
int *packpointer;
unsigned char *pack;
unsigned int *cmask;
unsigned char *pmask;

fw: output file
code: input integer showing the codetable position
      that must be sent to the output file.
csize: number of most significant bits of code that must be sent
      to the output file.
packpointer: number of least significant bits of pack
      that contain useful information
pack: output buffer (1 byte)
cmask: code mask
pmask: pack mask
```

SHORT DESCRIPTION: This subroutine sends packed bits to output file.
RETURN VALUE: 0
SEE ALSO: lzw

13.20 sendToOutput()

Subroutine that sents packed bits to output file.
LIBRARY MODE:

```
int sendToOutput(fw,code,packpointer,pack)
FILE *fw;
char far* code;
int *packpointer;
unsigned char *pack;

fw: output file name
code: codeword to be sent to output
packpointer: number of bits in buffer pack that contain
coded information.
pack: 8-bit (1 byte) buffer
```

SHORT DESCRIPTION: Subroutine that sends packed bits to output file. It is used in hufcod() for Huffman coding.
RETURN VALUE: 0
SEE ALSO: hufcod

13.21 treeaccess()

Subroutine to perform preorder access of Huffman tree
LIBRARY MODE:

```
int treeaccess(root,stackpointer,stack,leaf,code)
SPOINTER root;
int *stackpointer;
char *stack;
struct symbol far*leaf;
unsigned char far* far * code;

root: tree root (input)
stackpointer: number of bits in stack
stack: stack of bits
leaf: array of tree leaves
code: table of strings containing codewords (output)
```

SHORT DESCRIPTION: This subroutine performs preorder access of Huffman tree. It is used in Huffman coding.
RETURN VALUE: 0, -1
SEE ALSO: hufcod

13.22 WriteString()

Subroutine that sends a string to the output image buffer
LIBRARY MODE:

```
int WriteString(T,old,i,j,N,M,im)
int far* far* T,old,*i,*j,N,M;
image im;

T : code table
old : position of string to be written to image buffer
i,j : position of current pixel
N,M : image sizes
im : image buffer
```

SHORT DESCRIPTION: This subroutine sends string S(old) to output (image buffer). It is used in invlzw.
RETURN VALUE: 0
SEE ALSO: invlzw

13.23 Test Programs

Subroutines for Huffman coding from this chapter can be tested using the program bellow:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
```

```
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc, char *argv[]) {
 image im;
 int NUMOFSYMBOLS=256,i,j,choice;
 FILE *fr,*fc,*fw;
 vector pdf;
 SPOINTER root;
 struct symbol far*leaf;
 int stackpointer;
 unsigned char far* far* code,* str;
 char * stack;

 choice=atoi(argv[1]);
 switch(choice)
 {
 case 1: /*Huffman table construction*/
  if(argc!=4) {
    printf("\nSyntax : huffman 1 [image file] [code file]");
    exit(1);
  }
  init_eikona(256,256,0,0,256);
  if((pdf=build_vector(256))==NULL) {
    printf("\nMemory allocation error -pdf-");
    exit(-1);
  }
  im=build_image(256,256);
  fromdisk(argv[2],im,0,0,NMAX,MMAX);
  hist(im,pdf,0,0,NMAX,MMAX);

 /* allocate space for codetable */
 if((code=matuc2(NUMOFSYMBOLS,NUMOFSYMBOLS))==NULL) {
   printf("\nAllocation error");
   exit(-1);
 }

/* Construct decoding tree */
 i=huftree(pdf,&root,&leaf,NUMOFSYMBOLS);
 printf("\nFunction HUFTREE returned %d",i);
 if(i) exit(i);
```

```
/* initialize "stack" and "stackpointer".    */
/* Access the tree and visit all its leaves in */
/* order to determine the code for each symbol. */
stack[0]=0;
stackpointer=0;
i=treeaccess(root,&stackpointer,stack,leaf,code);
printf("\nFunction TREEACCESS returned %d",i);
if(i) exit(i);

/* output results */
if((fw=fopen(argv[3],"wt"))==NULL) {
  printf("\nCannot open write file");
  exit(-2);
}
if((str=(unsigned char *)malloc(NUMOFSYMBOLS*sizeof(unsigned char)))==NULL) {
  printf("\nMemory allocation error - str -");
  exit(-10);
}
for(i=0;i<NUMOFSYMBOLS;i++) {
  j=-1;
  do{
    j++;
    str[j]=code[i][j];
  } while(code[i][j]);
  fprintf(fw,"\n%s",str);
}
fclose(fw);

break;


case 2: /*coding*/

if(argc!=5) {
  printf("\nSyntax : huffman 2 [image file] [code file]");
  printf(" [encoded image file]");
  exit(1);
}
init_eikona(256,256,0,0,256);
im=build_image(256,256);
fromdisk(argv[2],im,0,0,NMAX,MMAX);
if((fc=fopen(argv[3],"rt"))==NULL) {
  printf("\nCannot read code file ");
```

```
  exit(-2);
}
if((fw=fopen(argv[4],"wb"))==NULL) {
  printf("\nCannot write to output file");
  exit(-2);
}

i=hufcod(fc,im,fw,NMAX,MMAX,NUMOFSYMBOLS);
printf("\nFunction HUFCOD returned %d",i);
fclose(fc);
fclose(fw);
break;

case 3: /*decoding*/

init_eikona(256,256,0,0,256);
im=build_image(256,256);
if(argc!=5) {
  printf("\nSyntax : huffman 3 [code file] [encoded image file]");
  printf(" [decoded image file]");
  exit(1);
}

if((fc=fopen(argv[2],"rt"))==NULL) {
  printf("\nCannot read code file ");
  exit(-2);
}

/* Construct decoding tree */
i=reconstr_tree(fc,&leaf,&root,NUMOFSYMBOLS);
printf("\nFunction RECONSTR_TREE returned %d",i);
if(i) exit(i);
fclose(fc);

/* open compressed image file */
if((fr=fopen(argv[3],"rb"))==NULL) {
  printf("\nCannot open compressed image file");
  exit(-2);
}

/* Perform decoding */
i=decode_huffman(fr,im,root,leaf,NMAX,MMAX);
fclose(fr);
printf("\nFunction DECODE_HUFFMAN returned %d",i);
```

```
 if(i) exit(i);

 init_graphics("VRES16COLOR");
 imdisp(im,16,0,0,0,0,NMAX,MMAX);
 getch();

 reset_graphics();

 /* sent decoded image to "decompfile" */
 todisk(im,argv[4],0,0,NMAX,MMAX);
 break;
 }
 return(0);
}
```

The syntax of the command line is self-explanatory by the program, after giving:

```
<Program Name> <Choice>
```

Depending on the value of `<Choice>` these are the possible outcomes:
1: Huffman coding table construction.
2: Huffman Coding.
3: Huffman Decoding.

Subroutines for LZW coding from this chapter can be tested using the program bellow:

```
#include <stdio.h>
#include <stdlib.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc,char *argv[])
{
 int Ns=256,Nc=4096,i,choice;
 image im;
 FILE *fr,*fw;

 if(argc!=4) {
   printf("\nSyntax : lzw [choice] [input file] [output file]");
   exit(1);
 }
```

```
choice = atoi(argv[1]);

switch(choice) {
  case 1:           /* coding */

    /* allocate space for image */
    init_eikona(256,256,0,0,256);
    im=build_image(256,256);

    fromdisk(argv[2],im,0,0,NMAX,MMAX);

    fclose(fr);

    if((fw=fopen(argv[3],"wb"))==NULL) {
      printf("\nError while opening write-file");
      exit(-2);
    }
    i=lzw(im,fw,Ns,Nc,NMAX,MMAX);
    printf("\nFunction LZW returned %d",i);
    if(i) exit(i);
    fclose(fw);

    break;

  case 2:           /* decoding */

    /* open image file */
    if((fr=fopen(argv[2],"rb"))==NULL) {
      printf("\nCannot open image file");
      exit(1);
    }

    /* allocate space for image */
    init_eikona(256,256,0,0,256);
    im=build_image(256,256);

    i=invlzw(fr,im,Ns,Nc,NMAX,MMAX);
    printf("\nFunction INVLZW returned %d",i);

    todisk(im,argv[3],0,0,NMAX,MMAX);

    break;
}
```

```
 return(0);
}
```

The program works for images of size 256×256 and the syntax of the command line is self-explanatory by the program, after giving:

```
<Program Name>
```

Depending on the value of `<Choice>` these are the possible outcomes:
1: LZW Coding.
2: LZW Decoding.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

14

Edge detection routines

14.1 compass()

Subroutine to detect edges at different directions
LIBRARY MODE:

```
int compass(a,b, th, N1,M1,N2,M2)
image a,b;
int th;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
th: edge direction (0, 45, 90, 135)
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
compass a b th [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Compass". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Compass". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine is used to detect edges of an image stored on buffer a along different directions. The result is stored in buffer b. Directions in 0, 45, 90 and 135 degrees are supported. The destination buffer b must be different from the source buffer a. When an output pixel value exceeds 255, it is truncated to 255. It is also truncated to 0, if its value is less than 0.
RETURN VALUE: 0, -1, -21, -70
SEE ALSO: range, sobel, prewitt, laplace, line_detect

14.2 edge_follow()

Subroutine to perform edge following starting from pixel nr,nc.
LIBRARY MODE:

```
int edge_follow(im_in,im_out,nr,nc,T,T1,N1,M1,N2,M2)
image im_in,im_out;
int nr,nc,T,T1,N1,M1,N2,M2;

im_in: input image
im_out: output image
nr: row of input pixel
nc: column of input pixel
T: threshold for edge pixel value
T1: threshold of difference
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Edge following". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Edge following". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs edge following starting from pixel (nr,nc). Search is based on the intensity difference of neighboring edge elements. Search for edge element successor is performed on the entire 8-point neighborhood of the current edge pixel.Search starts from the NW neighbor and continues search in row-wise manner, until it finds the first valid successor. The successor must not be equal to the current pixel predecessor. The procedure continues recursively. Subroutine stops if no valid successor has been found. The

edge to be followed must have pixel values greater than T. The value difference of successive pixels must be less than threshold T1.
RETURN VALUE: 0,-1,-21,-90,-632,return values of copy.
SEE ALSO: edge_dynamic_prog, modified_heuristic_edge_search

14.3 edge_dynamic_prog()

Subroutine for edge following based on a dynamic-programming method.
LIBRARY MODE:

```
int edge_dynamic_prog(xin_or,xA,xB,T,outstack)
image xin_or;
CPIXEL xA,xB;
int T;
NODE far** outstack;

xin_or: input image
xA: starting pixel
xB: goal pixel
T: threshold for edge pixel value.
outstack: pointer to the start pointer of output stack
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Dynamic progr. Edge foll.". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Dynamic progr. Edge foll.". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs edge following based on a dynamic-programming method. It produces a path from starting pixel xA to goal pixel xB. All edge pixels must have value greater than T. Two stacks are used. Stack outstack contains the pixels of the path (the winner of every step of the algorithm). Stack tempstack contains the probable winners of every step (in the 8-neighborhood). It finds possible edge element successors and finds the possible edge successor that is in the direction defined by previous and current edge pixel. If this is a valid direction and the corresponding edge pixel is a valid candidate, we choose this pixel as the current winner. Otherwise, sort the successors according their cost. We put possible successors in stack tempstack. We choose as most possible successor the one with the maximal cost. If such successor does

not exist, we backtrace by using pixels from tempstack. If tempstack is empty, -98 is returned. If end pixel (goal) has been reached, return.
RETURN VALUE: 0,-1,-2,-90,-91,-98,-99,-632, return values of copy(),push().
SEE ALSO: stack2image, edge_follow, modified_heuristic_edge_search

---

14.4 hough()

Subroutine to perform Hough transform.
LIBRARY MODE:

```
int hough(xin,p,n,m,N,M,COS,SIN)
int N,M,m,n;
image xin;
imatrix p;
float *COS,*SIN;

xin: input image
p: parameter matrix  in the r,theta space
N: number of rows of image
M: number of columns of image
n: number of rows of matrix p
m: number of columns of matrix p
COS: look-up table for cos function
SIN: look-up table for sin function
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Hough". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Hough". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs Hough transform on the input image xin. The result is a two dimensional matrix p in the (r,theta) parameter space. For any pixel having value 1 in the input image the corresponding elements of matrix p are incremented, according to (r, theta) Hough transform. Theta ranges from -pi/2 to pi/2. R ranges from -sqrt(N*N+M*M) to sqrt(N*N+M*M). The R and theta buffer sizes determine the resolution of the parameter space. For example, if the user specifies 100 as R buffer size and 180 as theta buffer size, then the (R, theta) parameter space

would have 2*sqrt(N*N+M*M)/100 cell resolution in the R axis, and 1 degree cell resolution in the theta axis. The lower the cell resolution, the smaller the number of lines detected by the algorithm.

RETURN VALUE: 0, -1, -2, -21, -90, -100

SEE ALSO: ihough, thres_par

14.5 ihough()

Subroutine to produce the inverse Hough transform

LIBRARY MODE:

```
int ihough(xout,p,N,M,n,m,COS,SIN)
image xout;
imatrix p;
int N,M,n,m;
float *COS,*SIN;

xout: output image
p: parameter matrix  in the r,theta space
N: number of rows of image
M: number of columns of image
n: number of rows of matrix p
m: number of columns of matrix p
COS: look-up table for cos function
SIN: look-up table for sin function
```

SHORT DESCRIPTION: This subroutine produces the inverse Hough transform from the (r,theta) space to the (x,y) space. Parameter matrix p has already been thresholded. For every element of p having value 1, the corresponding straight line (consisting of 1s) is drawn in the output matrix xout.

RETURN VALUE: 0, -1, -2, -21, -90, -100

SEE ALSO: hough, thres_par

14.6 laplace()

Subroutine for Laplace edge detector
LIBRARY MODE:

```
int laplace(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
laplace a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Laplace". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Laplace". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates Laplace edge detector output of an image stored on buffer a and stores the result on buffer b. The value 128 is added to the result. Therefore, zero transition is equivalent to 128 transition. If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively. The Laplace edge detector uses a 3x3 window for its operation. The destination buffer b must be different form the source buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: prewitt, sobel, range, roberts

14.7 look_up_table()

Subroutine to produce look-up tables for COS and SIN functions.
LIBRARY MODE:

```
int look_up_table(COS,SIN,m)
```

```
int m;
float *COS,*SIN;

COS: look-up table for cos function.
SIN: look-up table for sin function.
m:   number of rows for COS, SIN look-up tables
```

SHORT DESCRIPTION: Subroutine to produce look-up tables for COS and SIN functions (to be used in Hough transform and its inverse).
RETURN VALUE: 0
SEE ALSO: hough, ihough

14.8 line_detect()

Subroutine to detect lines
LIBRARY MODE:

```
int line_detect(a,b, th, N1,M1,N2,M2)
image a,b;
int th;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
th: line direction (0, 45, 90, 135)
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
line_detect a b th [N1 N2 M1 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Line detect". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Line detect". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine finds the lines along a certain direction th of an image stored on buffer a and stores the result on buffer b. Allowable

directions are 0, 45, 90 and 135 degrees. The subroutine uses a 3x3 window to produce its output. If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21, -70
SEE ALSO: compass, point_detect, sobel, prewitt, roberts, range, laplace

---

14.9 modified_heuristic_edge_search()

Subroutine for edge following based on heuristic search.
LIBRARY MODE:

```
int modified_heuristic_edge_search(xin_or,xA,xB,T,outstack)
image xin_or;
CPIXEL xA,xB;
int T;
NODE far** outstack;

xin_or: input image
xA: starting pixel
xB: goal pixel
T: threshold for edge pixels values in the path
outstack: pointer to the start pointer of output stack
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Heurist. Edge following". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Heurist. Edge following". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs edge following based on heuristic search. It produces a path from starting pixel xA to goal pixel xB. Two stacks are used. Stack outstack contains the pixels of the path (the winner of every step of the algorithm). Stack tempstack contains the probable winners of every step (in the 8-neighborhood). All edge pixel values must be greater than T. It finds possible edge element successors and finds the possible edge successor that is in the direction defined by previous and current edge pixel. If this is a valid direction and the corresponding edge pixel is a valid candidate, we choose this pixel as the current winner. Otherwise, sort the successors according their

cost. We put possible successors in stack tempstack. We choose as most possible successor the one with the minimal local cost. If such successor does not exist, we backtrace by using pixels from tempstack. If tempstack is empty, -98 is returned. If end pixel (goal) has been reached, return. RETURN VALUE:
0,-1,-2,-90,-91,-98,-99,-632,copy,push.
SEE ALSO: edge_follow, edge_dynamic_prog

14.10 point_detect()

Subroutine to detect isolated points
LIBRARY MODE:

