

https://dspace.org/

A knowledge-based machine vision system

Lu, Ning

1997
https://hdl.handle.net/10539/28954

Downloaded from a DSpace repository. https://dspace.org/

```
        return;
}

obj_inference( int nobj )
{
        Obj_table       *obj_pt,*base_pt;
        int             i,j,k;

        obj_pt = &object[nobj];
        if ( obj_pt->nhole == 0 )
        return 2;
        for ( i=0; i<NOBJ_BASE; i++ )
        {
                base_pt = &obj_base[i];
                if ( obj_pt->gen_no[0] == base_pt->gen_no[0] )
                {
                        if ( obj_pt->nhole == base_pt->nhole )
                        {
                                k = obj_pt->nhole;
                                for( j=1; j<k+1; j++ )
                                {
                                        if( obj_pt->gen_no[j] != base_pt->gen_no[j] )
                                        goto next_obj;
                                }
                                goto found_obj;
                        } else
                        goto next_obj;
                }
next_obj:       continue;
        }
        return  0;
found_obj:
        obj_pt->no = base_pt->no;
        obj_pt->name = base_pt->name;
        return 1;
}
/*
int add_obj_base()
{
        return;
}
*/
obj_analysis(flg,b,fbu,w)
        but_t           *b;
        win_t   *fbu,*w;
{
        register  en, key;
        coor_t          svpos;
        int             i,n, infer_out,k;

        switch (flg) {
        case F_RFSH: showbut(b,BUBCOL,BUTCOL); break;
        case F_EXEC:
                Gcursor(0);
                Gcursor(3,&svpos);
```

018AKAN_ANIL @ U"ANIL KA[illegible] DEPT OF PSYCH

```
                        else if (en) break;
        printf("\n");
printf("\n Number of Object:%d",Nobj);
if ( *io_key == 'y' ) {
        fprintf(io,"\n Object Inference result:");
        fprintf(io,"\n Number of Object:%d",Nobj);
}
n = 2;
for ( i=1; i<Nobj+1; i++ )
{
        infer_out = obj_inference( i );
        printf("\n inference result:%d",infer_out);
        if ( infer_out == 0 )
        add_obj_base();
        if ( infer_out == 1 )
        {
                k = object[i].no -1;
                display_otable( 1,k );
                display_otable( 2,i );
                if ( *io_key == 'y' ) {
                printf("\n Save dada.Waiting...");
                save_otable( 1,k );
                save_otable( 2,i );
                }
        }
        if ( infer_out == 2 ) {
                k = object[i].gen_no[0];
                display_gtable(3,k);
                display_gtable( 3,n );
                if ( *io_key == 'y' ) {
                printf("\n Save data. Waiting...");
                save_gtable( 3,n );
                }
        }
        printf("\n Click to continue ");
        for (en = 0;;)
                if (Gcursor(2))
                        if (Gcursor(3,0) == 0) en = 1;
                        else if (en) break;
        printf("\n");
        n+=2;
}
showbut(b,BUBCOL,BUTCOL);
Gcursor(4,&svpos);
```

```
                Gcursor(1);
                break;
        }
        return;
}

/*
int add_gen_base()
{
        return;
}
*/
int blob_verify( int n )
{
        register  en;
        Gen_table       *gen,*blob_ptr;
        int             k;

        k = blob[n].no - 1;
        gen = &gen_base[k];
        blob_ptr = &blob[n];

        display_gtable(1,k);
        blob_ptr->name = gen->name;
        display_gtable(2,n);
        printf(" Click to continue ");
        for (en = 0;;)
                if (Gcursor(2))
                        if (Gcursor(3,0) == 0) en = 1;
                        else if (en) break;
        printf("\n");
        return 1;
}

int blob_rule( int n )
{
        Gen_table       *blob_ptr;

        flg_triangle = 0;
        flg_quart = 0;
        blob_ptr = &blob[n];

        if ( blob_ptr->circle == 1 )
        {
                blob_ptr->no = 1;
                return 1;
        }
        if ( blob_ptr->arc_num ==0 && blob_ptr->line_num ==3 )
        {
                flg_triangle = 1;
                goto triangle_test;
        }
        if ( blob_ptr->arc_num ==0 && blob_ptr->line_num ==4 )
        {
                flg_quart = 1;
```

```
                goto quart_test;
        }
six_gon_test:
        if ( blob_ptr->arc_num ==0 && blob_ptr->line_num ==6 )
        {
                if ( blob_ptr->eqa_num != 6 ) {
                blob_ptr->no = 4;
                return 1;
                }
                return 0;
        }
window_test:
        if ( blob_ptr->arc_num == 1  && blob_ptr->line_num ==3 )
        {
                if( blob_ptr->eq_90 ==2 )
                {
                        blob_ptr->no = 10;
                        return 1;
                }
        }
        return 0;
triangle_test:

        if ( blob_ptr->eql_num==0 )
        {
                if( blob_ptr->gt_90 == 0 ) {
                        blob_ptr->no = 5;
                        return 1;
                }
                if( blob_ptr->gt_90 == 1 ) {
                        blob_ptr->no = 6;
                        return 1;
                }
        }
        if( blob_ptr->eql_num == 2 && blob_ptr->eq_90==0 ) {
                blob_ptr->no = 7;
                return 1;
        }
        if( blob_ptr->eql_num == 3 || blob_ptr->eqa_num ==3 ) {
                blob_ptr->no = 8;
                return 1;
        }
        if( blob_ptr->eq_90 == 1 ) {
                blob_ptr->no = 9;
                return 1;
        }
        return 0;
quart_test:

        if ( blob_ptr->eql_num==4 )
        {
                if( blob_ptr->eq_90 >= 1 ) {
                        blob_ptr->no = 2;   /* square */
                        return 1;
                }
```

```
                if( blob_ptr->eqa_num==2 ) {
                        blob_ptr->no = 11; /* parrell */
                        return 1;
                }
        }
        if ( blob_ptr->eql_num>=3 )
        {
                if( blob_ptr->eq_90 >= 2 ) {
                        blob_ptr->no = 2;/* square */
                        return 1;
                }
        }
        if ( blob_ptr->eql_num==2 )
        {
                if( blob_ptr->eq_90 >2  ) {
                        blob_ptr->no = 3;/* rectangle */
                        return 1;
                }
                if ( blob_ptr->eqa_num==2 && blob_ptr->gt_90==2)  {
                        blob_ptr->no = 11;          /* parrel */
                        return 1;
                }
        }
        if ( blob_ptr->eq_90==2 && blob_ptr->gt_90==1 && blob_ptr->les_90==1 ) {
                blob_ptr->no = 12;
                return 1;
        }

        return 0;
}

save_blob_inf( int n )
{
        register  en;
        Gen_table       *gen,*blob_ptr;
        int             k;

        k = blob[n].no - 1;
        gen = &gen_base[k];
        blob_ptr = &blob[n];

        save_gtable(1,k);
        blob_ptr->name = gen->name;
        save_gtable(2,n);

        return;
}
int blob_inference( int n )
{
        int       rule_out;
        rule_out = blob_rule( n );
        printf("\n Inference processsing: %d", rule_out);
        printf("\n flg_triangle=%d flg_quart=%d",flg_triangle,flg_quart);

        if ( rule_out == 0 )
```

```
        add_gen_base();
        if ( rule_out == 1 )
        {
                flg_retry = blob_verify( n );
                if ( *io_key == 'y' ) {
                        printf("\n Save data. Waiting...");
                        save_blob_inf( n );
                }
        }
        return( flg_retry);
}
```

Head File: macro.h

```
/* Limits definition */
#define DTB     2
#define Tbelta  0.1745
#define NUMbk 3
#define KK      3
#define DHT     4
#define Lratio  0.05
#define Rratio  0.025
#define Num_out_arc     3
#define PI      3.141592654
#define Delt_r  0.1

/* Color  Definition */
#define RED     0xc1
#define BLUE    0x0d
#define GREEN 0x31
#define BGCOL 0x05    /* backgroud color:        gray blue */
#define BUBCOL        0x0d    /* button back ground:     blue    */
#define BUHCOL        0xc1    /* button highlight:       red     */
#define BUTCOL        0xfe    /* buttun text color:      white   */
#define CUHCOL        0x31    /*                         green   */

#define LUT_F2D               5

#define F_RFSH          1
#define F_EXEC                2
#define F_RECO                3


typedef struct {
        char    *b_text;
        int     (*b_proc)();
        win_t   b_win;
} but_t;

typedef struct
{
        coor_t  P1;
        coor_t  P2;
```

```
        int     type;
        int     sym;
        double  bta;
        double  len;
} Primitive;

typedef struct
{
        int     no;
        char    *name;
        int     circle;
        int     line_num;
        int     arc_num;
        int     eql_num;
        int     eqa_num;
        int     eq_90;
        int     gt_90;
        int     les_90;
        float   compct;
        float   thin;
        int     area;
        int     peri;
        int     xo,yo;
} Gen_table;

typedef struct
{
        int     no;
        char    *name;
        int     nhole;
        int     gen_no[4];
        float   r_area[4];
}Obj_table;

/* function protypes */
int verify( int n );
int inference( int n );
int rule( int n );
int add_base();
int gen_display(int flg,but_t *b,win_t *fbu, win_t *w);
int showbut( but_t *b, int bcol, int tcol );
int b_quit( int flg, but_t *b );
int b_live( int flg, but_t *b, win_t *fbu, win_t *w);
int b_blob_analysis( int flg, but_t *b );
int setupluts();
int main( );
```

File Name: imagep.h

```
/* Frame buffer */
/*win_t  fbu[4];*/
```

APPENDIX A

```
/* terminate flag */
int     term;

/* Corner points */
coor_t  left_t,left_b,right_t,right_b,top_l,top_r,bottom_l,bottom_r;
coor_t  left,right,top,bottom,Plm,Pam,Pc,P1,P2;
coor_t  A[6],B[6],C[6],D[6],cp[20],cp1[20];
coor_t  brk[128],maxh;
int     Ak,Bk,Ck,Dk,brk_num,Ncp;
int     brow_st,brow_end,bcol_st,bcol_end;

/* Blobs */
win_t   bbox[256];
int     Nblack,Nwhite,barea[256];
short   btree[256][4];
int     Nobj;
/* primitives */
int     xam,yam,xlm,ylm,x1,y1,x2,y2,xo,yo,xc,yc,plabel,Bnum;
int     flg_circle,flg_line,flg_arc,flg_dir,flg_multi,flg_p,flg_brk,flg_mh,flg_arc_low;
int     flg_triangle,flg_quart,flg_retry;
double  afa,belta,dd,hh,R,ro,Rx,max_dh, fit_ratio;
float   kxy;

Primitive       pr[20],pr1[20];
Gen_table       blob[256];
Obj_table       object[10];
```

APPENDIX B

C Source Code for the Fruit Inspection Machine Vision System

File Name: MV.C

```
#include <graph.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include   <dos.h>
#include   <process.h>
#include   <memory.h>
#include   <string.h>
#include <time.h>
#include "mvmenu.h"
#include "mvblst.h"
#include "mv.h"

/* Function prototypes */
int  main( void );


/* Array and enum for main menu */
ITEM mnuMain[] =
{               /* Highlight Char  Pos */
   { 0, "Quit"         },
   { 1, "Live"                 },
   { 0, "Snap"         },
   { 0, "Analysis"     },
   { 0, "Learning"     },
   { 0, "Write File"           },
   { 1, "Read File"            },
   { 0, "Display Data"    },
   { 0, "Classification"  },
   { 0, ""             }
};

/* Define constants (0, 1, 2,...) for menu choices */
enum CHOICES
{
   QUIT,LIVE.SNAP,ANALYSIS,LEARNING,WRITE,READ.DISPLAY,CLASSIFICATION,
   HISTOGRAM=10,THRESHOLD,MEASUREMENT,EXIT
};


/* Global video configuration */
struct videoconfig vc;

int main()
{
   short rowMid, colMid;
   short fColor, fFirstTime = TRUE;
   short iMode, iLastMode, iMainCur = 0, iModesCur = 0, iAnlyCur=0;
```

```
   /*allocate space for Raw data */

BufSeg=(unsigned char __huge *)_halloc(0x10000L,sizeof(unsigned char));
   if (BufSeg==NULL)
         {  printf("Insufficient memory available. \n");
            exit(1);
         }
/* Allocate memory for binary data */
Binary_data = ( unsigned char __far *)_fmalloc(0xff00);
   if ( Binary_data == NULL )
         { printf("Insufficient memory available. \n");
            exit(1);
         }
/* Allocate memory for BitMap */
BitMap = (unsigned char __far *)_fmalloc(0xe000);
   if ( BitMap==NULL )
         {  printf("Insufficient memory available. \n");
            exit(1);
         }

/* Initializing Global Data Table */

   GDT1.zero2=0;
   GDT1.s_acc_right=0x93;
   GDT1.d_acc_right=0x93;
   GDT1.s_length=0x8000;
   GDT1.d_length=0x8000;


   GDT2.zero2=0;
   GDT2.s_acc_right=0x93;
   GDT2.d_acc_right=0x93;
   GDT2.s_length=0x8000;
   GDT2.d_length=0x8000;


   for( j=0; j < 16; j++)
   {
         GDT1.zero1[j]=0;
         GDT1.zero3[j]=0;
   }
   GDT1.zero3[16]=0;
   GDT1.zero3[17]=0;

   for( j=0; j < 16; j++)
   {
         GDT2.zero1[j]=0;
         GDT2.zero3[j]=0;
   }
   GDT2.zero3[16]=0;
   GDT2.zero3[17]=0;

/* Screen          */

   _displaycursor( _GCURSOROFF );
   _getvideoconfig( &vc );
   rowMid = vc.numtextrows / 2;
```

```
colMid = vc.numtextcols / 2;
_clearscreen( _GCLEARSCREEN );

learn_num=0;
classify_flag=0;
while( TRUE )
{
      /* Select from menu. */
      colMid = 50;
   iMainCur = Menu( rowMid, colMid, mnuMain, iMainCur );

   /* Branch to menu choice. */
   switch( iMainCur )
   {
         case QUIT:
               /* End the program */
                 /* Disable driver */
                       DisableDriver();
               /* Free Memory */
               _ffree(BitMap);
               _ffree(Binary_data);
               _hfree(BufSeg);
         _setvideomode( _DEFAULTMODE );
         return FALSE;
           case LIVE:
                 get_image();
                 break;
           case SNAP:
                 Freeze();
                 break;
           case LEARNING:
                 learn();
         break;
           case ANALYSIS:
                 image_analysis();
                 break;
           case WRITE:
                 file_io('w');
                 break;
           case READ:
                 file_io('r');
                 break;
           case DISPLAY:
                 display_data();
                 break;
           case CLASSIFICATION:
                 Freeze();
                 classify();
                 break;
   }
  }

}
```