```
int point_detect(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
point_detect a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Point detect". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Point detect". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine detects isolated points on an image stored on buffer a and stores the result on buffer b. The subroutine uses a standard 3x3 window. If an output pixel value lies outside the range [0..255] it is truncated to 0 or 255 respectively. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: line_detect, sobel, laplace, prewitt, roberts, range

14.11 prewitt()

Subroutine for Prewitt edge detector
LIBRARY MODE:

```
int prewitt(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
prewitt a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Prewitt". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Prewitt". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates Prewitt edge detector output of an image stored on buffer a and stores the result on buffer b. The Prewitt edge detector uses a 3x3 window for its operation. If an output pixel value lies outside the range [0..255] it is truncated to 0 or 255 respectively. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: laplace, sobel, range, roberts

14.12 range()

Subroutine for 2-d range edge detection
LIBRARY MODE:

```
int range(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
NW,MW: window size
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
range a b NW MW [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Range". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Range". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the range edge detector output of an image stored on buffer a and stores the result on buffer b. The range edge detector is simple the difference of the local minimum from the local maximum in a window of size NWxMW. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21, -24
SEE ALSO: laplace, sobel, prewitt, roberts

14.13 roberts()

Subroutine for Roberts egde detector
LIBRARY MODE:

```
int roberts(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
```

```
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
roberts a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Roberts". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Roberts". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates Roberts edge detector output of an image stored on buffer a and stores the result on buffer b. If an output pixel value lies outside the range [0..255] it is truncated to 0 or 255 respectively. The Prewitt edge detector uses a 2x2 window for its operation. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: laplace, prewitt, sobel, range

14.14 stack2image()

Subroutine to create an image using the contents of outstack .
LIBRARY MODE:

```
int stack2image(outstack,xout)
image xout;
NODE far** outstack;

outstack: pointer to the start pointer of output stack
xout: output image
```

SHORT DESCRIPTION: Subroutine to write the contents of stack in the output image. It puts 1 to output image whereever there is 1 in outstack.
RETURN VALUE: 0,-1,-98.
SEE ALSO: edge_dynamic_prog, modified_heuristic_edge_search

14.15 sobel()

Subroutine for Sobel edge detection
LIBRARY MODE:

```
int sobel(a,b, N1,M1,N2,M2)
image a,b;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
sobel a b [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Sobel". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Edge Detection" and "Sobel". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates Sobel edge detector output of an image stored on buffer a and stores the result on buffer b. If an output pixel value lies outside the range [0..255], it is truncated to 0 or 255 respectively. The Sobel edge detector uses a 3x3 window for its operation. The destination buffer b must be different from the source buffer a.
RETURN VALUE: 0, -1, -21
SEE ALSO: laplace, prewitt, roberts, range

14.16 thres_par()

Subroutine to threshold arrays of type imatrix.
LIBRARY MODE:

```
int thres_par(p,n,m,t)
imatrix p;
```

```
int n,m,t;

p: parameter matrix  in the r,theta space
n: number of rows of matrix p
m: number of columns of matrix p
t: threshold
```

SHORT DESCRIPTION: This subroutine thresholds arrays of type imatrix. It is used in line detection by Hough transform. Each element of array p which is above the given threshold t declares a line.
RETURN VALUE: 0, -1, -90
SEE ALSO: hough, ihough

14.17 Test Programs

Subroutines from this chapter can be tested using the program below:

```
/*-------------------------------------------------------------------------*/
/*                              EDGETEST.C                                 */
/*-------------------------------------------------------------------------*/
#include <stdio.h>
#include <malloc.h>
#include <float.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

/*-------------------------------------------------------------------------*/
main(argc,argv)
int argc;
char *argv[];

{  int i,j,t,choice,r;
   char imfile[60];
   image xin,xout;
   imatrix p;

   float *COS,*SIN;
   int KA,LA,t1,A;
   int ii,jj,d,h;
   CPIXEL *index,last,xA,xB;
   NODE far** outstack=NULL;
   int NOS=9;

   if(argc<3){
   printf("You didn't give NMAX and MMAX\n");
   printf("Start again\n");
   exit(0);
}
   NMAX=atoi(argv[1]);
   MMAX=atoi(argv[2]);

   init_eikona(NMAX,MMAX,-1,-1,-1);
```

```
   xin=build_image(NMAX,MMAX);
   xout=build_image(NMAX,MMAX);
   if(!xin || !xout ) {
      printf(" There isn't enough memory \n");
      exit(1);
}
   printf("Input  image >"); scanf("%s" ,imfile);
   d=fromdisk(imfile,xin,0,0,NMAX,MMAX);
   printf("fromdisk = %d\n",d);
   printf("Output image >"); scanf("%s" ,imfile);

   printf("1: Hough\n");
   printf("2: Edgefollow\n");
   printf("3: Dyn.prog. \n");
   printf("4: Heuristic Search\n");
   scanf("%d" ,&choice);

   switch(choice) {
    case 1: {
    p=build_imatrix(NMAX,MMAX);
    COS=(float *)malloc(NMAX*sizeof(float));
    SIN=(float *)malloc(NMAX*sizeof(float));
    if(!COS || !SIN || !p) {
      printf(" There isn't enough memory for COS and SIN and p\n");
      exit(1);
     }
    printf("threshold t ="); scanf("%d", &t);
    r=thres(xin,xin,t,0,0,NMAX,MMAX);
    printf("thres = %d\n",r);
    r=look_up_table(COS,SIN,NMAX);
    printf("look_up_table =%d\n", r);
    printf("hough transform begins\n");
    r=hough(xin,p,NMAX,MMAX,NMAX,MMAX,COS,SIN);
    printf("hough ended\n");
    printf("hough = %d\n",r);
    printf("thres_par t ="); scanf("%d", &t);
    r=thres_par(p,NMAX,MMAX,t);
    printf("thres_par=%d\n",r);
    printf("ihough begins\n");
    r=ihough(xin,xout,p,NMAX,MMAX,NMAX,MMAX,COS,SIN);
    init_graphics("VRES16COLOR");
    bindisp(xout,16,16,0,0,0,NMAX,MMAX);
    getch();
    reset_graphics();
```

```
    printf("ihough = %d\n",r);
    r=todisk(xout,imfile,0,0,NMAX,MMAX);
    printf("todisk = %d\n",r);
}
    break;

   case 2: {
    printf("Threshold: t ="); scanf("%d" , &t);
    printf("Startpixel.row: KA ="); scanf("%d" ,&KA);
    printf("Startpixel.column: LA ="); scanf("%d" ,&LA);
    while(xin[KA][LA]<t) {
    printf("Startpixel.row: KA ="); scanf("%d" ,&KA);
    printf("Startpixel.column: LA ="); scanf("%d" ,&LA);
    }
    printf("Threshold of diff. t1 ="); scanf("%d",&t1);
    printf("Factor: A ="); scanf("%d" ,&A);
    r=clear(xout,0,0,0,NMAX,MMAX);
    printf("clear = %d\n",r);
    printf("I will enter in edge_follow\n");
    r=edge_follow(xout,xin,KA,LA,t1,t,A);
    printf("edge_follow = %d\n",r);
    init_graphics("VRES16COLOR");
    bindisp(xout,16,16,0,0,0,NMAX,MMAX);
    getch();
    reset_graphics();
    printf("I will make image\n");
    r=todisk(xout,imfile,0,0,NMAX,MMAX);
    printf("todisk = %d\n",r);
}
    break;

    case 3: {
    index=(CPIXEL *)malloc(NOS*sizeof(CPIXEL));
    printf("Threshold = "); scanf("%d" ,&t);
    printf("StartPIXEL.row ="); scanf("%d" ,&i);
    printf("StartPIXEL.column ="); scanf("%d" ,&j); printf("%d\n",xin[i][j]);
    while (xin[i][j]<t) {
    printf("StartPIXEL.row ="); scanf("%d" ,&i);
    printf("StartPIXEL.column ="); scanf("%d" ,&j); printf("%d\n",xin[i][j]);
    }
    printf("GoalPIXEL.row ="); scanf("%d" ,&ii);
    printf("GoalPIXEL.column ="); scanf("%d" ,&jj); printf("%d\n",xin[ii][jj]);while
    printf("GoalPIXEL.row ="); scanf("%d" ,&ii);
    printf("GoalPIXEL.column ="); scanf("%d" ,&jj); printf("%d\n",xin[ii][jj]);
```

```
    }
    r=initialize_index(index);
    printf("initialize_index = %d\n",r);
    xA.row = i;
    xA.column = j;
    xA.cost = xin[i][j];
    xB.row =ii;
    xB.column = jj;
    xB.cost = xin[ii][jj];
    d=edge_dynamic_prog(xA,xB,xin,t,index,last,outstack);
    printf("edge_dynamic_prog = %d\n",d);
    if(*outstack==NULL) printf("l NULL in main\n");
    if(d==0) printf("I have found Goalpixel\n");
    printf("I will make image\n");
    r=stack2image(outstack,xout);
    init_graphics("VRES16COLOR");
    bindisp(xout,16,16,0,0,0,NMAX,MMAX);
    getch();
    reset_graphics();
    printf("stack2image = %d \n",r);
    r=todisk(xout,imfile,0,0,NMAX,MMAX);
    printf("todisk = %d \n",r);
}
    break;

    case 4: {
    index=(CPIXEL *)malloc(NOS*sizeof(CPIXEL));
    printf("Threshold = "); scanf("%d" ,&t);
    printf("StartPIXEL.row ="); scanf("%d" ,&i);
    printf("StartPIXEL.column ="); scanf("%d" ,&j);printf("%d\n",xin[i][j]);
    while (xin[i][j]<t) {
    printf("StartPIXEL.row ="); scanf("%d" ,&i);
    printf("StartPIXEL.column ="); scanf("%d" ,&j);printf("%d\n",xin[i][j]);
    }
    printf("GoalPIXEL.row ="); scanf("%d" ,&ii);
    printf("GoalPIXEL.column ="); scanf("%d" ,&jj); printf("%d\n",xin[ii][jj]);
    while(xin[ii][jj]<t) {
    printf("GoalPIXEL.row ="); scanf("%d" ,&ii);
    printf("GoalPIXEL.column ="); scanf("%d" ,&jj);printf("%d\n",xin[ii][jj]);
    }
    r=initialize_index(index);
    printf("initialize_index = %d\n",r);
    xA.row = i;
    xA.column = j;
```

```
    xA.cost = xin[i][j];
    xB.row =ii;
    xB.column = jj;
    xB.cost = xin[ii][jj];
    h=modified_heuristic_edge_search(xA,xB,xin,t,index,last,outstack);
    printf("modified_heuristic_edge_search = %d\n",h);
    if(outstack==NULL) printf("l NULL in main\n");
    if(h==0) printf("I have found Goalpixel\n");
    printf("I will make image\n");
    r=stack2image(outstack,xout);
    init_graphics("VRES16COLOR");
    bindisp(xout,16,16,0,0,0,NMAX,MMAX);
    getch();
    reset_graphics();
    printf("stack2image = %d \n",r);
    r=todisk(xout,imfile,0,0,NMAX,MMAX);
    printf("todisk = %d \n",r);
}
    break;

    default :
     break;
  }
return(0);
}
/*-------------------------------------------------------------------------*/
```

The syntax of the command line for this program is:

```
<Program Name> <Rows> <Columns>
```

Then on, the program guides the user through 4 different choices:
1: Edge detection by using Hough Transform.
2: Edge following.
3: Edge following based on dynamic programming method
4: Edge following based on heuristic search.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

15

Region segmentation and texture analysis

15.1 count()

Subroutine for counting objects in an image
LIBRARY MODE:

```
int count(a, N, h, N1,M1,N2,M2)
image a;
int *N
vector h;
int N1,M1,N2,M2;

a: image buffer
*N: number of objects
h: buffer to store object areas
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
count a h [N1 M1 N2 M2]
a: integer
h: integer (0 or 1)
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Count items". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Count items". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to count objects on a binary image stored on buffer a and to measure their area in pixels. It counts up to NSIG objects. If the image is not binary, the subroutine returns an error value. The results of area counting are stored on signal buffer h. In interpreter mode, the results are typed on the screen. The image buffer a is cleared after counting.
RETURN VALUE: 0, -1, -2, -21, -100
SEE ALSO: grass_fire, dump_signal, bin, findch

15.2 grass_label()

Subroutine for counting objects in a binary image
LIBRARY MODE:

```
int grass_label(a,b,N,h,NS,N1,M1,N2,M2)
image a, b;
int *N;
vector h;
int NS;
int N1,M1,N2,M2;

a: input image buffer
b: output labeled image buffer
*N: number of objects
h: buffer to store object areas
NS: maximal allowable number of objects (NS must be less than 255!)
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine counts objects in a binary image. Its operation is performed through subroutine grass() which clears a binary object and calculates its area. It operates only on binary images.
RETURN VALUE: 0, -1, -2, -21, -100
SEE ALSO: count, grass_fire, grass

15.3 grass()

Subroutine to clear a binary object and calculate its area
LIBRARY MODE:

```
int grass(a,b,row,col,number,area,N1,M1,N2,M2)
image a,b;
int row,col,number;
float far* area;
int N1,M1,N2,M2;

a: input image buffer
b: labeled image buffer
row,col: pixel coordinates
number: current object number
*area: object area (in pixels)
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine clears a binary object that starts at (row, col) and calculates its area. It is a recursive routine. Therefore, if the object to be counted has a very large area, the subroutine may create memory problems during run-time (especially under DOS).
RETURN VALUE: 0
SEE ALSO: grass_fire, grass_label

15.4 grass_fire()

Subroutine to clear a binary object and calculate its area
LIBRARY MODE:

```
grass_fire(a,row,col,area,N1,M1,N2,M2)
image a;
int row,col;
float far* area;
int N1,M1,N2,M2;

a: image buffer
row,col: pixel coordinates
*area: object area (in pixels)
```

```
N1,M1: start coordinates
N2,M2: end   coordinates
```

SHORT DESCRIPTION: This subroutine sets a "grass fire" to clear a binary object starting at pixel (row,col). At the same time it calculate its area. It is used in the subroutine count. The fire propagates along vertical and horizontal directions.
RETURN VALUE: 0, -1, -21, -90
SEE ALSO: count

15.5 mode_filter()

Subroutine to perform mode filtering
LIBRARY MODE:

```
int mode_filter(a,b, NW,MW, N1,M1,N2,M2)
image a,b;
int NW,MW;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
NW,MW: filter window
N1,M1: upper left  corner coordinates
N2,M2: lower rignt corner coordinates
```

INTERPRETER MODE:

```
mode_filter a b NW MW [N1 M1 N2 M2]
a,b: integers
```

SHORT DESCRIPTION: This subroutine implements the majority filter of an image stored on buffer a and stores the result on buffer b. An output pixel value is the intensity that is the majority in the local filter window of the input image. The window size is NWxNW. The destination buffer b must be different from the source buffer a. In the case of binary or two-valued images, this filter is equivalent to the median filter.
RETURN VALUE: 0, -1, -10, -21, -24
SEE ALSO: median

15.6 psRtheta()

Subroutine to calculate angular and radial distribution of power spectrum.
LIBRARY MODE:

```
int psRtheta(ps,psr,pstheta,N,M,NS)
matrix ps;
vector psr,pstheta;
int N,M,NS;

ps: magnitude of power spectrum.
psr: vector containing radial ps distribution.
     Its domain is [0,..,NMAX/2-1].
pstheta: vector containing angular ps distribution.
         Its domain corresponds to [-pi,..,pi].
N,M: size of power spectrum buffer
NS: size of vectors psr, pstheta
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Angular-Radial". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Angular-Radial". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates angular and radial distribution of power spectrum. The power spectrum can be calculated by any of the methods described in this manual, i.e. by periodogram, Blackman-Tukey or 2d AR modeling. We rearrange the input matrix so that dc coefficient (0,0) is at position (N/2,M/2). Rearrangement is performed in place. We initialize psr, pstheta vectors. We scan the entire image and calculate psr, pstheta by taking into account only power spectrum points lying at maximal distance N/2-1 from central point (N/2, M/2). The subroutine operates only on square matrices.
RETURN VALUE: 0, -1, -2, -34, -90
SEE ALSO: fft2d, reim2magnphase, blackman_tukey_psd, arpsd