File Name: MVMENU.C

```
/* MVMENU.C - Module of functions to put menus on the screen and handle keyboard
 * input. To use it, include the MENU.H file in your program. The following
```

```
* functions are public:
*
*  Menu     -  Puts a menu on screen and reads input for it
*  Box      -  Puts a box on screen (fill it yourself)
*  GetKey   -  Gets ASCII or function key
*  _outchar -  Displays character using current text position and color
*
* The following structures are defined:
*
*  MENU     -  Defines menu colors, box type, and centering
*  ITEM     -  Defines text of menu item and index of highlight character
*
* The global variable "mnuAtrib" has type MENU. Change this variable to
* change menu appearance.
*/

#include <string.h>
#include <stddef.h>
#include <ctype.h>
#include <graph.h>
#include <bios.h>
#include "mvmenu.h"
#include "mvmouse.h"

/* Prototype for internal function */
static void Itemize( int row, int col, int fCur, ITEM itm, int cBlank );

/* Default menu attribute. The default works for color or B&W. You can
 * override the default value by defining your own MENU variable and
 * assigning it to mnuAtrib. Or you can modify specific fields at
 * run time. For example, you could use a different attribute for color
 * than for black and white.
 */
MENU mnuAtrib =
{
   _TBLACK, _TBLACK, _TWHITE, _TBRIGHTWHITE, _TBRIGHTWHITE,
   _TWHITE, _TWHITE, _TBLACK, _TWHITE, _TBLACK,
   TRUE,
   (unsigned char)'Ú', (unsigned char)'¿', (unsigned char)'Ù', (unsigned char)'À',
   (unsigned char)'³', (unsigned char)'Ä'
};


/* Menu - Puts menu on screen and reads menu input from keyboard. When a
 * highlighted hot key or ENTER is pressed, returns the index of the
 * selected menu item.
 *
 * Params: row and col - If "fCentered" attribute of "mnuAtrib" is true,
 *         center row and column of menu; otherwise top left of menu
 *       altem - array of structure containing the text of each item
 *         and the index of the highlighted hot key
 *       iCur - index of the current selection--pass 0 for first item,
 *         or maintain a static value
 *
 * Return: The index of the selected item
 *
 * Uses:  mnuAtrib
 */
```

```
int Menu( int row, int col, ITEM aItem[], int iCur )
{
   int i;
   int cItem, cchItem = 2; /* Counts of items and chars per item      */
   int acchItem[MAXITEM]; /* Array of counts of character in items   */
   char achHilite[36];    /* Array for highlight characters          */
   long bgColor;          /* Screen color, position, and cursor      */
   short fgColor;
   struct rccoord rc;
   unsigned fCursor;

   /* Save screen information. */
   fCursor = _displaycursor( _GCURSOROFF );
   bgColor = _getbkcolor();
   fgColor = _gettextcolor();
   rc = _gettextposition();

   /* Count items, find longest, and put count of each in array. Also,
    * put the highlighted character from each in a string.
    */
   for( cItem = 0; aItem[cItem].achItem[0]; cItem++ )
   {
      acchItem[cItem] = strlen( aItem[cItem].achItem );
      cchItem = (acchItem[cItem] > cchItem) ? acchItem[cItem] : cchItem;
      i = aItem[cItem].iHilite;
      achHilite[cItem] = aItem[cItem].achItem[i];
   }
   cchItem += 2;
   achHilite[cItem] = 0;        /* Null-terminate and lowercase string */
   strlwr( achHilite );

   /* Adjust if centered, and draw menu box. */
   if( mnuAttrib.fCentered )
   {
      row -= cItem / 2;
      col -= cchItem / 2;
   }
   Box( row++, col++, cItem, cchItem );

   /* Put items on menu. */
   for( i = 0; i < cItem; i++ )
   {
      if( i == iCur )
         Itemize( row + i, col, TRUE,  aItem[i], cchItem - acchItem[i] );
      else
         Itemize( row + i, col, FALSE, aItem[i], cchItem - acchItem[i] );
   }
   SetPtrPos( col + (cchItem / 2), row + iCur );

   iCur = EventLoop( row, col, aItem, iCur, cItem, cchItem,
                     acchItem, achHilite );
   _setbkcolor( bgColor );
   _settextcolor( fgColor );
   _settextposition( rc.row, rc.col );
   _displaycursor( fCursor );
   return iCur;
}
```

```
int EventLoop( int row, int col, ITEM altem[], int iCur, int cItem,
        int cchItem, int acchItem[], char achHilite[] )
{
  unsigned  uKey;    /* Unsigned key code                */
  int       iPrev;   /* Previous index                   */
  EVENT     meEvent;
  char      *pchT;   /* Temporary character pointer      */
  static    int fBtnDown = FALSE;

  while( TRUE )
  {
    uKey = GetKey( NO_WAIT );
    if( uKey )
    {
      switch( uKey )
      {
        case U_UP:          /* Up key     */
          iPrev = iCur;
          iCur = (iCur > 0) ? (--iCur % cItem) : cItem - 1;
          break;
        case U_DN:          /* Down key   */
          iPrev = iCur;
          iCur = (iCur < cItem) ? (++iCur % cItem) : 0;
          break;
        default:
          if( uKey > 256 )  /* Ignore unknown function key  */
            continue;
          pchT = strchr( achHilite, (char)tolower( uKey ) );
          if( pchT != NULL ) /* If in highlight string, evaluate */
            iCur = pchT - achHilite;/* and fall through  */
          else
            continue;     /* Ignore unkown ASCII key     */
        case ENTER:
          return iCur;
      }
    }
    else if( GetMouseEvent( &meEvent ) )
    {
      SetPtrVis( SHOW );

      /* If mouse is on the menu, respond to various events. */
      if( (meEvent.x >= col) && (meEvent.x < (col + cchItem)) &&
        (meEvent.y >= row) && (meEvent.y < (row + cItem)) )
      {
        /* If button is down, drag selection. */
        if( meEvent.fsBtn & LEFT_DOWN )
        {
          fBtnDown = TRUE;
          iPrev = iCur;
          iCur = meEvent.y - row;
        }
        /* If button goes up from down, select current. */
        else if( fBtnDown && !(meEvent.fsBtn & LEFT_DOWN) )
        {
          fBtnDown = FALSE;
          iCur = meEvent.y - row;
          return iCur;
        }
```

```
            /* Ignore if no button has been pushed. */
            else
               continue;
         }
         /* Ignore if off menu. */
         else
            continue;
      }
      else
         continue;

      /* Redisplay current and previous if we get here through arrow
       * move or drag.
       */
      Itemize( row + iCur,  col, TRUE,  altem[iCur],
            cchItem - acchItem[iCur] );
      Itemize( row + iPrev, col, FALSE, altem[iPrev],
            cchItem - acchItem[iPrev] );
   }
}

/* Box - Draw menu box, filling interior with blanks of the border color.
 *
 * Params: row and col - upper left of box
 *         rowLast and colLast - height and width
 *
 * Return: None
 *
 * Uses:   mnuAtrib
 */
void Box( int row, int col, int rowLast, int colLast )
{
   int i;
   char achT[MAXITEM + 2];        /* Temporary array of characters */

   /* Set color and position. */
   _settextposition( row, col );
   _settextcolor( mnuAtrib.fgBorder );
   _setbkcolor( mnuAtrib.bgBorder );

   /* Draw box top. */
   achT[0] = mnuAtrib.chNW;
   memset( achT + 1, mnuAtrib.chEW, colLast );
   achT[colLast + 1] = mnuAtrib.chNE;
   achT[colLast + 2] = 0;
   _outtext( achT );

   /* Draw box sides and center. */
   achT[0] = mnuAtrib.chNS;
   memset( achT + 1, ' ', colLast );
   achT[colLast + 1] = mnuAtrib.chNS;
   achT[colLast + 2] = 0;
   for( i = 1; i <= rowLast; ++i )
   {
      _settextposition( row + i, col );
      _outtext( achT );
   }
```

```
    /* Draw box bottom. */
    _settextposition( row + rowLast + 1, col );
    achT[0] = mnuAtrib.chSW;
    memset( achT + 1, mnuAtrib.chEW, colLast );
    achT[colLast + 1] = mnuAtrib.chSE;
    achT[colLast + 2] = 0;
    _outtext( achT );
}

/* Itemize - Display one selection (item) of a menu. This function
 * is normally only used internally by Menu.
 *
 * Params: row and col - top left of menu
 *         fCur - flag set if item is current selection
 *         itm - structure containing item text and index of highlight
 *         cBlank - count of blanks to fill
 *
 * Return: none
 *
 * Uses:   mnuAtrib
 */
void Itemize( int row, int col, int fCur, ITEM itm, int cBlank )
{
    int i;
    char achT[MAXITEM];          /* Temporary array of characters */

    /* Set text position and color. */
    _settextposition( row, col );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelect );
        _setbkcolor( mnuAtrib.bgSelect );
    }
    else
    {
        _settextcolor( mnuAtrib.fgNormal );
        _setbkcolor( mnuAtrib.bgNormal );
    }

    /* Display item and fill blanks. */
    strcat( strcpy( achT, " " ), itm.achItem );
    _outtext( achT );
    memset( achT, ' ', cBlank-- );
    achT[cBlank] = 0;
    _outtext( achT );

    /* Set position and color of highlight character, then display it. */
    i = itm.iHilite;
    _settextposition( row, col + i + 1 );
    if( fCur )
    {
        _settextcolor( mnuAtrib.fgSelHilite );
        _setbkcolor( mnuAtrib.bgSelHilite );
    }
    else
    {
        _settextcolor( mnuAtrib.fgNormHilite );
        _setbkcolor( mnuAtrib.bgNormHilite );
```

```
    }
    _outchar( itm.achItem[i] );
}

/* GetKey - Gets a key from the keyboard. This routine distinguishes
 * between ASCII keys and function or control keys with different shift
 * states. It also accepts a flag to return immediately if no key is
 * available.
 *
 * Params: fWait - Code to indicate how to handle keyboard buffer:
 *   NO_WAIT    Return 0 if no key in buffer, else return key
 *   WAIT       Return first key if available, else wait for key
 *   CLEAR_WAIT Throw away any key in buffer and wait for new key
 *
 * Return: One of the following:
 *
 *   Keytype                          High Byte   Low Byte
 *   -------                          ---------   --------
 *   No key available (only with NO_WAIT)    0          0
 *   ASCII value                       0      ASCII code
 *   Unshifted function or keypad      1      scan code
 *   Shifted function or keypad        2      scan code
 *   CTRL function or keypad           3      scan code
 *   ALT function or keypad            4      scan code
 *
 * Note:  getkey cannot return codes for keys not recognized by BIOS
 *        int 16, such as the CTRL-UP or the 5 key on the numeric keypad.
 */
unsigned GetKey( int fWait )
{
    unsigned uKey, uShift;

    /* If CLEAR_WAIT, drain the keyboard buffer. */
    if( fWait == CLEAR_WAIT )
        while( _bios_keybrd( _KEYBRD_READY ) )
            _bios_keybrd( _KEYBRD_READ );

    /* If NO_WAIT, return 0 if there is no key ready. */
    if( fWait && !_bios_keybrd( _KEYBRD_READY ) )
        return FALSE;

    /* Get key code. */
    uKey = _bios_keybrd( _KEYBRD_READ );

    /* If low byte is not zero, it's an ASCII key. Check scan code to see
     * if it's on the numeric keypad. If not, clear high byte and return.
     */
    if( uKey & 0x00ff )
        if( (uKey >> 8) < 69 )
            return( uKey & 0x00ff );

    /* For function keys and numeric keypad, put scan code in low byte
     * and shift state codes in high byte.
     */
    uKey >>= 8;
    uShift = _bios_keybrd( _KEYBRD_SHIFTSTATUS ) & 0x000f;
    switch( uShift )
    {
```

```
      case 0:
        return( 0x0100 | uKey ); /* None (1)   */
      case 1:
      case 2:
      case 3:
        return( 0x0200 | uKey ); /* Shift (2)  */
      case 4:
        return( 0x0300 | uKey ); /* Control (3) */
      case 8:
        return( 0x0400 | uKey ); /* Alt (4)    */
    }
}

/* _outchar - Display a character. This is the character equivalent of
 * _outtext. It is affected by _settextposition, _settextcolor, and
 * _setbkcolor. It should not be used in loops. Build strings and then
 * _outtext to show multiple characters.
 *
 * Params: ch - character to be displayed
 *
 * Return: none
 */
void _outchar( char ch )
{
   static char ach[2] = " ";     /* Temporary array of characters */

   ach[0] = ch;
   _outtext( ach );
}


/* ClickOrPress - Checks to see if a key has been pressed or a mouse
 * button clicked. A click is defined as pressing and then releasing.
 *
 * Params: none
 *
 * Return: TRUE or FALSE
 */
int ClickOrPress()
{
   EVENT ev;
   int i = 0;

   /* Check for press. */
   if( GetKey( NO_WAIT ) )
      return TRUE;

   /* Check for click. If button is down, wait until it is released. */
   if( !GetMouseEvent( &ev ) )
      return 0;
   if( ev.fsBtn )
   {
      while( TRUE )
         if( GetMouseEvent( &ev ) && !ev.fsBtn )
            return TRUE;
   }
   return FALSE;
}
```

File Name: MVMOUSE.C

```
/* MVMOUSE.C - Module of mouse functions. To use it, include the MOUSE.H file
 * in your program. The following functions are public:
 *
 *  MouseInit     - Initialize mouse
 *  GetMouseEvent - Get information about most recent mouse event
 *  SetPtrVis     - Set visibility of pointer to HIDE or SHOW
 *  SetPtrPos     - Set position of pointer
 *  SetPtrShape   - Set shape of pointer in graphics modes, or
 *                  character and color in text modes
 *  GetPtrPos     - Get pointer position and button status
 *
 * The following structure is defined:
 *
 *  EVENT     -  Defines x, y, and mouse status of a mouse event
 */

#include <graph.h>
#include "mvmouse.h"

/* Internal information used by various mouse functions. */
struct MOUINFO
{
   int     fExist, fInit, fGraph,
   short   xVirtual, yVirtual;
   short   xActual,  yActual;
   short   xLast,    yLast;
   unsigned fsBtnLast, cBtn;
} static mi =
{
   1, 0, 0,
   0, 0,
   0, 0,
   0, 0,
   0, 0
};

#pragma optimize( "lge", off )    /* /Ol, /Og, and /Oe cannot be used */
                                  /* with inline assembler          */

/* MouseInit - Initialize mouse and turns on mouse pointer. Initializes
 * all internal variables used by other mouse functions. This function
 * should be called whenever a new video mode is set, since internal
 * variables are mode-dependent.
 *
 * Params: none
 *
 * Return: 0 if no mouse available, otherwise number of buttons available
 */
int MouseInit()
{
   struct videoconfig vc;
   char __far *pMode = (char __far *)0x00000449L; /* Address for mode */

   /* Get video configuration. */
   _getvideoconfig( &vc );
```

```
/* Handle special case of Hercules graphics. To use mouse with video
 * page 0. assume mode 6. To use mouse with page 1, assume mode 5.
 * Since the mouse functions couldn't easily detect and adjust for
 * page changes anyway, this code assumes page 0. Note also that the
 * mouse for Hercules graphics must be set in text mono mode.
 */
if( vc.mode == _HERCMONO )
{
   _setvideomode( _TEXTMONO );
   *pMode = 6;
}

mi.fInit = 1;
__asm
{
   sub    ax, ax          ; Mouse function 0, reset mouse
   mov    mi.cBtn, ax     ; Assume no mouse buttons
   int    33h
   mov    mi.fExist, ax   ; Set existence flag for future calls
   or     ax, ax          ; If AX = 0, there is no mouse
   jz     nomouse
   mov    mi.cBtn, bx     ; Save number of mouse buttons for return
nomouse:
}
if( !mi.fExist )
   return 0;

/* Set graphics flag. */
if( vc.numxpixels )
{
   mi.fGraph = 1;
   mi.yActual = vc.numypixels - 1;
   mi.xActual = vc.numxpixels - 1;
}
else
   mi.fGraph = 0;

/* The mouse works on a virtual screen of 640 x pixels by (8 * textrows)
 * vertical pixels. By default, it assumes 640 x 200 for 25-line mode.
 * You must call function 8 to adjust for other screen sizes.
 */
mi.xVirtual = 639;
if( mi.fGraph )
   mi.yVirtual = vc.numypixels - 1;
else
   mi.yVirtual = (vc.numtextrows << 3) - 1;

/* Reset Hercules graphics mode and reset the height. */
if( vc.mode == _HERCMONO )
{
   _setvideomode( _HERCMONO );
   mi.xVirtual = 719;
}

__asm
{
   mov    ax, 8           ; Set minimum and maximum vertica'
```

```
    sub   cx, cx          ; Minimum is 0
    mov   dx, mi.yVirtual ; Maximum is 8 * rows (or rows SHL 3)
    int   33h             ; Adjust for 25, 30, 43, 50, or 60 lines

    mov   ax, 1           ; Turn on mouse pointer
    int   33h

    mov   ax, 3           ; Get initial position and button status
    int   33h
    mov   mi.xLast, cx    ; Save internally
    mov   mi.yLast, dx
    mov   mi.fsBtnLast, bx
  }
  return mi.cBtn;         /* Return the number of mouse buttons */
}

/* GetMouseEvent - Check to see if there has been a mouse event. If event
 * occurred, update event structure.
 *
 * Params: pEvent - Pointer to event structure
 *
 * Return: 1 if event, 0 if no event
 */
int GetMouseEvent( EVENT __far *pEvent )
{
  int rtn;

  /* Make sure that mouse is initialized and exists. */
  if( !mi.fInit )
    MouseInit();
  if( !mi.fExist )
    return 0;

  __asm
  {
    mov   ax, 3           ; Get Mouse position and button status
    int   33h
    sub   ax, ax          ; Assume no event

    cmp   cx, mi.xLast    ; Has column changed?
    jne   event
    cmp   dx, mi.yLast    ; Has row changed?
    jne   event
    cmp   bx, mi.fsBtnLast ; Has button changed?
    je    noevent
event:
    mov   ax, 1           ; If something changed, event occurred
    mov   mi.xLast, cx    ; Update internal variables
    mov   mi.yLast, dx
    mov   mi.fsBtnLast, bx
noevent:
    mov   rtn, ax         ; Set return value
  }

  /* If event, put adjust values in structure. */
  if( rtn )
  {
    /* If graphics mode, adjust virtual mouse position to actual
```

```
     * screen coordinates.
     */
    if( mi.fGraph )
    {
      pEvent->x = (short)((long)mi.xLast * mi.xActual) / mi.xVirtual;
      pEvent->y = (short)((long)mi.yLast * mi.yActual) / mi.yVirtual;
    }
    /* If text mode, adjust virtual mouse position to 1-based
     * row/column.
     */
    else
    {
      pEvent->x = (mi.xLast >> 3) + 1;
      pEvent->y = (mi.yLast >> 3) + 1;
    }
    pEvent->fsBtn = mi.fsBtnLast;
  }
  return rtn;
}

/* GetPtrPos - Get mouse pointer position and button status regardless of
 * whether there was an event.
 *
 * Params: pEvent - Pointer to event structure
 *
 * Return: 0 if no mouse, otherwise 1
 */
int GetPtrPos( EVENT __far *pEvent )
{
  /* Make sure that mouse is initialized and exists. */
  if( !mi.fInit )
    MouseInit();
  if( !mi.fExist )
    return 0;

  __asm
  {
    mov   ax, 3          ; Get Mouse position and button status
    int   33h
    les   di, pEvent
    mov   es:pEvent[di].x, cx
    mov   es:pEvent[di].y, dx
    mov   es:pEvent[di].fsBtn, bx
  }

  /* If graphics mode, adjust virtual mouse position to actual
   * screen coordinates.
   */
  if( mi.fGraph )
  {
    pEvent->x = (short)((long)pEvent->x * mi.xActual) / mi.xVirtual;
    pEvent->y = (short)((long)pEvent->y * mi.yActual) / mi.yVirtual;
  }
  /* If text mode, adjust virtual mouse position to 1-based
   * row/column.
   */
  else
  {
```

```
      pEvent->x >>= 3;
      pEvent->y >>= 3;
      pEvent->x++;
      pEvent->y++;
    }
    return 1;
}

/* SetPtrVis - Set pointer visibility.
 *
 * Params: state - SHOW or HIDE
 *
 * Return: 0 if no mouse, otherwise 1
 */
int SetPtrVis( PTRVIS pv )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    __asm
    {
        mov ax, pv              ; Show or hide mouse pointer
        int 33h
    }
}

/* SetPtrPos - Set mouse pointer position.
 *
 * Params: x - column position in text modes, actual x coordinate in graphics
 *         y - row position in text modes, actual y coordinate in graphics
 *
 * Return: 0 if no mouse, otherwise 1
 */
int SetPtrPos( short x, short y )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
        MouseInit();
    if( !mi.fExist )
        return 0;

    /* If graphics, adjust actual coordinates to virtual coordinates. */
    if( mi.fGraph )
    {
        x = (short)((long)x * mi.xActual) / mi.xVirtual;
        y = (short)((long)y * mi.yActual) / mi.yVirtual;
    }
    /* If text, adjust row/column to 0-based virtual coordinates. */
    else
    {
        x--;
        y--;
        x <<= 3;
        y <<= 3;
    }
```

```
    __asm
    {
      mov   ax, 4           ; Set mouse position
      mov   cx, x
      mov   dx, y
      int   33h
    }
    return 1;
}

/* SetPtrShape - Set mouse pointer shape.
 *
 * Params: x - column position in text modes, actual x coordinate in graphics
 *         y - row position in text modes, actual y coordinate in graphics
 *
 * Return: 0 if no mouse, otherwise 1
 */
int SetPtrShape( PTRSHAPE _far *ps )
{
    /* Make sure that mouse is initialized and exists. */
    if( !mi.fInit )
      MouseInit();
    if( !mi.fExist )
      return 0;

    /* If graphics, use pointer shape bitmask array. */
    if( mi.fGraph )
    {
      __asm
      {
        les   di, ps
        mov   bx, es:[di].g.xHot    ; Load hot spot offsets
        mov   cx, es:[di].g.yHot
        mov   dx, di
        add   dx, 4

        mov   ax, 9             ; Set graphics pointer
        int   33h
      }
    }
    /* If text, use pointer color/character values. */
    else
    {
      __asm
      {
        les   di, ps
        mov   bx, 0             ; Use software cursor
        mov   cl, es:[di].t.chScreen
        mov   ch, es:[di].t.atScreen
        mov   dl, es:[di].t.chCursor
        mov   dh, es:[di].t.atCursor

        mov   ax, 10            ; Set text pointer
        int   33h
      }
    }
    return 1;
```

```
}

#pragma optimize( "", on )
```