15.7 region_grow()

Subroutine for region growing.
LIBRARY MODE:

```
int region_grow(a,b,N1,M1,N2,M2,N,T)
image a;
image b;
int N1,N2,M1,M2,N,T;

a: input image buffer
b: output image buffer (initially it contains seed points )
N1,M1: start coordinates
N2,M2: end   coordinates
N: number of regions
T: threshold
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Grow". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Grow". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs region growing. Seed pixels labeled by their region number [1,..,N] are initially contained in array b. The seed pixels are grown at each pass of the image (iteration). For any current border point of each region, the subroutine checks if there is any pixel at its neiborhood that can be merged to the region where the border belongs to. The candidate pixel must not be assigned to a region and its intensity must be close to the mean value of the region under consideration (their difference must be less than threshold T). After each pass of the entire image, it counts the pixels that have not been assigned to any region. If this number did not change in this iteration, it stops. At the end, it scans the output image and label the assigned pixels by the mean value of the corresponding region.
RETURN VALUE: 0,-1,-10,-21,-90,-92,-97.
SEE ALSO: segment, region_split, region_merge, region_split_merge

---

15.8 region_merge()

Subroutine for region segmentation by merging.
LIBRARY MODE:

```
int region_merge(a,b,N1,M1,N2,M2,T,REGMAX)
image a;
```

```
image b;
int N1,N2,M1,M2,T,REGMAX;

a: input image buffer
b: output image buffer
N1,M1: start coordinates
N2,M2: end   coordinates
T: threshold
REGMAX: maximum number of regions
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Merge". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Merge". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs region segmentation by merging. It creates the first region for the pixel at (1,1) and continues for the rest of the image pixels. For each pixel, we examine its NSHP 4-neighborhood and we test if this pixel can be merged with any region in this neighborhood. Merging is allowed if the pixel intensity is close to region mean value. If more than one merges are possible, the region with the closest mean value is chosen. If merge is not allowed with the immediately neighboring regions, we try to merge it with any other possible region. If no merge is allowed and the max number of region is not exceeded, we create a new region. At the end, we scan output image and label the assigned pixels by the mean value of the corresponding region.
RETURN VALUE: 0,-1,-10-21,-90,number of regions created.
SEE ALSO: segment, region_split, region_grow, region_split_merge

---

15.9 region_split()

Subroutine for region splitting.
LIBRARY MODE:

```
int region_split(a,b,N1,M1,N2,M2,N,T)
image a;
image b;
int N1,N2,M1,M2,T;
unsigned long *N;
```

```
a: input image buffer
b: output image buffer
N1,M1: start coordinates
N2,M2: end   coordinates
N: number of regions (originally 0);
T: threshold
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Split". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Split". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs region splitting. We test homogeneity of an image region. If it is inhomogeneous, we split it in four regions. If the region is homogeneous, calculate its mean and update output image buffer. It uses the subroutine test_homogeneity() for homogeneity tests.
RETURN VALUE: 0, -1, -4, -21, -90
SEE ALSO: test_homogeneity, segment, region_merge, region_grow, region_split_merge

---

15.10 region_split_merge()

Subroutine for split and merge algorithm.
LIBRARY MODE:

```
int region_split_merge(a,b,MARRAY,N1,M1,N2,M2,N,T,REGMAX)
image a;
image b;
unsigned char far* MARRAY;
int N1,N2,M1,M2,T,REGMAX;
int *N;

a: input image buffer
b: output image buffer
   MARRAY: vector containing the mean values of each region.
N1,M1: start coordinates
N2,M2: end   coordinates
```

```
N: number of regions (originally 0);
T: threshold
REGMAX: maximal allowable number of image regions
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Spit merge". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Region Spit merge". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine implements the split and merge algorithm. At each iteration, we test homogeneity of each image region. If the region is inhomogeneous and its size is greater than 1 pixel, we split it in four regions. If the region is homogeneous, we calculate its mean value.We try to merge adjacent regions by employing subroutine rmerge(). It returns -92 if we exceed the allowable number of regions REGMAX.
RETURN VALUE: 0, -1, -21, -92
SEE ALSO: segment, region_merge, region_grow, region_split, region_split_merge_c

---

15.11 rmerge()

Subroutine for merging adjacent regions.
LIBRARY MODE:

```
int rmerge(a,b,MARRAY,Nmin,Mmin,Nmax,Mmax,N1,M1,N2,M2,N,T)
image a;
image b;
unsigned char far* MARRAY;
int Nmin,Mmin,Nmax,Mmax,N1,N2,M1,M2;
int *N;
int T;

a: input image buffer
b: output image buffer
MARRAY: vector containing the mean values of each region.
Nmin,Mmin: start coordinates of global ROI
Nmax,Mmax: end   coordinates of global ROI
N1,M1: start coordinates
```

```
N2,M2: end   coordinates
N: number of regions
T: threshold
```

SHORT DESCRIPTION: This subroutine performs merging adjacent regions. We find the region having the closest mean, along the upper, lower, left and right boundary of current region and we perform merging. We do not update the mean of the merged region.
RETURN VALUE: 0
SEE ALSO: region_split_merge

15.12 region_split_merge_c()

Subroutine for split and merge algorithm with region mean updating.
LIBRARY MODE:

```
int region_split_merge_c(a,b,MARRAY,LABEL,N1,M1,N2,M2,N,T,REGMAX)
image a;
image b;
unsigned char far* MARRAY;
int far* far* LABEL;
int N1,N2,M1,M2,T,REGMAX;
int *N;
```

```
a: input image buffer
b: output image buffer
MARRAY: vector containing the mean values of each region.
LABEL:  int array containing region labels.
N1,M1: start coordinates
N2,M2: end   coordinates
N: number of regions (originally 0);
T: threshold
REGMAX: maximal allowable number of image regions
```

SHORT DESCRIPTION: This subroutine implements the split and merge algorithm. At each iteration, we test homogeneity of each image region. If the region is inhomogeneous and its size is greater than 1 pixel, we split it in four regions. If the region is homogeneous, we calculate its mean value. We try to merge adjacent regions by employing subroutine rmerge_c(). This routine

updates the region intensity means during merging. It returns -92 if we exceed the allowable number of regions REGMAX.
RETURN VALUE: 0, -1, -21, -92
SEE ALSO: segment, region_merge, region_grow, region_split, region_split_merge

---

15.13 rmerge_c()

Subroutine for merging adjacent regions.
LIBRARY MODE:

```
int rmerge_c(a,b,MARRAY,LABEL,Nmin,Mimn,Nmax,Mmax,N1,M1,N2,M2,N,T)
image a;
image b;
unsigned char far* MARRAY;
int far* far* LABEL;
int N1,N2,M1,M2;
int *N;
int T;

a: input image buffer
b: output image buffer
MARRAY: vector containing the mean values of each region.
LABEL:  int array containing the region labels.
Nmin,Mmin: start coordinates of global ROI
Nmax,Mmax: end   coordinates of global ROI
N1,M1: start coordinates
N2,M2: end   coordinates
N: number of regions
T: threshold
```

SHORT DESCRIPTION: This subroutine performs merging adjacent regions. We find the region having the closest mean, along the upper, lower, left and right boundary of current region and we perform merging. We update the mean of the merged region.
RETURN VALUE: 0
SEE ALSO:

15.14 segm()

Subroutine to segment an image in nr regions
LIBRARY MODE:

```
int segm(a,b, nr, N1,M1,N2,M2)
image a,b;
int nr;
int N1,M1,N2,M2;

a: source image buffer
b: destination image buffer
nr: number of regions
N1,M1: upper left  corner coordinates
N2,M2: lower right corner coordinates
```

INTERPRETER MODE:

```
segm a b nr [N1 M1 N2 M2]
a,b: integers
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Segment". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Region Segmentation" and "Segment". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine is used to segment a grayscale image uniformly in nr different grayscale regions. The grayscale range of each region is 256/nr. The resulting image is grayscale. If an image is to be segmented in two regions (nr=2), the subroutine thres can be used instead. However, in this case, the resulting image is binary.
RETURN VALUE: 0, -1, -21, -90
SEE ALSO: thres, region_split, region_merge, region_split_merge

---

15.15 test_homogeneity()

Subroutine to test region homogeneity.
LIBRARY MODE:

```
int test_homogeneity(a,N1,M1,N2,M2,T)
image a;
int N1,N2,M1,M2,T;

a: input image buffer
N1,M1: start coordinates
N2,M2: end   coordinates
T: threshold
```

SHORT DESCRIPTION: This subroutine tests region homogeneity. A region is considered to be homogeneous if its range (maximal pixel value minus minimal pixel value) is less than threshold T. It returns 1 for homogeneous regions and 0 for inhomogeneous regions.
RETURN VALUE: 0, 1
SEE ALSO: region_split

15.16 Subroutines for histogram parameters

LIBRARY MODE:

```
float meanvalu(hv)
vector hv;
Subroutine to calculate histogram mean value
hv: histogram buffer

float varianc(hv)
vector hv;
Subroutine to calculate histogram variance
hv: histogram buffer

float skewnes(hv)
vector hv;
Subroutine to calculate histogram skewness. If variance equals 0,
then skewness equals -1.0.
hv: histogram buffer

float kurtosi(hv)
vector hv;
```

```
Subroutine to calculate histogram kurtosis. If variance equals 0,
then kurtosis equals -1.0.
hv: histogram buffer

float entrop(hv)
vector hv;
Subroutine to calculate histogram entropy.
hv: histogram buffer.
Every element of hv must be greater than $10^{-5}$.
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Hist. Parameters". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Histogram" and "Hist. Parameters". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This set of subroutines calculates the following parameters of the image histogram: mean, variance, skewness, kurtosis and entropy. All return floating point numbers.
RETURN VALUE: float numbers, -1 (if hv == NULL or divide by zero occurs).
SEE ALSO: hist

15.17 difhist()

Subroutine to calculate the histogram of gray level differences.
LIBRARY MODE:

```
int difhist(a,hv,dx,dy,N1,M1,N2,M2)
image a;
vector hv;
int dx,dy;
int N1,M1,N2,M2;
a: input image buffer
hv: output histogram buffer
dx,dy: displacement vector
N1,M1: start coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Gray Level Dif. Histogram".

Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Gray Level Dif. Histogram". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the histogram of gray level differences. We calculate the valid search area. and perform histogram calculation of gray level differences within this area.
RETURN VALUE: 0, -1, -2, -10, -21, -90
SEE ALSO: Subroutines for parameters of the histogram of gray-level differences

---

15.18 Subroutines for parameters of the histogram of gray-level differences

LIBRARY MODE:

```
float c_d(hv)
vector hv;
Subroutine to calculate contrast of histogram of gray-level differences
hv: histogram buffer

float acs(hv)
vector hv;
Subroutine to calculate the angular second moment
of histogram of gray-level differences
hv: histogram buffer

float entropy_d(hv)
vector hv;
Subroutine to calculate the entropy of histogram of gray-level differences
hv: histogram buffer

float m_d(hv)
vector hv;
Subroutine to calculate the mean value
of histogram of gray-level differences
hv: histogram buffer
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Gray Lev. Dif. Hist. Parameters". Fill-in the required fields and press "OK".

X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Gray Lev. Dif. Hist. Parameters". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: These subroutines calculate the following parameters of the histogram of gray-level differences: contrast, angular second moment, entropy, mean value. All return floating point numbers.
RETURN VALUE: float numbers
SEE ALSO: diffhist

15.19 runlen()

Subroutine for the calculation of the run length matrix
LIBRARY MODE:

```
int runlen(a,rc,theta,N1,M1,N2,M2)
image a;
imatrix rc;
int theta;
int N1,M1,N2,M2;

a: input image buffer
rc: output run length matrix (integers)
theta: run orientation  (0, 45, 90, 135 degrees)
N1,M1: start coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Run-Length". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Run-Length". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs the calculation of the run length matrix along the directions 0, 45, 90, 135 degrees. It returns -90 if the orientation theta has different value for the above-mentioned ones. In the case of 45 and 135 degrees we split the rectangular ROI in two triangles (along either rectangle diagonal) and we perform run length calculation in each triangle independently.
RETURN VALUE: 0, -1, -21, -90

SEE ALSO: Subroutines for parameters of the run length matrix

---

15.20 Subroutines for parameters of the run length matrix

LIBRARY MODE:

```
double T_R(rc)
imatrix rc;
Subroutine to calculate the double sum of run length matrix elements.
rc: run length matrix.

double sre(rc)
imatrix rc;
Subroutine to calculate short run emphasis.
rc: run length matrix.
If T_R equals zero, sre = -1.0.

double lre(rc)
imatrix rc;
Subroutine to calculate long run emphasis.
rc: run length matrix.
If T_R equals zero, lre = -1.0.

double gld(rc)
imatrix rc;
Subroutine to calculate gray-level distribution.
rc: run length matrix.
If T_R equals zero, gld = -1.0.

double rld(rc)
imatrix rc;
Subroutine to calculate run length distribution.
rc: run length matrix.
If T_R equals zero, rld = -1.0.

double rp(rc)
imatrix rc;
Subroutine to calculate run percentages.
```

```
rc: run length matrix.
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Run-Length Parameters". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Run-Length Parameters". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: These subroutines calculate the following parameters of the run length matrix: double sum of run length matrix elements, short run emphasis, long run emphasis, gray-level distribution, run length distribution, run percentages. All subroutines return double precision numbers.
RETURN VALUE: double precision numbers, -1 (if rc = NULL or divide by zero occurs).
SEE ALSO: runlen

15.21 glcmarr()

Subroutine to calculate gray level co-occurrence matrices.
LIBRARY MODE:

```
int glcmarr(a,pdf,dx,dy,N1,M1,N2,M2)
image a;
matrix pdf;
int dx,dy;
int N1,M1,N2,M2;

a: input image buffer
pdf: output GLCM
dx,dy: displacement vector
N1,M1: start coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Co-occurence". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Co-occurence". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine calculates gray level co-occurrence matrices.
RETURN VALUE: 0, -1, -21, -89
SEE ALSO: Subroutines for parameters of the co-occurrence matrix

---

15.22 Subroutines for parameters of the co-occurrence matrix

LIBRARY MODE:

```
float c_glcm(pdf)
matrix pdf;
Subroutine to calculate contract of GLCM

float asm_glcm(pdf)
matrix pdf;
Subroutine to calculate angular second moment of GLCM

float idm_glcm(pdf)
matrix pdf;
Subroutine to calculate inverse difference moment of GLCM

float entropy_glcm(pdf)
matrix pdf;
Subroutine to calculate entropy of GLCM

float corr_glcm(pdf,m,s)
matrix pdf;
float *m,*s;
Subroutine to calculate correlation of GLCM
if *s=0.0 return(-1).
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Co-occurence Parameters". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Texture" and "Co-occurence Parameters". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: Subroutine to calculate the following parameters of gray level co-occurence matrices: angular second moment, inverse difference moment, entropy and correlation. All subroutines return float numbers.
RETURN VALUE: float number, -1 (if pdf == NULL or divide by zero occurs).
SEE ALSO: glcmarr