File Name: ANALYSIS.C

```
/*****************************************************************************/
/*                                                                           */
/*   ANALYSIS.C   Image analysis & measurment                                */
/*                                                                           */
/*****************************************************************************/

#include   <stdio.h>
#include   <stdlib.h>
#include   <malloc.h>
#include   <signal.h>
#include   <dos.h>
#include   <process.h>
#include   <memory.h>
#include   <string.h>
#include   <graph.h>
#include   <math.h>
#include   "mv.h"
#include   <time.h>
#include   "mvmenu.h"
#include   "mvblst.h"
int            xleft,yleft,xright,yright;
unsigned   char   mask0=0x80;
unsigned char far   *Screen=( unsigned char far *)0xA0000000;
/* Function prototypes */
int image_analysis();
// Frame Buffer Operation
void bmove();
void bread( unsigned char x);
//Binary_data operation
void _frame_to_binary(),
void _pixel_cnt();
void _threshold();
void remove_noise();
void search_diameter();
void grayscale_analysis();
void menu_grayscale();
_measurement();
// Display functions
void binary_to_bitmap();
void binary_image_dsp();
//
void valley( char key);

/* Array and enum for sub-menu Analysis   */
ITEM mnuAnly[] =
{
   { 0, "Histogram"          },
   { 0, "Threshold"          },
   { 0, "Measurement"        },
   { 0, "Intensity" },
   { 0, "Exit"       },
```

```
};
enum ANALY
{
  EXIT,THRESHOLD,MEASUREMENT,HISTOGRAM,INTENSITY
};
void sum()
{
  unsigned      int      i;


  for(i=0;i<0xfffe;i+=2)
  {
        pixel_val= *(BufSeg+i);
        pixel_val= pixel_val >> 1;
        pixel_num[pixel_val] ++ ;
   }
        i=0xfffe;
        pixel_val=*(BufSeg+i);
        pixel_val= pixel_val >> 1;
        pixel_num[pixel_val] ++ ;
}

void bmove()
{
  struct GDT far *gdtp;
  gdtp = &GDT1;
  inregs.h.ah=0x87;
  inregs.x.cx= 0x4000;
  inregs.x.si= FP_OFF(gdtp);
  segreg.es = FP_SEG(gdtp);
  _int86x(0x15,&inregs,&outregs,&segreg );
  if (outregs.x.cflag)
        printf(" Data Transferring Error(First 32K) !");

/* Another 32k data transferring */

  gdtp = &GDT2;
  inregs.h.ah=0x87;
  inregs.x.cx= 0x4000;
  inregs.x.si= FP_OFF(gdtp);
  segreg.es = FP_SEG(gdtp);
  _int86x(0x15,&inregs,&outregs,&segreg );
  if (outregs.x.cflag)
        printf("Data transferring error (Second 32K)!");
}

void bread( unsigned char x)
{
  /* ' itialising soure and destine address */
  ad  r=(unsigned long )BufSeg;
  GDT1.d_ad0 = (unsigned char)(addr >> 12);
  GDT1.d_ad1 = (unsigned char)(addr >> 20);
  GDT1.d_ad2 = (unsigned char)(addr >> 28);
  GDT1.s_ad1 = 0;
  GDT1.s_ad0=0;

  addr=addr+0x8000000L;
  GDT2.d_ad0 = (unsigned char) (addr >> 12);
```

```
   GDT2.d_ad1 = (unsigned char)(addr >> 20);
   GDT2.d_ad2 = (unsigned char)(addr >> 28);
   GDT2.s_ad1 = 0x80;
   GDT2.s_ad0=0;

         GDT1.s_ad2=x;
         GDT2.s_ad2=x;
         bmove();

}

void bwrite( unsigned x)
{

   /* Initialising soure and destine address */
   addr=(unsigned long )BufSeg;
   GDT1.s_ad0 = (unsigned char)(addr >> 12);
   GDT1.s_ad1 = (unsigned char)(addr >> 20);
   GDT1.s_ad2 = (unsigned char)(addr >> 28);
   GDT1.d_ad1 = 0;
   GDT1.d_ad0=0;

   addr=addr+0x8000000L,
   GDT2.s_ad0 = (unsigned char) (addr >> 12);
   GDT2.s_ad1 = (unsigned char)(addr >> 20);
   GDT2.s_ad2 = (unsigned char)(addr >> 28);
   GDT2.d_ad1 = 0x80;
   GDT2.d_ad0=0;

   GDT1.d_ad2=x;
   GDT2.d_ad2=x;
   bmove();

}
void pixel_cnt()
{
         unsigned char ad2;
         int x,y;

/* Clear up array pixel_num[]  */

   for (j=0; j< 128; j++)
         {
               _num[j]=0;
         }

/* Data transferring  and  Counting */

   for(j=0xf0; j<0xf6; j++)
   {
         ad2=(unsigned char)j;
         bread(ad2);
         for ( y=0; y<32; y++)
            {
                  for ( x=0; x<320; x++)
                     {
                           pixel_val=*(BufSeg+y*2048+x*2);
                           pixel_val= pixel_val>>1;
```

```
                        pixel_num[pixel_val]++;
                    }
                }
            }
            bread(0xf6);
            for ( y=0; y<8; y++)
                {
                  for ( x=0; x<320; x++)
                        {
                            pixel_val=*(BufSeg+y*2048+x*2);
                            pixel_val= pixel_val>>1;
                            pixel_num[pixel_val]++;
                        }
                }
}


void threshold()
{
   unsigned long   line1;
   unsigned short  line;
/* threshold pixel values and data transferring */

   col_st=0;
   col_end=320;
   for(j=0xf0; j<0xf6; j++)
   {
/* Move 64k data from Frame Buffer to BufSeg */
        bread((unsigned char)j);
        lin0=(j&0x0f)*32;
/*threshold pixel values and put them in Binary_data */
        for(line=0; line<32; line++)
        {
            line1=((unsigned long)line)*2048;
            lin=lin0+line;
         for ( col=col_st; col<col_end; col++)
            {
              pixel_val= *(BufSeg +line1+col*2);
              pixel_val >>=1;
              if( pixel_val < threshold_val )
                    *(Binary_data +lin*XWIDTH+col )= 0;
                  else
                        *(Binary_data+lin*XWIDTH+col)=1;
            }
        }
   }
   bread( 0xf6 );
   lin0=6*32;
   for(line=0; line<8; line++)
        {
            line1=line*2048;
            lin=lin0+line;
         for ( col=col_st; col<col_end; col++)
            {
              pixel_val= *(BufSeg +line1+col*2);
              pixel_val >>=1;
              if( pixel_val < threshold_val )
```

```
                    *(Binary_data +lin*XWIDTH+col )= 0;
                        else
                          *(Binary_data+lin*XWIDTH+col)=1;
                    }
                }

}

void _frame_to_binary()
{
  unsigned long  line1;
  unsigned short  line;
/* Selection of range */
  col_st=0;
  col_end=320;
  for(j=0xf0; j<0xf6; j++)
  {
/* Move 64k data from Frame Buffer to BufSeg */
        bread((unsigned char)j);
        lin0=(j&0x0f)*32;
/* Take grayscale values and put them in Binary_data*/
          for(line=0; line<32; line++)
          {
                line1=((unsigned long)line)*2048;
                lin=lin0+line;
           for ( col=col_st; col<col_end; col++)
                {
                  pixel_val= *(BufSeg +line1+col*2);
                  pixel_val >>=1;
                  *(Binary_data +lin*XWIDTH+col )= pixel_val;
                }
          }
  }
  bread( 0xf6 );
  lin0=6*32;
  for(line=0; line<8; line++)
          {
                line1=line*2048;
                lin=lin0+line;
           for ( col=col_st; col<col_end; col++)
                {
                  pixel_val= *(BufSeg +line1+col*2);
                  pixel_val >>=1;
                    *(Binary_data +lin*XWIDTH+col )= pixel_val;
                }
          }

}