15.23 Test Programs

The subroutines for counting binary objects from this chapter can be tested using the program bellow:

```
/* Test program to count binary objects in an image                */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graph.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(argc,argv)
int argc;
char *argv[];
{ int i,N,NN,MM;
/* define two image buffers  a, b                                  */
  image a,b;
/* define one signal buffer h                                      */
  vector h;
  if(argc!=5) {
    printf("\nSyntax : count [NMAX] [MMAX] [input image file] [output file]");
    exit(1); }

/* define the image buffer size
   the window size  15x15 and the vector size                      */
  NN=atoi(argv[1]);
  MM=atoi(argv[2]);
  init_eikona(NN,MM,15,15,256);
/* create two image buffers a, b of size  NNxMM                 */
  a=build_image(NN,MM);
  b=build_image(NN,MM);
/* create a vector h of size 256                                   */
  h=build_vector(256);
/* load image, initialize graphics and display image               */
  fromdisk(argv[3], a, 0, 0, NN,MM);
  init_graphics("VRES16COLOR");
  _settextposition(1,5); _outtext("original image");
  _settextposition(1,5); _outtext("original image");
  imdisp(a, 16, 16, 0, 0, 0, NN,MM);
```

```
/* take the image negative and display it                    */
  neg(a, b, 0, 0, NN,MM);
  _settextposition(1,40);
  printf("negative  image");
  imdisp(b, 16, 16, MM, 0, 0, NN,MM);

/* find negative image histogram                             */
  _settextposition(21,1);
  printf("histogram calculation starts...");
  hist(b, h, 0, 0, NN,MM);
  sigshow(h, 256, 10, 10, 400, 300);

/* threshold the image and display it                        */
  _settextposition(1,1);
  printf("negative image histogram (intensities 0-255)-choose a threshold:");
  scanf("%d",&i);
  clearscreen();
  thres(b, b, i, 0, 0, NN,MM);
  _settextposition(1,5);
  printf("original  image");
  imdisp(a, 16, 16, 0, 0, 0, NN,MM);
  _settextposition(1,40);
  printf("thresholded image");
  bindisp(b, 16, 16, MM, 0, 0, NN,MM);
  getch();
/* perform counting and store the results on buffer h        */
  _settextposition(21,1);
  printf("counting starts....");
  for(i=0; i<256; i++) h[i]=(float)(0.0);
  count(b, &N, h, 0, 0, NN,MM);

/* display the results of counting and store them in a file */
  reset_graphics();
  _settextposition(1,1);
  printf("%d particles have been found", N);
  for(i=0; i<N; i++)
  printf("\n particle %2d has area %4.0f pixels", i, h[i]);
  printf("\n \n the results are now stored");
  dump_signal(h,argv[4]);
  return(0);
}
```

The syntax of the command line is self-explanatory by the program, after giving:

<Program Name>

Subroutines for region segmentation from this chapter can be tested using the program bellow:

```
 /*      MAIN PROGRAM FOR SEGMENTATION         */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <float.h>
#include <math.h>
#include <malloc.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main()
 {
  int x,y,N,i,T,select,ret,nr,REGMAX;
  image im;
  image imout;
  char filename[60];
  unsigned char far* MARRAY;


  printf("NMAX="); scanf("%d",&NMAX);
  printf("MMAX="); scanf("%d",&MMAX);

  init_eikona(NMAX,MMAX,-1,-1,-1);
  im=build_image(NMAX,MMAX);
  imout=build_image(NMAX,MMAX);
  clear(im,0,0,0,NMAX,MMAX);
  clear(imout,0,0,0,NMAX,MMAX);

  printf("Input  image >") ; scanf("%s",filename) ;
  fromdisk(filename,im,0,0,NMAX,MMAX);



  printf("\nOutput image  >") ; scanf("%s",filename) ;
```

```
do {
     printf(" 0.REGION_SEGM \n");
     printf(" 1.REGION_GROW \n");
     printf(" 2.REGION_MERGE \n");
     printf(" 3.REGION_SPLIT \n");
     printf(" 4.REGION_SPLIT_MERGE\n");
     printf(" Which segmentation method  >" );
     scanf("%d",&select); }
while (  (select<0)  && (select>5) ) ;

 switch(select)  {
       case 0 :
           printf("nr="); scanf("%d",&nr);
           segm(im,imout,nr,0,0,NMAX,MMAX);
           break;
       case 1 :
         printf(" T = ") ; scanf("%d",&T) ;
         printf("N="); scanf("%d",&N);
         for (i=0 ; i<N ; i++)
        {
         printf("Give x,y seed for image >") ; scanf("%d %d",&x,&y);
         imout[y][x]=i+1;
        }

        region_grow(im,imout,0,0,NMAX,MMAX,N,T);
        break;
       case 2 :
       printf(" T = ") ; scanf("%d",&T) ;
       printf("REGMAX="); scanf("%d",&REGMAX);
       N=region_merge(im,imout,0,0,NMAX,MMAX,T,REGMAX);
       printf ( " segmentation in %3d  regions \n",N);
       break;
       case 3 :
        printf(" T = ") ; scanf("%d",&T) ;
        N=0;
        region_split(im,imout,0,0,NMAX,MMAX,&N,T);
        printf ( " segmentation in %3d  regions \n",N);
        break;
       case 4 :
   printf(" T = ") ; scanf("%d",&T) ;
   N=0;
   printf("REGMAX="); scanf("%d",&REGMAX);
   MARRAY=(unsigned char far*)_fmalloc((unsigned int)REGMAX*sizeof(unsigned char));
   if (MARRAY==NULL) return(-10);
```

```
   ret=region_split_merge(im,imout,MARRAY,0,0,NMAX,MMAX,&N,T,REGMAX);
   printf ( " segmentation in %3d  regions \n",N);
   break;
   }

    init_graphics("VRES16COLOR");
    imdisp(im, 16,0,0,0,0,NMAX,MMAX);
    imdisp(imout, 16,0,MMAX,0,0,NMAX,MMAX);
    getch();
    reset_graphics();


  todisk(imout,filename,0,0,NMAX,MMAX);


  mfreeuc2(im,MMAX);
  mfreeuc2(imout,MMAX);
}
```

The program runs simply by its name:

```
<Program Name>
```

The program guides the user through 5 different options: 1: Region Segmentation.
2: Region Grow.
3: Region Merge.
4: Region Split.
5: Region Split – Merge.

Subroutines for parameter evaluation from this chapter can be tested using the program bellow:

```
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <conio.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;
```

```
/*-------------------------------------------------------------------*/

main()
{
 int option,nmax,mmax,nsig;


 void inputimage(void);


 printf("\n NMAX  = ");scanf("%d",&nmax);
 printf("\n MMAX  =");scanf("%d",&mmax);
 printf("\n NSIG  = ");scanf("%d",&nsig);
 init_eikona(nmax,mmax,0,0,nsig);

 option=0;
 do {
      printf("\nMAIN MENU");
      printf("\n(1)Input image");
      printf("\n(2)Exit to dos");
      printf("\nEnter your option");
      scanf("%d",&option);
      switch(option)
      {
     case 1:
        inputimage();
        break;
     case 2:
        exit(0);
        break;
      }
    }
 while ( option != 1 || option != 2 );
 return(0);
}

/*-------------------------------------------------------------------*/

void inputimage()
{
 image im1;
 int n,m,hisopt;
 char filename[60];
```

```
void histo(image);
void difhisto(image);
void runhisto(image);
void glcmarro(image);

im1=build_image(NMAX,MMAX);
printf("\nN = ");scanf("%d",&n);
printf("\nM = ");scanf("%d",&m);
do {
    printf("\n Input image >");
    gets(filename);
   }
while (fromdisk(filename,im1,0,0,n,m) != 0);

hisopt=0;

do {
    printf("\n(1)Histogram Calculation");
    printf("\n(2)Histogram of gray-level differences");
    printf("\n(3)Run length calculation");
    printf("\n(4)Gray Level Cooccurrence Matrices");
    printf("\n(5)Exit");
    do {
     printf("\nEnter option");
     scanf("%d",&hisopt);
     printf(" ");
   }
    while ( hisopt < 1 || hisopt > 5 );
    switch(hisopt)
      {
    case 1 :
      histo(im1);
      break;
    case 2 :
      difhisto(im1);
      break;
    case 3 :
      runhisto(im1);
      break;
    case 4 :
      glcmarro(im1);
      break;
    case 5 :
```

```
       break;
       }
    }
 while ( hisopt != 5 );
 mfreeuc2(im1,NMAX);
 return;
}


void histo(a)
image a;
{
 vector histogram;
 int opt1,N1,M1,N2,M2;
 float meanvalue,variance,skewness,kurtosis,entropy;

 int hist(image,vector,int,int,int,int);
 float meanvalu(vector);
 float varianc(vector);
 float skewnes(vector);
 float kurtosi(vector);
 float entrop(vector);

 histogram=build_vector(NSIG);
 printf("\nN1 =");scanf("%d",&N1);
 printf("\nM1 =");scanf("%d",&M1);
 printf("\nN2 =");scanf("%d",&N2);
 printf("\nM2 =");scanf("%d",&M2);
 do {
     printf("Compute histogram...");
     hist(a,histogram,N1,M1,N2,M2);
    }
 while ( hist(a,histogram,N1,M1,N2,M2) != 0);
 printf("\nOk.");
 opt1=0;

 do {
      printf("\nMENU");
      printf("\n(1)Meanvalue");
      printf("\n(2)Variance");
      printf("\n(3)Skewness");
      printf("\n(4)Kurtosis");
      printf("\n(5)Entropy");
      printf("\n(6)Exit");
```

```
      do {
       printf("\nEnter option");
       scanf("%d",&opt1);
     }
      while ( opt1 < 1 || opt1 > 6 );
      switch(opt1)
    {
      case 1 :

        { meanvalue=meanvalu(histogram);
        printf("\n %f",meanvalue);}
        break;
      case 2 :
       { variance=varianc(histogram);
         printf("\n %f",variance);}
        break;
      case 3 :
       { skewness=skewnes(histogram);
        printf("\n %f",skewness);}
        break;
      case 4 :
       { kurtosis=kurtosi(histogram);
        printf("\n %f",kurtosis);}
        break;
      case 5 :
       { entropy=entrop(histogram);
        printf("\n %f",entropy);}
        break;
      case 6 :
        break;
    }
    }
 while ( opt1 != 6 );

 _ffree(histogram);
 return;
}

/*-------------------------------------------------------------------*/

void difhisto(a)
image a;
{
 vector histogram;
```

```
int opt2,DX,DY;
float a1,a2,a3,a4;

int difhist(image,vector,int,int,int,int,int,int);
float c_d(vector);
float acs(vector);
float entropy_d(vector);
float m_d(vector);

histogram=build_vector(NSIG);
do {
    printf("x="); scanf("%d",&DX);
    printf("y="); scanf("%d",&DY);
    printf("Compute differences histogram...");
    difhist(a,histogram,DX,DY,0,0,NMAX,MMAX);
   }
while ( difhist(a,histogram,DX,DY,0,0,NMAX,MMAX) !=0 );
opt2=0;
do {
     printf("\nMENU");
     printf("\nEstimate :");
     printf("\n(1) A1");
     printf("\n(2) A2");
     printf("\n(3) A3");
     printf("\n(4) A4");
     printf("\n(5) Exit");
     do  {
       printf("\tEnter option");
       scanf("%d",&opt2);
     }
     while ( opt2<1 || opt2>5 );
     switch(opt2)
   {
     case 1 :
      { a1=c_d(histogram);
       printf("\n %f",a1);}
       break;
     case 2 :
      { a2=acs(histogram);
        printf("\n %f",a2);}
        break;
     case 3 :
       { a3=entropy_d(histogram);
         printf("\n entropy=%e",a3);}
```

```
        break;
      case 4 :
        {a4=m_d(histogram);
         printf("\n %f",a4);}
         break;
      case 5 :
        break;
    }
    }
 while (opt2 != 5);

 _ffree(histogram);
 return;
}
/*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
void runhisto(b)
image b;
{
 double a1,a2,a3,a4,a5;
 int opt3,angle;
 long int sum;
 imatrix runlength;
 float maxmagn;

 double T_R(imatrix);
 double sre(imatrix);
 double lre(imatrix);
 double gld(imatrix);
 double rld(imatrix);
 double rp(imatrix);
 int runlen(image,imatrix,int,int,int,int,int);

 sum=(long int)((long int)NMAX*NMAX+(long int)MMAX*MMAX);
 maxmagn=(float)fabs(sqrt((double)sum));
 runlength=build_imatrix
    ((unsigned int)NSIG,(unsigned int)maxmagn);
 do {
      printf(" Input angle ? \n");scanf("%d",&angle);
     }
 while(runlen(b,runlength,angle,0,0,NMAX,MMAX)!=0);
 opt3=0;
 do {
```

```
      printf("\nMENU");
      printf("\nEstimate :");
      printf("\n(1) A1");
      printf("\n(2) A2");
      printf("\n(3) A3");
      printf("\n(4) A4");
      printf("\n(5) A5");
      printf("\n(6) Exit");
      do  {
        printf("\nEnter option");
        scanf("%d",&opt3);
      }
      while ( opt3<1 || opt3>6 );
      switch(opt3)
    {
      case 1 :
        { a1=sre(runlength);
          printf("\n %e",a1);}
        break;
      case 2 :
       { a2=lre(runlength);
        printf("\n %e",a2);}
        break;
      case 3 :
       { a3=gld(runlength);
        printf("\n %e",a3);}
        break;
      case 4 :
        {a4=rld(runlength);
        printf("\n %e",a4);}
        break;
      case 5 :
        {a5=rp(runlength);
        printf("\n %e",a5);}
        break;
      case 6 :
        break;
    }
    }
 while (opt3 != 6);

 mfreein2(runlength,NSIG);
 return;
}
```

```
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/

void glcmarro(ima)
image ima;
{
 int opt4;
 float a1,a2,a3,a4,a5;
 float meanvalue,variance;
 int DX,DY;
 matrix pdfarr;

 int glcmarr(image,matrix,int,int,int,int,int,int);
 float c_glcm(matrix);
 float asm_glcm(matrix);
 float idm_glcm(matrix);
 float entropy_glcm(matrix);
 float corr_glcm();

 do {
 pdfarr=matf2(NSIG,NSIG);
 printf("x="); scanf("%d",&DX);
 printf("y="); scanf("%d",&DY);
    }
 while (glcmarr(ima,pdfarr,DX,DY,0,0,NMAX,MMAX) != 0);
 opt4=0;
 do {
      printf("\nMENU");
      printf("\nEstimate :");
      printf("\n(1) A1");
      printf("\n(2) A2");
      printf("\n(3) A3");
      printf("\n(4) A4");
      printf("\n(5) A5");
      printf("\n(6) Exit");
      do  {
        printf("\nEnter option");
        scanf("%d",&opt4);
      }
      while ( opt4<1 || opt4>6 );
      switch(opt4)
    {
      case 1 :
        { a1=c_glcm(pdfarr);
```

```
          printf("\n c_glcm=%f",a1); }
        break;
      case 2 :
       { a2=asm_glcm(pdfarr);
        printf("\n %f",a2);}
        break;
      case 3 :
        {a3=idm_glcm(pdfarr);
         printf("\n %f",a3);}
        break;
      case 4 :
        {a4=entropy_glcm(pdfarr);
         printf("\n %f",a4);}
        break;
      case 5 :  {
        printf("Input meanvalue : \n");scanf("%f",&meanvalue);
        /*fgets(ma,10,stdin);
        meanvalue=(float)atof(ma);*/
        printf("%f",meanvalue);
        printf("Input variance  : \n");scanf("%f",&variance);
        /*fgets(va,10,stdin);
        variance=(float)atof(va);*/
        printf("%f",variance);
        a5=corr_glcm(pdfarr,&meanvalue,&variance);
         printf("\n %f",a5);}
        break;
      case 6 :
        break;
    }
    }
 while (opt4 != 6);

 mfreef2(pdfarr,NSIG);
 return;
}
```

The program runs simply by its name:

```
<Program Name>
```

and is extremely user friendly and self explanatory.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[LEV85] M.D.Levine, *Vision in man and machine*, McGraw-Hill, 1985.

16

Shape description

16.1 chain_code()

Subroutine to perform chain coding
LIBRARY MODE:

```
int chain_code(a, ch, is, js, TMAX, N1, M1, N2, M2)
imagea;
CHAIN   *ch;
int     *is, *js;
int N1,M1,N2,M2;
long TMAX;

a:  input image
ch:    output chain code (linked list)
is,js: output starting points of image
TMAX: maximal allowable number of chain nodes
N1,M1: start coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Chain Coding". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Chain Coding". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine to perform chain coding. If the borders of ROI have been violated it returns -93. If the turtle follower is running wild, it returns -94. Only one object can be followed. The object should lie inside the ROI borders.
RETURN VALUE: 0, -1, -10, -21, -90, -93, -94, -95, -100
SEE ALSO: chain_decode

16.2 create_node()

Subroutine to create a node of a quadtree as a child of node father
LIBRARY MODE:

```
QNODE far* create_node(father,t,col)
QNODE far* father;
unsigned char t,col;

father: pointer to the father of the node to be created
t   : the orientation of the node to be created
col : the color of the node
```

SHORT DESCRIPTION: Subroutine to create a node of a quadtree as a child of node father with orientation t and color col.
RETURN VALUE: QNODE *, NULL
SEE ALSO:

16.3 curve_split()

Subroutine for polygon approximation of curve by splitting.
LIBRARY MODE:

```
int curve_split(x, p, n, T)
PIXEL far* x;
LIST *p;
int n;
```

```
double T;

x: array of PIXELs
p: output list of polygon vertices
n: length of pixel array
T: threshold
```

SHORT DESCRIPTION: This subroutine performs polygon approximation of curve by splitting. It uses routine dist() for Euclidean distance calculation.
RETURN VALUE: 0, -2, -10, -90
SEE ALSO: curve_merge

16.4 curve_merge()

Subroutine for polygon approximation of curve by merging.
LIBRARY MODE:

```
int curve_merge(x, p, n, T)
PIXEL far* x;
LIST *p;
int n;
double T;

x: array of PIXELs
p: output list of polygon vertices
n: length of pixel array
T: threshold
```

SHORT DESCRIPTION: This subroutine performs polygon approximation of curve by merging. It uses routine dist() for Euclidean distance calculation.
RETURN VALUE: 0, -10
SEE ALSO: curve_split

16.5 create_curve()

Subroutine to follow a border.
LIBRARY MODE:

```
int create_curve(a,x,c, NS,N1,M1, N2, M2)
image a;
PIXEL far* x;
int  *c;
int NS, N1,M1, N2, M2;

a: input image
x: output array
c: number of border pixels found
NS: size of x
N1, M1: start coordinates
N2,M2: end coordinates
```

SHORT DESCRIPTION: This subroutine follows a border. Border coordinates are stored in output array of pixels x. It returns -93 if ROI border has been violated. It returns -96 if the pixels found are more than the size of x (NS).
RETURN VALUE: 0, -1, -2, -21, -90, -93, -95, -96, -100
SEE ALSO: turtle_follow

16.6 chain_decode()

Subroutine to perform chain decoding
LIBRARY MODE:

```
int chain_decode(a, ch, is, js, N1,M1,N2,M2)
imagea;
CHAIN   ch;
int  is, js;
int N1,M1,N2,M2;

a: output image
ch: input chain
is,js: input image starting points
N1,M1: start coordinates
N2,M2: end coordinates
```

SHORT DESCRIPTION: This subroutine performs chain decoding. It returns -93 if the borders of ROI are violated.
RETURN VALUE: 0, -1, -2, -21, -90, -93.
SEE ALSO: chain_code

16.7 compress_chain()

Subroutine to compress a chain
LIBRARY MODE:

```
CHAIN compress_chain(chain)
CHAIN chain;

ch : input chain
```

SHORT DESCRIPTION: This subroutine compresses 4 nodes of an input chain into one node of the output chain. The subroutine as is, cannot distinguish between direction 0 and end of chain at the last compressed node.
RETURN VALUE: CHAIN
SEE ALSO: decompress_chain

16.8 construct_quadtree()

Subroutine to build the quadtree from an image region
LIBRARY MODE:

```
int construct_quadtree(a,ptr,level,col,N1,M1,x,y)
image a;
QNODE far* far* ptr;
unsigned char level,*col;
int x,y,N1,M1;

ptr  : pointer to the subtree to be constructed
level: New level of the quadtree. (level is 0 for
```

```
       nodes representing pixels.)
col  : Node color to be returned. (0,1, GRAY for non leaf nodes).
N1,M1: starting point of the region of interest in the image
x,y  : The coordinates of the south-most,
       east-most pixel of the square
       subregion which will be visited.
image: Pointer to the array which is the image stored.
```

SHORT DESCRIPTION: It is a recursive routine which builds the quadtree which corresponds to the image of dimensions $2^l x 2^l$ (l=level).
RETURN VALUE: 0, -10
SEE ALSO: image_to_quadtree

16.9 calc_features_1()

Subroutine to calculate moment-related features
LIBRARY MODE:

```
int calc_features_1(x,N1,M1,N2,M2,area,xmean,ymean,orientation,
          eccentricity1, eccentricity2,spread)
image x;
int N1,M1,N2,M2;
float *area,*xmean,*ymean,*orientation,
 *eccentricity1,*eccentricity2,*spread;
x: input image buffer
N1,M1: start coordinates
N2,M2: end coordinates
area: object area
xmean, ymean: center of gravity coordinates
   orientation: object orientation
   eccentricity1, eccentricity2: object eccentricities
   spread: object spread
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Moment Features". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Moment Features". Fill-in the required fields and press "OK".

SHORT DESCRIPTION: This subroutine calculates moment-related features: object area, center of gravity coordinates, object orientation, object eccentricities and object spread. It uses find_moment() for moment calculation.
RETURN VALUE: 0, -1, -21, -89, -100.
SEE ALSO: find_moment, calc_features_2

16.10 calc_features_2()

Subroutine to calculate boundary-related features
LIBRARY MODE:

```
int calc_features_2(l,perimeter,bending_energy,norm_bending_en,
                    compactness, norm_compactness,area)
LIST l;
float *perimeter,*bending_energy,*norm_bending_en,*compactness,
*norm_compactness;
float area;

l: list of pixels
perimeter: object perimeter
bending_energy: object bending energy
norm_bending_en: normalized object bending energy
compactness: object compactness
norm_compactness: normalized compactness
area: object area
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Boundary Features". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Boundary Features". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates moment-related features: perimeter, bending energy, normalized object bending energy, compactness, normalized compactness and object area.
RETURN VALUE: 0, -2, -89, -90
SEE ALSO: calc_features_1

16.11 decompress_chain()

Subroutine to decompress a chain
LIBRARY MODE:

```
CHAIN decompress_chain(chain)
CHAIN chain;

ch : input chain
```

SHORT DESCRIPTION: This subroutine decompresses 1 node of a compressed chain into 4 nodes of the output chain. The subroutine as is cannot distinguish between direction 0 and end of chain at the last compressed node. It may add up to 3 extra nodes of direction 0 at the end of the decompressed list.
RETURN VALUE: CHAIN
SEE ALSO:compress_chain

16.12 dist()

Subroutine to calculate Euclidean distance
LIBRARY MODE:

```
double dist(x1, y1, x2, y2, x3, y3)
int x1, y1, x2, y2, x3, y3;

x1,y1: line  start
x2,y2: point
x3,y3: line end
```

SHORT DESCRIPTION: Subroutine to calculate the Euclidean distance of point (x2,y2) from the straight line (x1,y1),(x3,y3).
RETURN VALUE: double number
SEE ALSO: curve_split, curve_merge

16.13 findch()

Subroutine to calculate object characteristics
LIBRARY MODE:

```
int findch(a,t,parea,pKBx,pKBy,pdeg,pvolume, N1,M1,N2,M2)
image a;
int t;
float *parea,*pKBx,*pKBy,*pdeg,*pvolume;
int N1,M1,N2,M2;

a: image buffer
t: threshold above which an object is present
parea: object area (in pixels)
pKBx, pKBy: center of gravity coordinates
pdeg: angle (in degrees) between the inertia axis
      of the object and x axis
pvolume: object volume
N1,M1: starting coordinates of ROI
N2,M2: end coordinates of ROI
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Find char.". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Find char.". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the area, center of gravity, rotation angle and volume of an object in the area of interest. The object must be located in the region of interest (N1,M1), (N2,M2). If angle is undefined, it returns pdeg = 0.
RETURN VALUE: 0, -1, -21
SEE ALSO: calc_features_1, calc_features_2

16.14 fd2image()

Subroutine to create image from its Fourier Descriptors
LIBRARY MODE:

```
int fd2image(xr,xi,a,c,nn,N1,M1,N2,M2)
```

```
vector xr,xi;
image a;
int c,nn;
int N1,M1,N2,M2;

xr,xi:  input FD's (real and imaginary parts)
a:  output image
c:  input number of FD's
nn:    length of vectors xr, xi
N1,M1: start coordinates
N2,M2: end coordinates
```

SHORT DESCRIPTION: Subroutine to create image from its Fourier Descriptors It uses fftc() to calculate the 1d inverse DFT.
RETURN VALUE: 0, -1, -2, -21, -33, -90, return values of fftc().
SEE ALSO: image2fd

16.15 find_level()

Subroutine to find the level of a quadtree
LIBRARY MODE:

```
int find_level(pr,lev)
QNODE far* pr;
int *lev;

pr : Pointer to the root of the quadtree
lev: The level at each stage of recursion
```

SHORT DESCRIPTION: This subroutine finds the level of a quadtree. It is a recursive routine.
RETURN VALUE: 0
SEE ALSO: quadtree_to_image

16.16 find_moment()

Subroutine to calculate object moments
LIBRARY MODE:

```
float find_moment(x,p,q,N1,M1,N2,M2,_x,_y)
image x;
int p,q,N1,M1,N2,M2;
float _x,_y;

x: input image buffer
p,q: moment indices
N1,M1: start coordinates
N2,M2: end coordinates
_x,_y: central point coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Central Moment". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Central Moment". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine calculates the sum $\sum_i \sum_i [(j - \_x)^p * (i - \_y)^q]$ over the ROI of the binary image x. Values of both p and q must be greater or equal to zero.
RETURN VALUE: float.
SEE ALSO: findch

16.17 find_perimeter()

Subroutine to calculate object perimeter
LIBRARY MODE:

```
float find_perimeter(l)
LIST l;

l: list of pixels
```

SHORT DESCRIPTION: This subroutine calculates object perimeter, by using the list of its border pixels.

RETURN VALUE: float
SEE ALSO: find_bending_energy_sum

16.18 find_bending_energy_sum()

Subroutine to calculate bending energy
LIBRARY MODE:

```
float find_bending_energy_sum(l)
LIST l;

l: list of pixels
```

SHORT DESCRIPTION: This subroutine calculates object bending energy, by using the list of its border pixels.
RETURN VALUE: float
SEE ALSO: find_perimeter

16.19 image2fd()

Subroutine to calculate Fourier Descriptors from a binary image
LIBRARY MODE:

```
int image2fd(a,xr,xi,c,nn,N1,M1,N2,M2)
image a;
vector xr,xi;
int *c,nn;
int N1,M1,N2,M2;

a:  input image
xr,xi: output data arrays (real and imaginary parts)
c:     output number of complex FD's
nn:    length of vectors xr, xi
N1,M1: start coordinates
N2,M2: end coordinates
```

SHORT DESCRIPTION: This subroutine calculate the Fourier Descriptors from a binary image. It uses fftc() to calculate the 1d DFT. It returns -93 if ROI borders have been violated. It returns -96 if the curve pixels exceed nn (the length of vectors).
RETURN VALUE: 0,-1,-2,-10,-21,-90,-93,-95,-96,return values of fftc().
SEE ALSO: fftc, fd2image

16.20 image_to_quadtree()

Subroutine to build a quadtree from an image.
LIBRARY MODE:

```
int image_to_quadtree(a,N1,M1,N2,M2,root)
image a;
int N1,N2,M1,M2;
QNODE far* far* root;

a: array of the binary image
N1,M1: start coordinates
N2,M2: end   coordinates
root: pointer to the root of the quadtree
      which the routine returns
```

SHORT DESCRIPTION: This subroutine builds a quadtree from a square image stored in array a. The procedure returns a pointer to the root of the quadtree. The image must be binary. It uses subroutine construct_quadtree(). It returns on errors: -34 (image is not square) and -33 (image dimensions is not a power of 2).
RETURN VALUE: 0, -1, -21, -33, -34, -100
SEE ALSO: construct_quadtree, quadtree_to_image

16.21 make_image()

Subroutine to construct the binary image from a quadtree.
LIBRARY MODE:

```
int make_image(ptr,a,level,N1,M1,x,y)
QNODE far* ptr;
image a;
int level,x,y,N1,M1;

ptr  : Pointer to the node of the quadtree which is visited
a    : output image buffer
level: The level of the quadtree.
       The size of the image i 2^levelx2^level
N1,M1: Destination point on the output image buffer.
x,y  : The coordinates of the south-most,
       east-most square subregion of the image
```

SHORT DESCRIPTION: This recursive routine constructs the binary image from a quadtree. It is used in quadtree_to_image().
RETURN VALUE: 0
SEE ALSO: quadtree_to_image

16.22 pyramid_1()

Subroutine that creates the pyramid and stores pyramid levels to an output image.
LIBRARY MODE:

```
int pyramid_1(x,y,N,M)
image x,y;
int N,M;

x: input image buffer
y: output image buffer
N,M: sizes of input and output image
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis" and "Pyramid". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis" and "Pyramid". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine creates the pyramid of input binary image x and stores pyramid levels (except the lowest one) to output image y. It

uses OR binary operations in the 2x2 image neighborhood to pass from one level to the next. It return values -34 if the image is not square.
RETURN VALUE: 0, -1, -34, -100
SEE ALSO: pyramid_fill

16.23 pyramid_alloc()

Subroutine to allocate memory for a pyramid
LIBRARY MODE:

```
pyramid pyramid_alloc(N)
int N;
N: size of lowest level image
```

SHORT DESCRIPTION: This subroutine allocates memory for a pyramid with $(\log_2 N + 1)$ levels.
RETURN VALUE: pyramid, NULL
SEE ALSO: pyramid_fill

16.24 pyramid_free()

Subroutine that frees allocated memory
LIBRARY MODE:

```
int pyramid_free(p,N)
pyramid p;
int N;

N: size of lowest level image
```

SHORT DESCRIPTION: This subroutine frees allocated memory for pyramid p with $(\log_2 N + 1)$ levels.
RETURN VALUE: 0
SEE ALSO: pyramid_alloc

16.25 pyramid_fill()

Subroutine to create the pyramid of input binary image
LIBRARY MODE:

```
int pyramid_fill(im,x,N)
image im;
pyramid x;
int N;

im: input image buffer
x: output pyramid
N: size of input image
```

SHORT DESCRIPTION: This subroutine creates the pyramid of input binary image im and stores pyramid levels to an output pyramid structure. The pyramid levers are calculated using OR operations.
RETURN VALUE: 0, -1, -90
SEE ALSO: pyramid_1

16.26 pyramid_edge_detection()

Subroutine for edge detection using a pyramid structure
LIBRARY MODE:

```
int pyramid_edge_detection(x,e,N,k,T)
pyramid x,e;
int N,k;
int T;

x:input pyramid
e:output pyramid containing the edge detection results
N:size of lowest level image
k:start level for edge detection
T:Threshold
```

SHORT DESCRIPTION: This subroutine performs edge detection using a pyramid structure. It performs local edge detection and refines the detected edges, i.e. the ones that have level greater than T.
RETURN VALUE: 0, -1, -90
SEE ALSO: sobel, range

16.27 quadtree_to_image()

Subroutine for loading a quadtree to an image buffer
LIBRARY MODE:

```
int quadtree_to_image(pt,a,N1,M1,N2,M2)
QNODE far* pt;
image a;
int N1,M1,N2,M2;

pt    : Pointer to the root of the quadtree.
a     : output image buffer.
N1,M1 : Start point on the output buffer.
N2,M2 : End point on the output buffer.
```

SHORT DESCRIPTION: This subroutine performs loading a quadtree to an image buffer. It uses find_level() to find the level of the quadtree and make_image() to make the binary image output.
RETURN VALUE: 0, -2, -21, -22
SEE ALSO: find_level, make_image

16.28 turtle_follow()

Subroutine to perform turtle following
LIBRARY MODE:

```
int turtle_follow(x,lst,N1,M1,N2,M2)
image x;
```

```
LIST *lst;
int N1,M1,N2,M2;

x: binary input image buffer
lst: pointer to list of points
N1,M1: start coordinates
N2,M2: end coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Turtle Follow". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Shape Analysis" and "Turtle Follow". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs turtle following in a binary source image.
RETURN VALUE: 0, -1, -21, -93, -94, -95, -100
SEE ALSO: chain_code, create_curve

16.29 thin_1()

Subroutine for one pass thinning algorithm
LIBRARY MODE:

```
int thin_1(x,N1,M1,N2,M2)
image x;
int N1,M1,N2,M2;

x:input and output image buffer
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Thinning" and "One Pass Thinning". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Thinning" and "One Pass Thinning". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs one pass thinning algorithm.
RETURN VALUE: 0, -1, -21

SEE ALSO: thin_2

16.30 thin_2()

Subroutine for two pass thinning algorithm
LIBRARY MODE:

```
int thin_2(x,N1,M1,N2,M2)
image x;
int N1,M1,N2,M2;

x:input and output image buffer
N1,M1: start coordinates
N2,M2: end   coordinates
```

MS–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Thinning" and "Two Pass Thinning". Fill-in the required fields and press "OK".
X–WINDOWS ENVIRONMENT: Open the "Black and White" menu, then consecutively select "Analysis", "Thinning" and "Two Pass Thinning". Fill-in the required fields and press "OK".
SHORT DESCRIPTION: This subroutine performs a two-pass thinning algorithm.
RETURN VALUE: 0, -1, -10, -21
SEE ALSO: thin_1

16.31 Test Programs

Subroutines for chain coding and decoding from this chapter can be tested using the program bellow:

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

    PROGRAM:  CHAIN CODE
              chain code/decode an image

    ROUTINES: chain_code(), chain_decode()


*************************************************************************/

#include    <stdio.h>
#include    <string.h>
#include    <stdlib.h>
#include    <conio.h>
#include    <malloc.h>
#include    <math.h>
#include    "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main()
{
    image   im;
    CHAIN   ch;
    char    fn1[15], fn2[15], fn3[15];
    int     is, js;
    int     d=0;
    short   ans;
    FILE    *fp;
    char    *s;
    int     chain_code(image, CHAIN*, int*, int*,int,int,int,int,int);
    int     chain_decode(image, CHAIN, int, int,int,int,int,int);

    int EOCH=-111;  /*end of chain marker*/

    printf("\n *** CHAIN CODE ***\n");

    printf("image filename (.PIC):"); scanf("%s", fn1);
```

```
strcpy(fn2, fn1);
strcpy(fn3, fn1);
strcat(fn1, ".pic");
strcat(fn2, ".out");
strcat(fn3, ".chn");

/* Menu */
printf("\n CODE IMAGE .... 1\n");
printf(" DECODE IMAGE .. 2\n");
printf(" EXIT ......... 0\n\n");
printf(" select :"); scanf("%d", &ans);
init_eikona(256,256,-1,-1,-1);
im=build_image(256,256);

/* CODE IMAGE */
if (ans == 1) {
    printf("%s->%s\n", fn1, fn3);
printf("loading image and converting it to binary...\n");
fromdisk(fn1,im,0,0,NMAX,MMAX);
thres(im,im,1,0,0,NMAX,MMAX);
init_graphics("VRES16COLOR");
bindisp(im, 16,0,0,0,0,NMAX,MMAX);
getch();
reset_graphics();

printf("chain coding image...\n");
chain_code(im, &ch, &is, &js,3000,0,0,NMAX,MMAX);
printf("saving chain...\n");
fprint_chain(fn3, is, js, ch);
mfreeuc2(im,MMAX);
free(ch.start); free(ch.end);
}

/* DECODE IMAGE */
if (ans == 2) {
    printf("%s->%s\n", fn3, fn2);
    printf("creating & loading chain ...\n");

    /* create chain */
ch.start = ch.end = getchnode();
    if ( (fp = fopen(fn3, "r")) == NULL)
    {
        printf("unable to fopen\n");
        exit(0);
```

```
        }

        /* read starting points */
        fscanf(fp, "%s", s); is = atoi(s);
        fscanf(fp, "%s", s); js = atoi(s);
        d = 0;
        /* read chain */
        while (d!=EOCH) {
            fscanf(fp,"%s", s); d = atoi(s);
        ch.end = addchnode((byte)d, ch.end);
        }

        /* allocate space for image */
        printf("chain decoding...\n");
    fromdisk(fn1,im,0,0,NMAX,MMAX);
    thres(im,im,1,0,0,NMAX,MMAX);
    init_graphics("VRES16COLOR");
    bindisp(im, 16,0,0,0,0,NMAX,MMAX);
    clear(im,0,0,0,NMAX,MMAX);
    chain_decode(im, ch, is, js,0,0,NMAX,MMAX);
    bindisp(im, 16,0,256,0,0,NMAX,MMAX);
    getch();
    reset_graphics();
    printf("saving image...\n");
    todisk(im,fn2,0,0,MMAX,NMAX);
    mfreeuc2(im, NMAX);
    free(ch.start); free(ch.end);
    }
    exit(0);
    return(0);
}
```

The program guides the user and is straightforward to use.

Subroutines for polygon approximation from this chapter can be tested using the program bellow:

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

    PROGRAM:  POLYGON APPROXIMATION
              split & merge algorithms for polygon approximation
    ROUTINES: split_curve()
              merge_curve()
    NOTES:    split_curve() & merge_curve() use dist()
```

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

#include    <stdio.h>
#include    <stdlib.h>
#include <conio.h>
#include <string.h>
#include    <malloc.h>
#include    <math.h>
#include    "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc, char **argv)
{   int    c;
    unsigned char   d=0;
    double  T;
    int    ans, err, t;
    char   *s;
    LNODE  *lptr;
    LIST   l;
    char   fn1[40], fn2[40];
    PIXEL  *x;
    image  im1;
    int NS=3000;
    int    curve_split(PIXEL*, LIST*,int, double);
    int    curve_merge(PIXEL*, LIST*,int, double);
    double dist(int, int, int, int, int, int);

    printf("\n *** POLYGON APPROXIMATION ***\n\n");
    printf("image filename (.PIC):"); scanf("%s", fn1);
    printf("threshold value      :"); scanf("%d", &t); T = t;
    strcpy(fn2, fn1);
    strcat(fn1, ".pic");
    strcat(fn2, ".out");

    /********* The following code is necessary only in order to *********
     ********* create a list of points (xi) that make up the curve *****/
    printf("Loading image...\n");
    init_eikona(256,256,-1,-1,-1);
    im1=build_image(256,256);
    fromdisk(fn1,im1,0,0,NMAX,MMAX);
    thres(im1,im1,1,0,0,NMAX,MMAX);
```

```
     init_graphics("VRES16COLOR");
     bindisp(im1, 16,0,0,0,0,NMAX,MMAX);
     getch();

    x=(PIXEL *)malloc((NS+1)*sizeof(PIXEL));
    if(x==NULL) exit(-1);
    printf("Turtle following...\n");
    err=create_curve(im1, x, &c,NS,0,0,NMAX,MMAX);
    printf("number of points= %d err=%d\n", c,err);
   /* initialize list */
    l.start = l.end = getnode();

    printf(" Split (1) or Merge (2) method ? "); scanf("%s", s); ans = atoi(s);
    if (ans == 1) {
        err = curve_split(x, &l, (int)(0.1*atoi(argv[1])*c), T);
        l.end=add_list(x[(int)(0.1*atoi(argv[1])*c)],l.end);
    }
    if (ans == 2) {
        err = curve_merge(x, &l, (int)(0.1*atoi(argv[1])*c), T);
    }
    if (err < 0) printf("Error: %d\n", err);

    clear(im1,0,0,0,NMAX,MMAX);
     lptr=l.start;
     do{
    pixw(im1,lptr->num.y,lptr->num.x,1);
     bindisp(im1, 16,0,256,0,0,NMAX,MMAX);
     getch();
     lptr=lptr->next;
     } while(lptr!=l.end);
    reset_graphics();

    fprint_list("list",0,0,l);
    return(0);
}
```

The program, like before, guides the user and is extremely easy to use.

The subroutines for building a quadtree from an image, from this chapter can be tested using the program bellow:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
#include <stdlib.h>
#include <malloc.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(argc,argv)
int argc;
char *argv[];

{  int N,M;
   image xin,xout;
   QNODE *root;
   int ret;

   N=atoi(argv[1]);
   M=atoi(argv[2]);

   init_eikona(N,M,0,0,0);
   xin=build_image(N,M);
   xout=build_image(N,M);
   fromdisk(argv[3],xin,0,0,N,M);
   thres(xin,xout,atoi(argv[4]),0,0,N,M);
   init_graphics("VRES16COLOR");
   bindisp(xout, 16,0,0,0,0,N,M);
   ret=image_to_quadtree(xout,0,0,N,M,&root);
   printf("\n ");
   printf("image_to_quadtree= %d", ret);

   clear(xout,0,0,0,N,M);
   ret=quadtree_to_image(root,xout,0,0,N,M);
   printf(" quadtree_to_image= %d", ret);

   bindisp(xout, 16,0,M,0,0,N,M);
   getch();
   reset_graphics();
   return(0);
}
```

The syntax of the command line for this program is:

```
<program name> <Rows> <Columns> <Input File> <Threshold>
```

The subroutines for creating pyramids from this chapter can be tested using the program bellow:

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc,char *argv[])
{
char filename[80];
int N=256,M=256;
int i,k,T,flag,s,n,nn,m,l;
image im,im2;
pyramid p,e;
if (argc!=4) {printf("\nUSAGE : Pyramid filename[.PIC] algorithm(1/2/3)
  thres");exit(1);}
strcpy(filename,argv[1]);
if (strchr(argv[1],'.')==NULL) strcat(filename,".pic");
flag=atoi(argv[2]);

init_eikona(256,256,-1,-1,-1);
im=build_image(256,256);
im2=build_image(256,256);
fromdisk(filename,im2,0,0,N,M);
sobel(im2,im,0,0,N,M);
thres(im,im,atoi(argv[3]),0,0,N,M);
init_graphics("VRES16COLOR");
bindisp(im, 16,0,0,0,0,N,M);
getch();
switch(flag){
case 1  :
      pyramid_1(im,im2,N,M);
      bindisp(im2, 16,0,0,0,0,N,M);
      getch();

      mfreeuc2(im2,N);
```

```
      break;
case 2  :p=pyramid_alloc(N);
     if (p==NULL) {puts("out of memory");exit(1);}
     pyramid_fill(im,p,N);
     view_pyramid(p,16,N);
     pyramid_free(p,N);
     break;
case 3  :printf("\rGive start level and threshold for edge
  detection : ");
     scanf("%d %d",&k,&T);
     p=pyramid_alloc(N);
     if (p==NULL) {puts("out of memory");exit(1);}
     pyramid_fill(im,p,N);
     view_pyramid(p,16,N);
     e=pyramid_alloc(N);
     if (e==NULL) {puts("out of memory");exit(1);}
     pyramid_edge_detection(p,e,N,k,T);

     for(n=0,nn=1;nn<N;n++) nn *=2;

     for(k=0; k<=n;k++)
      {
        for (i=0,s=1;i<k;i++) s*=2;
        for (m=0;m<s;m++)
         for (l=0;l<s;l++)
          if(e[k][m][l]>0) e[k][m][l]=1;}
     view_pyramid(e,16,N);
     pyramid_free(p,N);
     pyramid_free(e,N);
}
     mfreeuc2(im,N);
     reset_graphics();
     return(0);
}
```

The syntax of the command line is self-explanatory and can be given by writing:

```
<program name>
```

The three options given by the program are:
1: Create Pyramid.
2: Create Pyramid using OR operators.
3: Perform edge detection using a pyramid structure.

The subroutines for calculating moment and boundary related features from this chapter can be tested using the program bellow:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include <math.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc,char *argv[])
{
char filename[80],c=0;
int N1=0, M1=0, N2=256,M2=256;
int x,y;
image im,im1;
LIST l;  /* list of pixels */
float area,xmean,ymean,orientation,eccentricity1,eccentricity2,spread;
float perimeter,bending_energy,norm_bending_en,compactness,norm_compactness;
float moment;
float parea,pKBx,pKBy,pdeg,pvolume;
if (argc!=3) {printf("\nUSAGE : Features filename[.PIC] threshold");exit(1);}
strcpy(filename,argv[1]);
if (strchr(argv[1],'.')==NULL) strcat(filename,".pic");
init_eikona(256,256,-1,-1,-1);
im=build_image(256,256);
im1=build_image(256,256);
fromdisk(filename,im1,N1,M1,N2,M2);
thres(im1,im,atoi(argv[2]),N1,M1,N2,M2);
    init_graphics("VRES16COLOR");
    bindisp(im, 16,0,0,N1,M1,N2,M2);
    getch();
    reset_graphics();
    printf("\n ");
findch(im1,atoi(argv[2]),&parea,&pKBx,&pKBy,&pdeg,&pvolume, N1,M1,N2,M2);
calc_features_1(im,N1,M1,N2,M2,&area,&xmean,&ymean,&orientation,
  &eccentricity1,&eccentricity2,&spread);
l.start = l.end = getnode();
turtle_follow(im,&l,N1,M1,N2,M2);
```

```
calc_features_2(l,&perimeter,&bending_energy,&norm_bending_en,&compactness,
&norm_compactness,area);
freelist(l);

printf("\n Area=%f m_x=%f m_y=%f theta=%f volume=%f",
    parea,pKBx,pKBy,pdeg,pvolume);

printf("\nCenter of gravity : (%.1f , %.1f)",xmean,ymean);
printf("\nArea : %.0f (pixels)",area);
printf("\nOrientation : %.1f degrees",orientation*180/3.14159);
printf("\nEccentricity (1st formula) : %.1f",eccentricity1);
printf("\nEccentricity (2nd formula) : %.1f",eccentricity2);
printf("\nSpread : %.1f ",spread);
printf("\n\nPerimeter : %.1f unitys (pixel sides)",perimeter);
printf("\nBending Energy : %.3f       Normalized : %.3f",bending_energy,
norm_bending_en);
printf("\nCompactness : %.3f       Normalized : %.3f",compactness,
norm_compactness);
do{
 printf("\nCalculate moments ? (M[oment]/C[entral moment]/Q[uit]) : ");
 c=tolower(getche());
 if (c!='q')
  {
  printf("\nGive x,y (for moment m(x,y)) : ");scanf("%d %d",&x,&y);
  moment=find_moment(im,x,y,N1,M1,N2,M2,(c=='c')?xmean:0,(c=='c')?ymean:0);
  printf("%s%d%d = %.1f",(c=='c')?"":"m",x,y,moment);
  }
}
while(c!='q');
mfreeuc2(im,N2);
return(0);
}
```

The syntax of the command line is self-explanatory and can be given by writing:

```
<program name>
```

The subroutines implementing thinning algorithms from this chapter can be tested using the program bellow:

```
#include <string.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <graph.h>
#include "eikona.h"

int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
int KMAX, FKMAX, IKMAX;

main(int argc,char *argv[])
{
char filename[80];
int N=256,M=256;
int flag;
image im1,im2;
strcpy(filename,argv[1]);
if (strchr(argv[1],'.')==NULL) strcat(filename,".pic");
flag=atoi(argv[2]);
init_eikona(256,256,-1,-1,-1);
im1=build_image(N,M);
im2=build_image(N,M);
fromdisk(filename,im1,0,0,N,M);
clear(im2,0,0,0,N,M);
sobel(im1,im2,0,0,N,M);
thres(im2,im2,atoi(argv[3]),0,0,N,M);
    init_graphics("VRES16COLOR");
    bindisp(im2, 16,0,0,0,0,N,M);

switch(flag){
case 1  : thin_1(im2,0,0,N,M);
          break;
default : thin_2(im2,0,0,N,M);
          break;
}
bindisp(im2, 16,0,N,0,0,N,M);
getch();
reset_graphics();

mfreeuc2(im1,N);
mfreeuc2(im2,N);
return(0);
}
```

The syntax of the command line for this program is:

```
<program name> <input file> <choice> <threshold>
```

where the `<input file>` is assumed to have an extension: `.pic`, the `<choice>` selects one pass or two pass thinning and `<threshold>` is used for the binary transformation of the image.

References

[PIT90] I.Pitas, A.N.Venetsanopoulos, *Nonlinear digital filters: Principles and applications*, Kluwer Academic, 1990.

[PIT93] I.Pitas *Digital image processing algorithms*, Prentice Hall, 1993.

[LEV85] M.D.Levine, *Vision in man and machine*, McGraw-Hill, 1985.