void _pixel_cnt()
{
  /* Clear up array pixel_num[]  */

  for (j=0; j< 128; j++)
        {
          pixel_num[j]=0;
        }
  /* Counting pixel intensity occurence number */
```

```
   for( lin=10; lin<190; lin++)
   {
         for ( col=10; col<310; col++)
           {
                 pixel_val=*(Binary_data +lin*XWIDTH+col);
                 pixel_num[pixel_val] ++;
           }
   }

}

void _threshold()
{
   for ( lin=10; lin<200; lin++)
   {
         for ( col=10; col<320; col++)
           {
                 pixel_val=*(Binary_data +lin*XWIDTH+col);
                 if ( pixel_val < threshold_val)
                   *(Binary_data +lin*XWIDTH+col)=0;
                 else
                   *(Binary_data +lin*XWIDTH+col)=1;
           }
   }

}


void binary_to_bitmap()
{

   /* clear the memory */
   _fmemset( BitMap,0,0xd000);
   /* convert binary data to bitmap */
   for ( lin=0; lin<200; lin++)
         {

           for ( col=0; col<XWIDTH; col++ )
                 {
                   pixel_val= *(Binary_data +lin*XWIDTH + col);
                   if ( pixel_val==0 )
                         {
                           col_bit=((unsigned char) col)&0x07;
                           mask=mask0>>col_bit;
                           byte_val= *(BitMap + lin*128 +(col>>3));
                           byte_val=byte_val|mask;
                           *(BitMap + lin*128 +(col>>3))=byte_val;
                         }
                 }
         }
}
void valley( char key )
{
   unsigned      long     peak[6],min[6];
   unsigned      char     peak_val[6], min_val[6], i,j,m;
/* Searching for peaks */
   m=1;
   for ( i=1; i<126; i++ )
```

```
        {
          if ( pixel_num[i+1] < pixel_num[i] )
                {
                  for ( j=i+2; j<126; j++)
                        {
                          if ( pixel_num[j] > pixel_num[i] )
                                { if ( j>i+DELT )
                                    {   peak[m] = pixel_num[i];
                                        peak_val[m]=i;
                                        m++;
                                        i=j-1;
                                        break;
                                    }
                                  else
                                        {  i=j-1;
                                           break;
                                        }
                                }
                        }
                 }

                if ( m == 7 )
                   {    printf( " peak number more than 6 !");
                        break;
                   }
                if ( j == 126)
                   {    peak[m] = pixel_num[i];
                        peak_val[m] = i;
                        m++;
                        break;
                   }

        }

/* Searching for two valleys */
   m=m-1;
   if ( m == 1 )
    {   printf( "Only one peak: %3d",peak_val[m]);
        return;
    }
   if ( m>2 )
        {
          for ( j=1; j<m; j++ )
                {
                  i= peak_val[j];
                  min[j]=pixel_num[i];
                  for ( i=peak_val[j]; i< peak_val[j+1]; i++ )
                        {
                          if ( pixel_num[i] < min[j] )
                                {  min[j] = pixel_num[i];
                                   min_val[j]=i;
                                }
                        }
                }
/*          if ( key=='M')
            { for ( j=1; j<m; j++)
              {
                    printf( " valley (%3d)= %3d ", j,min_val[j]);
```

```
            }
          }
*/
        }
   if ( m==2 )
        {
          i=peak_val[1];
          min[1]=pixel_num[i];
          for ( i=peak_val[1]; i<peak_val[2]; i++ )
                {
                  if ( pixel_num[i] < min[1] )
                        {  min[1] = pixel_num[i];
                          min_val[1]=i;
                        }
                }
/*          if ( key=='M' )
            printf( " Only one Valley = %3d ", min_val[1]);
*/
        }
/*   if ( (peak[1]<peak[3]) && (peak[3]< peak[2]))
        threshold_val=min_val[2];
   else
*/
   threshold_val=min_val[1];

}

data_compress()
{
   unsigned char k;
   unsigned short i;
   int    x,y;

/* Clear memory of BitMap */
   _fmemset( BitMap,0, 0xd000);
/* Compress raw data freezed in frame buffer into the bit-plane named as BitMap, according to the
given threshold value */
   for ( k=0xf0; k<0xfc; k++)
        {
          /* Move 64k data from FrameBuffer to BufSeg */
          bread(k);
          lin0 = ( k&0x0f )*32;
          /*Write to BitMap */
          for ( i=0; i<0xfffe; i+=2)
                {
                  col = ( i&0x07ff )>>1;
                  lin = lin0 + (i>>11);
                  col_bit = ( (unsigned char)col )& 0x07;

                  pixel_val=*(BufSeg +i);
                  pixel_val=pixel_val>>1;
                  if ( pixel_val<threshold_val )
                        {
                          mask= mask0>>col_bit;
                          byte_val= *(BitMap+lin*128+ (col>>3));
                          byte_val=byte_val | mask;
                          *(BitMap+ lin*128 + ( col>>3)) = byte_val;
                        }
```

```
                }
                i=0xfffe;
                  col = ( i&0x07ff )>>1;
                  lin = lin0 + (i>>11);
                  col_bit = ( (unsigned char)col )& 0x07;

                  pixel_val=*(BufSeg +i);
                  pixel_val=pixel_val>>1;
                  if ( pixel_val<=threshold_val )
                        {
                          mask= mask0>>col_bit;
                          byte_val= *(BitMap+lin*128+ (col>>3));
                          byte_val=byte_val | mask;
                          *(BitMap+ lin*128 + ( col>>3)) = byte_val;
                        }

            }

}

void binary_image_dsp()
{
   int x,y;
/*    printf("\n Starting line(0,200) &End line(0,200) ");
   scanf("%d%d",&lin_st,&lin_end);
   printf("\n Starting column(0,40)& End column(0,40) ");
   scanf("%d%d",&col_st,&col_end);
*/

/*  Selected image range */
   lin_st=10;
   lin_end=190;
   col_st=0;
   col_end=40;

/* Display on screen */
   _setvideomode( _VRES16COLOR );

/* clear screen */
   _fmemset( Screen, 0, 0x9600);

/* Displaying */
   for ( y=lin_st; y<lin_end;y++)
         {
           for ( x=col_st; x<col_end; x++)
                 {
                   *(Screen+y*80+x) = *( BitMap + y*128 +x);
                 }
         }

/* drawing diameter */
   _setcolor(2);
   _setlinestyle( 0xffff );
   _moveto( xleft,yleft);
   _lineto( xright,yright);
```

```
/* set cursor position          */
   regs.h.ah=2;
   regs.h.bh=0;
   regs.h.dh=13;
   regs.h.dl=0;
   int86(0x10,&regs,&regs);
   printf("\n Threshold value= %d",threshold_val);
   printf("\n Diameter = %d ", diameter);
   getch();
}

void remove_noise()
{
   unsigned       short    p4=0,p8=0,area=0;
   unsigned       char     a0,a1,a2,a3,a4,a5,a6,a7,a8;
   int    col_end_b;
   float  pi;
/*  Selected image range */
   lin_st=10;
   lin_end=200;
   col_st=20;
   col_end=40;
//   col_end_b=col_end*8;
   for ( lin=lin_st; lin<lin_end; lin++)
         {
           for ( col=col_st; col<300; col++)
                 {
                   a0 = *(Binary_data+ lin*XWIDTH + col);
                   if ( a0==1 )    /* a0=1 is foreground */
                         {
                           a1= *(Binary_data +(lin-1)*320 +col-1);
                           a2= *(Binary_data + (lin-1)*320 +col);
                           a3= *(Binary_data + (lin-1)*320 +col+1);
                           a4= *(Binary_data + lin*320 +col+1);
                           a5= *(Binary_data +(lin+1)*320 +col+1);
                           a6= *(Binary_data +(lin+1)*320 +col);
                           a7= *(Binary_data +(lin+1)*320 +col-1);
                           a8= *(Binary_data +lin*320 +col-1);
                           if ( a1 || a2 || a3 || a4 || a5 || a6 || a7 || a8 )
                                 p8 ++;
                           else
                                 *(Bir.ary_data+ lin*XWIDTH + col)=0;
                         }
                 }
         }

}
void search_diameter()
{
   int x,y;

   xleft=0;
   yleft=0;
   xright=0;
   yright=0;
   diameter=0;
   for ( x=20; x<300; x++)
```

```
        {
        for ( y=20; y<180; y++)
          {
                pixel_val= *(Binary_data + y*320 + x);
                if ( pixel_val==1 )
                  {
                        xright=x;
                        yright=y;

                  }
          }
        }

   for ( x=300; x>10; x--)
        {
          for ( y=20; y<180; y++)
          {
                pixel_val= *(Binary_data + y*320 + x);
                if ( pixel_val==1 )
                  {
                        xleft=x;
                        yleft=y;
                  }
          }
        }
   diameter=xright-xleft;

}

 void grayscale_analysis()
{
   int x,y,point1,point2,point3;
   int pixel_num_r2,pixel_num_r3,pixel_num_r4,foreground;
   int
max_num,max_val,avg_num,sub_sum,sub_val,max_sub,i,cut_point,avg_grayscale,sum_num;
   float  rate2,max_r;
   long   sum_pixel_val;
   char   key;
   _frame_to_binary();
   _pixel_cnt();
   /* Display on screen */
   _clearscreen( _GCLEARSCREEN );
   menu_grayscale();
   while( (key=getch()) != ESCAPE )
   {
    /* clear screen */
   _clearscreen( _GCLEARSCREEN );
   switch(key)
   {
         case 'R':
   _measurement();
   _frame_to_binary();
   _pixel_cnt();
   _settextposition(20,0);
   printf("  Plotting. Please waiting...");
/*  Selected image range */
   lin_st=10;
   lin_end=190;
```

```
col_st=1;
col_end=40;

pixel_num_r2=0;
pixel_num_r3=0;
pixel_num_r4=0;

/* Displaying */
for ( x=10; x<310;x++)
        {
          for ( y=10; y<190;y++)
                {
                  pixel_val= *(Binary_data+ y*320 +x);
                  if ( pixel_val <= threshold_val)
                        {
                          _setcolor(0);
                          _setpixel(x,y);
                        }
                  if ( pixel_val>threshold_val && pixel_val< cut_point1 )
                        {
                          _setcolor(1);
                          _setpixel(x,y);
                          pixel_num_r2++;
                        }
                  if ( (pixel_val>= cut_point1) && pixel_val<cut_point2 )
                        {
                          _setcolor(2);
                          _setpixel(x,y);
                          pixel_num_r2++;
                        }
                  if ( (pixel_val>= cut_point2) && pixel_val< cut_point3 )
                        {
                          _setcolor(3);
                          _setpixel(x,y);
                          pixel_num_r3++;
                        }
                  if ( pixel_val >= cut_point3)
                        {
                          _setcolor(4);
                          _setpixel(x,y);
                          pixel_num_r4++;
                        }
                }
        }
/* set cursor position        */
regs.h.ah=2;
regs.h.bh=0;
regs.h.dh=13;
regs.h.dl=0;
int86(0x10,&regs,&regs);
printf("\n Grayscale distribution analysis by assigning color to selected ranges ");
printf("\n Range0: [0--threshold] (black color)point1= %d ",cut_point1);
printf("\n Range1: [threshod--point1] (blue color)point1= %d ratio1=%f",cut_point1,ratio1);
printf("\n Range2: [point1--point2]   (Green color) point2= %d ratio2=%f ",cut_point2,ratio2);
printf("\n Range3: [point2--point3](light blue color) point3= %d ratio3=%f",cut_point3,ratio3);
printf("\n Range4: >point3 (  Red color      ) ");
printf("\n point1 -- point3 ratio=%f",ratio4);
getch();
```

```
  break;

        case 'C':
                    printf(" \n Input cut point:");
                    scanf("%d",&cut_point);
                    /* Calculating average number of pixels */
                    avg_num=0;
                    max_num=pixel_num[cut_point];
                    max_val=0;
                    sum_pixel_val=0;
                    sum_num=0;
                    for ( i=cut_point; pixel_num[i]>0; i++)
                          {
                            sum_num += pixel_num[i];
                            if ( pixel_num[i] > max_num )
                              { max_num = pixel_num[i];
                                    max_val = i;
                              }
                            sum_pixel_val += (long)i*((long)pixel_num[i]);

                          }
                    /* Object mean suface gray level */
                    avg_grayscale= (int)(sum_pixel_val/sum_num);
                    avg_num = sum_num/(i-cut_point);
                    sub_sum=0;
                    for ( i=cut_point; i<128; i++)
                          {
                            sub_val=pixel_num[i]-avg_num;
                            if ( sub_val>0)
                                  sub_sum += sub_val;
                          }
                    max_sub=pixel_num[max_val] - avg_num;
                    max_r= ((float)avg_num )/((float)pixel_num[max_val]);
                    printf("\n Average number=%d; Total num of pixel above
avg=%d",avg_num,sub_sum);
                    printf("\n Max val=%d at %d ; Average Grayscale=%d;\n max sub=%d;
ratio=%f",max_num,max_val,avg_grayscale,max_sub,max_r);
                    getch();
                    break;

   }
   menu_grayscale();
  }
  _fmemset( Screen, 0, 0x9600);
}


_measurement()
{
  unsigned      short    sum_pixel_num=0;
  unsigned      char     a0,a1,a2,a3,a4,a5,a6,a7,a8;
  long          sum_multi_val_num=0;
  int    col_end_b;

  get_color();
/* Transfer data */
 _frame_to_binary();
 _pixel_cnt();
```