[GON87] R.C.Gonzalez, P.Wintz, *Digital image processing*, Addison-Wesley, 1987.

17

Appendix

17.1 Contents of EIKONA.H

```
/*--------------------------- EIKONA.H -----------------------------------*/
/*EIKONA v.3.2 Last revision  20-3-97,                                     */
/*-------------------------------------------------------------------------*/

#ifndef _EIKONA_H_
#define _EIKONA_H_

#ifdef _WINDOWS
#include <windows.h>
#else
#define far
#define  huge
#define _fmalloc malloc
#define _fcalloc calloc
#define _ffree free
#define _fmemcpy memcpy
#endif

typedef unsigned char far* far* image;
typedef float far* far* matrix;
typedef int far* far* imatrix;
typedef float far* vector;

/* Type definitions for edge following algorithms */
typedef struct pixel {int row; int column; long cost; int help;} CPIXEL;
```

```
/*typedef struct pixel CPIXEL;*/
typedef struct node { CPIXEL info; struct node far* link;} NODE;
/*typedef struct node NODE;*/


/* Type definitions for chain codes */
typedef unsigned char byte;
typedef struct CHNODE {unsigned char num; struct CHNODE far *next;} CHNODE;
typedef struct CHAIN {CHNODE far *start; CHNODE far *end;} CHAIN;

/* Type definitions for polygon approximations */
typedef struct PIXEL {int x; int y;} PIXEL;
typedef struct LNODE {PIXEL num; struct LNODE far *next;} LNODE;
typedef struct LIST {LNODE far *start; LNODE far *end;} LIST;

/* Type definitions for quadtrees */
typedef struct QNODE { unsigned char color;
              struct QNODE far *father;
              struct QNODE far *quad[4];
             } QNODE;
typedef image far* pyramid;

/* Type definitions for Huffman coding */
typedef struct symbol far* SPOINTER;

struct symbol {
           float prob;
           SPOINTER comp[2];
          };


/* Type definitions for LZW coding */
typedef struct llink far*LPOINTER;
struct llink {
 int n;
 LPOINTER next;
};

/*-------------------------------------------------------------------------*/
/* Global variables used in EIKONA, internal use only                      */
/* Remove extern for DLL !!!! */
extern int NMAX, MMAX, INMAX, IMMAX, FNMAX, FMMAX ,NWMAX, MWMAX, NSIG;
extern int KMAX, FKMAX, IKMAX;
```

```
int improinit(void);


#ifdef _WINDOWS
/*-------------------------------------------------------------------------*/
/*Windows additions. Subroutines in winmem30.c  */
image  win_build_image(HANDLE far * far*,int,int);
void win_freeuc2(HANDLE far * far*,int,int);
matrix win_build_matrix(HANDLE far * far*,int,int);
void win_freef2(HANDLE far * far*,int,int);
imatrix win_build_imatrix(HANDLE far * far *,int,int);
void win_freei2(HANDLE far * far*,int,int);

/*-------------------------------------------------------------------------*/
/*Windows additions. Subroutines in windisp3.c */

int wincolorimdisp(HWND,LOGPALETTE *,PBITMAPINFO,PBITMAPINFO,unsigned
long,int,image,image,image,int,int,int,int,int,int,int);
int winbwimdisp(HWND,LOGPALETTE *,PBITMAPINFO,unsigned
long,image,int,int,int,int,int,int,int);
/* int refresh(int buffer, int display_type); */ /* change - it is in eikona2D.h
  now */
int create_disp_bmp(HWND,PBITMAPINFO *,PBITMAPINFO *,unsigned long *);
LOGPALETTE *alloc_logpalette(HPALETTE *,unsigned long);
/*HPALETTE *hpal;
unsigned long PALETTESIZE;*/
#define SIGNAL_AXIS_Y_SIZE 162
#define SIGNAL_AXIS_X_SIZE 500
#define SIGNAL_WINDOW_Y_SIZE 200
#define SIGNAL_WINDOW_X_SIZE 570
#define SY0 170
#define SX0 50
#define COEFF_MINY_MAXY 0.3

struct SIGNAL
  { int x;
    int y;
    float far* valuesx;
    float far* valuesy;
    int numofvalues;
    int orderinc;
    float startx;
    float endx;
```

```
    int far* values;
    int numlabelsx;
    int poslabelsx[10];
    long divx;
    long firstlabelx;
    int numlabelsy;
    int poslabelsy[10];
    long divy;
    long firstlabely;
    long coeffy;
    long coeffx;
  };

int computesignal(struct SIGNAL far*, float, float);
int displaysignal(HDC, struct SIGNAL, int);
#endif

/*-------------------------------------------------------------------------*/
/* Subroutines in build30.c                                                */

int init_eikona(int,int,int,int,int);
int set_parameters(int, int,  int, int,  int,  int, int,  int, int);
int get_parameters(int*,int*, int*,int*, int*, int*,int*, int*,int*);
image  build_image(int,int);
vector build_vector(int);
matrix build_matrix(int,int);
unsigned char far* far* matuc2(unsigned int, unsigned int); /* internal use*/
float far* far* matf2(unsigned int, unsigned int);   /* internal use only */
void mfreeuc2(unsigned char far* far*,unsigned int); /* internal use only */
void mfreef2(float far* far*,unsigned int);          /* internal use only */
int far* far* matin2(unsigned int,unsigned int);
void mfreein2(int far* far*,unsigned int);
int far* far* build_imatrix(int,int);
float far* matf1(int);
void mfreef1(float far*);
signed char far* far* matsc2(int , int );
void mfreesc2(signed char far* far* , int);


/*-------------------------------------------------------------------------*/
/* Subroutines in bas30.c                                                  */

int add(image,image,image, int,int,int,int);
int addc(image,image, int, int,int,int,int);
```

```
int and(image,image,image, int,int,int,int);
int bin(image, int,int,int,int);
int clear(image, int, int,int,int,int);
int clearmatrix(matrix, float, int,int,int,int);
int clip(image,image, int,int, int,int,int,int);
int copy(image,image, int,int, int,int,int,int);
int copymatrix(matrix, matrix, int,int, int,int,int,int);
int diff(image,image,image, int,int,int,int);
int image2matrix(image, matrix, int,int, int,int,int,int);
int matrix2image(matrix, image, int,int, int,int,int,int);
int mix(image,image,image, float,float, int,int,int,int);
int multc(image,image, float, int,int,int,int);
int neg(image,image, int,int,int,int);
int nltran(image,image, vector, int,int,int,int);
int not(image,image, int,int,int,int);
int or(image,image,image, int,int,int,int);
int overmix(image,image,image,image, int,int,int,int);
int pixr(image, int,int);
int pixw(image, int,int, int);
int rotim(image,image, float, int,int, int,int, int,int,int,int);
int thres(image,image, int, int,int,int,int);
int xor(image,image,image, int,int,int,int);

/*--------------------------------------------------------------------*/
/* Subroutines in io30.c                                              */

int dump(image, char *, int,int,int,int);
int dump_signal(vector, char *);
int fromdisk(char *, image, int,int, int,int);
int printim(image, char *, int,int,int,int);
int todisk(image, char *, int,int,int,int);
int loadmatrix(char *, matrix, int,int, int,int);
int savematrix(matrix, char *, int,int,int,int);
int print_hp(image, char*,int,int,int,int,int,int,int);
int totga(image,image,image,char*,int,int,int,int,unsigned char);
int fprint_chain(char *,int,int,CHAIN);
int fprint_list(char *,int,int,LIST);
int thumbnail(image,image,image,image,image,image,int,int,int,int,int,int,int);
int dump_matrix_hist(vector [],char*,int);
int dumpmatrix(matrix, char*,int,int,int,int);
int topostscript(image, image, image, char *, int, int, int, int, int, int);

int gettiffinfo(char *, unsigned long *, unsigned long *, unsigned char *);
int fromtiff(char *, image, image, image);
```

```
int totiff(image, image, image, char *, unsigned long, unsigned short,
  unsigned short, unsigned short, int, int, int, int);

/*-------------------------------------------------------------------------*/
/* Subroutines in iooth.c                                                  */

int frombmp(char * ,image ,image ,image );
int fromtga(char * ,image, image ,image ,image );
int fromgif(char * ,image ,image ,image );
int fromjpeg (char * ,image, image ,image ,image , int *, int *);
int tojpeg (char * ,image ,image ,image ,int ,int ,int ,int ,int ,int ,int);

/* Additions for TGA,JPEG format-----------------------------------------*/
typedef struct FORMAT_COLORTYPE_SIZE
{
   int rows;
   int cols;
   int colortype;
} FORMAT_COLORTYPE_SIZE;
int ReadTGAHead(char *fname,FORMAT_COLORTYPE_SIZE *tga_coltype_size);
int ReadJPEGHead(char *filename,FORMAT_COLORTYPE_SIZE *jpeg_coltype_size);
int ReadBMPHead(char *fname,FORMAT_COLORTYPE_SIZE *bmp_coltype_size);
int ReadGIFHead(char *fname,FORMAT_COLORTYPE_SIZE *gif_coltype_size);
/* My Additions for TGA,JPEG format---------------------------------------*/
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
/* Subroutines in disp30.c                                                 */

int getvideomode(void);                      /* internal use only */

int bindisp(image, int, int,int, int,int,int,int);
int clearscreen(void);
int imdisp(image, int, int,int, int,int,int,int);
int init_graphics(char *);
int overdisp(image,image, int, int,int, int,int,int,int);
int reset_graphics(void);
int sigshow(vector, int, int,int, int,int);

int coeff(vector, int);
int two_d_coeff(matrix, int,int);

/* Subroutines in assemply                                                 */
```

```
int disp13(image, int, int,int, int,int,int,int);
int dispb13(image, int, int,int, int,int,int,int);
int disp10(image, int, int,int, int,int,int,int, int);
int dispb10(image, int, int,int, int,int,int,int, int);

int vgadisp(image,int,int,int,int,int,int);
int video_mode(unsigned char);
int extvgadisp(image,int,int,int,int,int,int,int);
int extdisp(image,int,int,int,int,int,int,int);
int extbindisp(image,unsigned char,int,int,int,int,int,int,int);
unsigned long far* build_palette();
int load_palette(unsigned long far*);
int default_palette(unsigned long far*,int);
int create_palette(image,image,image,unsigned long far*,int,int,int,int);
int rgbtovga(image,image,image, image, unsigned long far*, int,int,int,int,int);

int view_pyramid(pyramid,int,int);

/*-------------------------------------------------------------------------*/
/* Subroutines in color30.c                                                */

int cieRGB_to_XYZ(image,image,image,image,image,image,int,int,int,int);
int XYZ_to_cieRGB(image,image,image,image,image,image,int,int,int,int);
int XYZ_to_ciexyY(image,image,image,matrix,matrix,matrix,int,int,int,int);
int ciexyY_to_XYZ(matrix,matrix,matrix,image,image,image,int,int,int,int);
int XYZ_to_cieuvY(image,image,image,matrix,matrix,matrix,int,int,int,int);
int cieuvY_to_XYZ(matrix,matrix,matrix,image,image,image,int,int,int,int);
int XYZ_to_cieUVW(image,image,image,matrix,matrix,matrix,int,int,int,int);
int cieUVW_to_XYZ(matrix,matrix,matrix,image,image,image,int,int,int,int);
int XYZ_to_cieUsVsWs(image,image,image,matrix,matrix,matrix,int,int,int,int);
int cieUsVsWs_to_XYZ(matrix,matrix,matrix,image,image,image,int,int,int,int);
int XYZ_to_SthetaWs(image,image,image,matrix,matrix,matrix,int,int,int,int);
int SthetaWs_to_XYZ(matrix,matrix,matrix,image,image,image,int,int,int,int);
int XYZ_to_Lsasbs(image,image,image,matrix,matrix,matrix,double,double,double,
  int,int,int,int);
int Lsasbs_to_XYZ(matrix,matrix,matrix,image,image,image,double,double,double,
  int,int,int,int);
int XYZ_to_Lsusvs(image,image,image,matrix,matrix,matrix,double,double,double,
  int,int,int,int);
int Lsusvs_to_XYZ(matrix,matrix,matrix,image,image,image,double,double,double,
  int,int,int,int);
int ntscRGB_to_XYZ(image,image,image,image,image,image,int,int,int,int);
int XYZ_to_ntscRGB(image,image,image,image,image,image,int,int,int,int);
int ntscRGB_to_YIQ(image,image,image,matrix,matrix,matrix,int,int,int,int);
```

```
int YIQ_to_ntscRGB(matrix,matrix,matrix,image,image,image,int,int,int,int);
int RGB_to_CMY(image,image,image,image,image,image,int,int,int,int);
int CMY_to_RGB(image,image,image,image,image,image,int,int,int,int);
int RGB_to_CMYK(image,image,image,image,image,image,image,int,int,int,int);
int CMYK_to_RGB(image,image,image,image,image,image,image,int,int,int,int);
int RGB_to_HSI(image,image,image,matrix,matrix,matrix,int,int,int,int);
int HSI_to_RGB(matrix,matrix,matrix,image,image,image,int,int,int,int);
int RGB_to_HSV(image,image,image,matrix,matrix,matrix,int,int,int,int);
int HSV_to_RGB(matrix,matrix,matrix,image,image,image,int,int,int,int);
int RGB_to_HLS(image,image,image,matrix,matrix,matrix,int,int,int,int);
int HLS_to_RGB(matrix,matrix,matrix,image,image,image,int,int,int,int);
int Points_cieRGB_to_XYZ(double,double,double,double*,double*,double*);
int Points_XYZ_to_Lab(double,double,double,double*,double*,double*,double,
  double,double);
int Points_RGB_to_HLS(double,double,double,double*,double*,double*);

static double value(double,double,double);
double atan3(double,double);

int YUV_to_RGB(image, image, image, image, image, image,int,int);
int RGB_to_YUV(image, image, image, image, image, image,int,int);
int RGB_to_Y(image , image , image , image , int ,int);
int UserDefinedTrans(matrix,matrix,matrix,matrix,matrix,matrix,matrix,
int,int,int,int,int,int,int,int);

/*-------------------------------------------------------------------------*/


/* Subroutines in trans30.c                                                */

int arpsd(image,matrix,int,int,int,int,int);
int blackman_tukey_psd(image,matrix,int,int,int,int);
int convolution(matrix,matrix, matrix, int,int, int,int,int,int);
int correlation(matrix,matrix, matrix, int,int, int,int,int,int);
int fft1d(float *, float *,int,float *, float *); /* internal use */
int rcfft(matrix,matrix, matrix,matrix, int, int,int, int,int,int,int);
int vrfft(matrix,matrix, matrix,matrix, int, int,int, int,int,int,int);
int fft2d(matrix,matrix, matrix,matrix, int, int,int, int,int,int,int);
int fft2image(matrix,matrix, image, int,int, int,int,int,int);
int magnphase2reim(matrix,matrix, matrix,matrix, int,int, int,int,int,int);
int pt(matrix,matrix,int,int);
int ptfft(matrix,matrix,int,int,int);
int reim2magnphase(matrix,matrix, matrix,matrix, int,int, int,int,int,int);
int windowcorrel(matrix,matrix,int,int,int,int);
```

```
int fft_IO_mc(FILE *, FILE *,int,int,int,int);
int fft_IO_trans(FILE *,FILE *,int,int);
int real_2d_fft(matrix,int,int);
int refft2images(matrix,matrix,int,int);
int fft(float *,float*,int,float *, float *,int); /* internal use only */
int init(vector,vector,int);
int DCT(matrix,vector,vector,int,int);
int IDCT(matrix,vector,vector,int,int);
int fftc(vector,vector,int,int); /*internal use*/

/*-------------------------------------------------------------------------*/
/* Subroutines in filt30.c                                                 */

int cdfhist(vector, vector);
int conv(image,image, matrix, int,int, int,int,int,int);
int decim(image,image, int, int,int, int,int,int,int);
int dither(image,image,int,int,int,int,int);
int halftone(image,image, int, int,int,int,int);
int hist(image, vector, int,int,int,int);
int histeq(image,image, vector, int,int,int,int);
int interpolation(image,image,float,int,int,int,int,int,int,int);
int L2_error_norm(image,image,float*,int,int,int,int);
int L2_error_ratio(image,image,image,image,float*,int,int,int,int);
int l_filter(image,image, vector, int,int, int,int,int,int);
int matrixcdfhist(vector, int);
int matrixhist(matrix, vector [],int,int,int,int,int);
int maxi(image,image, int,int, int,int,int,int);
int median(image,image, int,int, int,int,int,int);
int mini(image,image, int,int, int,int,int,int);
int movav(image,image, int,int, int,int,int,int);
int noise_add_gauss(image,image, float, int,int,int,int);
int noise_add_laplace(image,image, float, int,int,int,int);
int noise_add_uni(image,image, float, int,int,int,int);
int noise_imp(image,image, float, int,int, int,int,int,int);
int noise_mult_gauss(image,image, float, int,int,int,int);
int noise_mult_uni(image,image, float, int,int,int,int);
int order(image,image, int, int,int, int,int,int,int);
float rand0(int *);        /* internal use only */
int sort(int far *, int); /* internal use only */
int searchmathistval(float,vector [], int, int);
int sharp(image,image, int,int,int,int);
int snr(image,image,float *,int,int,int,int);
int zoom(image,image, int, int,int, int,int,int,int);
```

```
int overlap_add(matrix,matrix,matrix,int,int,int,int,int,int,int,int);
int overlap_add_c(matrix,matrix,matrix,matrix,matrix,matrix,int,int,int,int,
  int,int,int,int);
int overlap_save(matrix,matrix,matrix,int,int,int,int,int,int,int,int);
int overlap_save_c(matrix,matrix,matrix,matrix,matrix,matrix,int,int,int,int,
  int,int,int,int);
int wienerfrequency(matrix,matrix,matrix,matrix,matrix,float,int,int);

/*--------------------------------------------------------------------*/
/* Subroutines in nlfilt30.c                                          */
int adapt_l_filter(image,image,image,vector,float,int,int,int,int,int,int);
int adapt_wei_median(image,image,int,float,int,int,int,int,int,int);
int a_trimmed_filt(image,image,int,int,int,int,int,int,int);
int bclose(image,image,image,int,int,int,int,int,int);
int bdilate(image,image,int,int,int,int,int);
int berode(image,image,int,int,int,int,int);
int bopen(image,image,image,int,int,int,int,int,int);
int dw_mtm(image,image,int,int,int,int,int,int,int,int,int,int);
int harm_filt(image,image,matrix,int,int,int,int,int,int);
int homom_filt(image,image,matrix,int,int,int,int,int,int);
int imclose(image,image,image,image,int,int,int,int,int,int);
int imdilate(image,image, image, int,int, int,int,int,int);
int imerode(image,image, image, int,int, int,int,int,int);
int imopen(image,image,image,image,int,int,int,int,int,int);
int Lp_filt(image,image,float,matrix,int,int,int,int,int,int);
int local_adapt_filt(image,image,float,int,int,int,int,int,int);
int max_median(image,image,int,int,int,int,int,int);
int medhybr(image,image,int,int,int,int,int,int);
int mnn(image,image,int,int,int,int,int,int,int);
int msd(image,image,image,image,int,int,int,int,int,int);
int mtm(image,image,int,int,int,int,int,int,int);
int mult_adapt_filt(image,image,float,int,int,int,int,int,int);
int multistage_median(image,image,int,int,int,int,int,int);
int optimal_L_filter(image,vector,int,int,int,int,int,int);
int sam(image,image,float,float,float,float,float,int,
        int,float,float,int,int,int,int,int,int,int,int,int,int);
int rec_median(image,image,int,int,int,int,int,int);
int run_max(image,image,image,int,int,int,int,int,int);
int run_median(image,image,int,int,int,int,int,int);
int run_min(image,image,image,int,int,int,int,int,int);
int sep_median(image,image,image,int,int,int,int,int,int);
int skeleton(image,image,image,image,int,int,int,int,int);
int top_hat(image,image,image,int,int,int,int,int,int);
int wei_median(image,image,image,int,int,int,int,int,int);
```

```
/*-------------------------------------------------------------------------*/
/* Subroutines in code30.c                                                 */

int treeaccess(SPOINTER,int *,char *,struct symbol far*,
        unsigned char far* far *);
int hufcod(FILE * ,image , FILE *, int , int , int );
int sendToOutput(FILE *,char far*,int *,unsigned char *);
int huftree(vector,SPOINTER *,struct symbol far**,int);
int initialize(SPOINTER *,struct symbol far**, vector,int);
int find2min(SPOINTER *,int *,int);
SPOINTER merge(SPOINTER *,int *);
int decode_huffman(FILE *,image,SPOINTER,struct symbol far*,int,int);
int reconstr_tree(FILE *,struct symbol far**,SPOINTER*,int);

int lzw(image,FILE*,int,int, int, int);
int init_lzw(int,int,int*,int far*,int far*,int*,int*,int*,LPOINTER*);
int inT(int far*,int far*,int,unsigned char,int,LPOINTER*);
int AddToTable(int far*,int,int,int far*,int,int*,int*,int*,int,LPOINTER*);
int send2output(FILE*,int,int,int*,unsigned char*,
         unsigned int*,unsigned char*);
int invlzw(FILE*,image,int,int,int,int);
int init_invlzw(int,int,int*,int far* far*,int*,int*);
int GetNextCode(int*,int,FILE*,unsigned char*,int*,
        int,int*,int*,unsigned int*,unsigned char*);
int AddEntry(int far* far*,int,int,int*,int*,int*,int);
int WriteString(int far* far*,int,int*,int*,int,int,image);


int arcoefficients(image,int,int,int,int,int,float [],float *);
int arsolve(float far* far*,float *,float *, int,int); /*internal use*/
float corsample(image,image,int,int,int,int); /*internal use*/

/*-------------------------------------------------------------------------*/
/* Subroutines in edge30.c                                                 */

int compass(image,image, int, int,int,int,int);
int laplace(image,image, int,int,int,int);
int line_detect(image,image, int, int,int,int,int);
int point_detect(image,image, int,int,int,int);
int prewitt(image,image, int,int,int,int);
int range(image,image, int,int, int,int,int,int);
int roberts(image,image, int,int,int,int);
int sobel(image,image, int,int,int,int);
```

```
int hough(image,imatrix,int,int,int,int,float *, float *);
int thres_par(int far* far*,int,int,int);
int ihough(image,imatrix,int,int,int,int,float *,float *);
int look_up_table(float *,float *,int);
int edge_follow(image,image,int,int,int,int,int,int,int,int);
int initialize_index(CPIXEL*);
int edge_dynamic_prog(image,CPIXEL,CPIXEL,int,NODE far**);
int calculate_Fx(image,CPIXEL,int,CPIXEL*,CPIXEL);
int sort_pixel(CPIXEL*);
int stack2image(NODE far**,image);
int find_direction(CPIXEL,CPIXEL);
int modified_heuristic_edge_search(image,CPIXEL,CPIXEL,int,NODE far**);
int find_successors(image,CPIXEL,int,CPIXEL*,CPIXEL);

/*-------------------------------------------------------------------------*/
/* Subroutines in list30.c                                                 */

int push(NODE far**,CPIXEL);
CPIXEL pop(NODE far**);

CHNODE far *getchnode(void);
CHNODE far *addchnode(int, CHNODE far *);
CHNODE far *searchch(byte, CHAIN);
void deletech(CHNODE far *, CHNODE far *);
LNODE far *getnode(void);
LNODE far *add_list(PIXEL, LNODE far *);
void deletel(LNODE far*, LNODE far*);
void freelist(LIST);
void freechain(CHAIN);
int list_decode(image,LIST);
int list_connect(image,LIST);

/*-------------------------------------------------------------------------*/
/* Subroutines in region30.c                                               */

int count(image,int *,vector,int,int,int,int);
int grass_fire(image,int,int, float far *,int,int,int,int);
int mode_filter(image,image, int,int, int,int,int,int);
int segm(image,image, int, int,int,int,int);
int grass_label(image,image,int *,vector,int,int,int,int,int);
int grass(image,image,int,int,int, float far*,int,int,int,int);/*internal use */
int region_grow(image,image,int,int,int,int,int,int);
int region_merge(image,image,int,int,int,int,int,int);
int region_split(image,image,int,int,int,int,unsigned long *,int);
```

```
int test_homogeneity(image,int,int,int,int,int);
int region_split_merge(image,image,unsigned char far*,
               int,int,int,int,int *,int,int);
int region_split_merge_recursive(image,image,unsigned char far*,
    int,int,int,int,int,int,int,int,int *,int,int);
int rmerge(image,image,unsigned char far*,int,int,int,int,int,int,int,int,int *,
  int);
int region_split_merge_c(image,image,unsigned char far*,int far* far*,
               int,int,int,int,int *,int,int);
int region_split_merge_c_recursive(image,image,unsigned char far*,
    int far* far*,int,int,int,int,int,int,int,int,int *,int,int);

int rmerge_c(image,image,unsigned char far*, int far* far*,
                     int,int,int,int,int,int,int,int,int *,int);

float meanvalu(vector);
float varianc(vector);
float skewnes(vector);
float kurtosi(vector);
float entrop(vector);

int difhist(image,vector,int,int,int,int,int,int);
float acs(vector);
float entropy_d(vector);
float m_d(vector);
float c_d(vector);

int runlen(image,imatrix,int,int,int,int,int);

double T_R(imatrix);
double sre(imatrix);
double lre(imatrix);
double gld(imatrix);
double rld(imatrix);
double rp(imatrix);

int glcmarr(image,matrix,int,int,int,int,int,int);
float c_glcm(matrix);
float asm_glcm(matrix);
float idm_glcm(matrix);
float entropy_glcm(matrix);
float corr_glcm(matrix,float *,float *);

int psRtheta(matrix,vector,vector,int,int,int);
```

```
/*-------------------------------------------------------------------------*/
/* Subroutines in shape30.c                                                */

int findch(image, int, float*,float*,float*,float*,float*, int,int,int,int);

int chain_code(image, CHAIN *, int *, int *,long,int,int,int,int);
int chain_decode(image,CHAIN,int,int,int,int,int,int);
CHAIN compress_chain(CHAIN);
CHAIN decompress_chain(CHAIN);

int curve_split(PIXEL far *, LIST *, int, double);
int curve_merge(PIXEL far *, LIST *, int, double);
double dist(int,int,int,int,int,int); /* internal use */
int create_curve(image, PIXEL far *, int *, int,int,int,int,int);

int image2fd(image,vector,vector,int*,int,int,int,int,int);
int fd2image(vector,vector,image,int,int,int,int,int,int);

int image_to_quadtree(image,int,int,int,int, QNODE far * far *);
int construct_quadtree(image, QNODE far * far *, unsigned char, unsigned char *,
  int,int,int,int);
int find_level(QNODE far *,int *);
int quadtree_to_image(QNODE far *,image,int,int,int,int);
int make_image(QNODE far *,image,int,int,int,int,int);
QNODE far *create_node(QNODE far *, unsigned char, unsigned char);

int pyramid_1(image,image,int,int);
pyramid pyramid_alloc(int);
int pyramid_free(pyramid,int);
int pyramid_fill(image,pyramid,int);
int pyramid_edge_detection(pyramid,pyramid,int,int,int);
int edge_detector(image,int,int);
int refine(pyramid,pyramid,int,int,int,int,int);

float find_moment(image,int,int,int,int,int,int,float,float);
int calc_features_1(image,int,int,int,int,float *,float *,float *,float *,
            float *,float *,float *);
int turtle_follow(image,LIST *,int,int,int,int);
float find_perimeter(LIST);
float find_bending_energy_sum(LIST);
int calc_features_2(LIST,float *,float *,float *,float *,float *, float);
int thin_1(image,int,int,int,int);
int thin_2(image,int,int,int,int);
```

```
/*-------------------------------------------------------------------------*/
/* Subroutines in arts30.c                                                 */

typedef struct FINDIMAG
{
   int poz1;
   int poz2;
   int coor1_X;
   int coor1_Y;
   int coor2_X;
   int coor2_Y;
} FINDIMAG;

typedef struct unit {
  int oriz ;
  int vert ;
} unit;

int Re_Order_misimage(FINDIMAG far* misimage,int No_pairs);
int Check_misimage(FINDIMAG far* misimage,int No_pairs,int No_Im_X,int No_Im_Y);
int refine_MM (image far*,int,FINDIMAG far*,int,int,int,int,int,int);
int Mosaic(int,int,image far*,int,int,int,int,int,int,int,FINDIMAG far*,
int,int,int,int,int*,int*,int*,int*,unit far* far*);
int WriteTheImage(int,int,int,int,image,image,image,image,image,image,
int,int,int,unit far* far*,int,int);
int registration(double far*,double far*,double far*,double far*,int,int,int,
  int,int,image,
int,int,int,int,image,int,int,int,int,image,int,double);
int image_merge(image,image,int,int,image,image,int,int,int,int);
int image_subtract(image,image,int,int,image,image,int,int,int,int);
int adjust(image,image,image,image,int,int,int,int,int,int);
int registration_new(double far*,double far*,double far*,double far*,int,int,
  int,int,int,
image,image,int,int,image,image,int,int,image*,image*,int*,int*,int,
  int,int*,int*,int);

/*-------------------------------------------------------------------------*/
/* Subroutines in water30.c                                                */
typedef struct
{
long rseed; /*seed of RND generator*/
   int wpow;   /* Embedding level */
} KEY;
```

```
int add_wtr_watermark(KEY *,image,image,image,int,int,int,image,image,image);
int detect_wtr_watermark(KEY *,image,image,image,int,int,int,double*,float*);

int Read_Signature(image, image, image, image,image,image,long,int,int,int,
  int,int);
int Add_Signature(image,image,image,image,image,image, image,image,image,long,
  int,int ,int ,int,int);
int Remove_Signature(image,image,image,image,image,image, image,image,image,
  long,int,int ,int ,int,int);


#endif
```

17.2 Basic error messages

```
-1 : NULL image, matrix pointer
-2 : NULL vector, window pointer
-3 : NULL misimage pointer for manual mosaic
-4 : NULL basic type pointer
-9, -10 :  Can not allocate internal buffer
-12 : Invalid number of pairs for manually mosaic
-13 : Image contains more than 255 colors
-21 : Invalid working area parameter
-22 : Entire source buffer cannot be stored on destination buffer
-23 : Entire image cannot be displayed
-24 : Invalid window size
-25 : Invalid order statistic parameter
-29 : Error in reading/writting JPEG files
-30 : Error in reading TGA files
-31 : Error in reading BMP files
-32 : Error in reading GIF files
-33 : Given size is not a power of 2
-34 : Image or matrix is not square
-36 : Destination buffer is identical to source buffer
-40 : Too few valid feature points  for refined registration
-41 : Invalid clip parameters, cmin>cmax
-44 : Invalid coordinates, i or j
-50 : Invalid size parameter, M>NSIG
-60 : Invalid number of gray levels
-62 : Invalid number of halftones
-70 : Invalid angle
-71 : Matrix cannot be inverted
-72 : The filter window is not square.
-80 : Invalid noise parameter
-81 : noise variance negative.
-82 : The large window in the dw_mtm filter is smaller than the
  small window.
-84 : The sum of the weights in the weighted median filter
  exceed the maximum allowable limit.
-85 : adaptation coefficient is less than zero.
-86 : The sum of the filter coefficients is zero.
-87 : The denominator or the numerator of the snr or L2 error
  ratio is zero
-88 : The power p of the Lp filter is zero.
-89 : Invalid intermediate results
-90 : Invalid parameter
-91 : current pixel has no successors
```

```
        -92 : maximum number of regions has been exceeeded
        -93 : ROI border has been violated
        -94 : Maximum number of nodes has been exceeded
        -95 : Chain is NULL
        -96 : Too many border pixels found
        -97 : Error in seeds
        -98 : Stack is empty
        -99 : Stack is not empty originally
       -100 : It is a grayscale image
       -200 : Invalid video mode
       -201 : Not supported video mode
       -300 : Unable to open file
       -310 : error in decoding tree construction
       -320 : decoding error
       -330 : input file does not contain enough information for decoding
       -340 : ClearCode missing from input file while decoding
          or must be outputed while encoding
       -400 : Watermarking cannot be applied to this image
       -600 : fopen error
       -601 : fread error
       -602 : fwrite error

TIFF error messages

       -1000 : TIFF Error
```

Index