APPENDIX B

```
/* Before threshold the Binary_data */

   area=0;
   for ( i=threshold_val; i<128;i++)
         {
           area+=(unsigned short)pixel_num[i];
           sum_multi_val_num += ((long)i) * ((long)(pixel_num[i]));
         }
   mean_grayscale = (int) ( sum_multi_val_num/((long)area));
   cut_point1 = (unsigned char) ( (float)mean_grayscale*0.92);
   cut_point2 = (unsigned char) ((float)mean_grayscale*0.95);
   cut_point3 = (unsigned char) ( (float)mean_grayscale*0.98);
   sum_pixel_num=0;
   for ( i=threshold_val; i<cut_point1;i++)
         sum_pixel_num+=(unsigned short)pixel_num[i];
   ratio1 =((float) sum_pixel_num)/((float)area);
   sum_pixel_num=0;
   for ( i= threshold_val; i<cut_point2; i++)
         sum_pixel_num+=pixel_num[i];
   ratio2 =((float) sum_pixel_num)/((f[illegible]a  a);
   sum_pixel_num=0;
   for ( i= threshold_val; i<cut_point3; i++)
         sum_pixel_num+=pixel_num[i];
   ratio3 =((float) sum_pixel_num)/((float)area);
    sum_pixel_num=0;
   for ( i=cut_point1; i< cut_point3; i++)
         sum_pixel_num+=pixel_num[i];
   ratio4 =((float) sum_pixel_num)/((float)area);
/* Threshold then calculate perimeter and area */
   _threshold();
   remove_noise();
   search_diameter();
   /*      Selected image range */
   lin_st=10;
   lin_end=190;
   col_st=20;
   col_end=40;
   col_end_b=310;
   area0=0;
   p4 = 0;
   p8 = 0;
   for ( lin=lin_st; lin<lin_end; lin++)
         {
           for ( col=col_st; col<col_end_b; col++)
                 {
                   a0 = *(Binary_data+ lin*XWIDTH + col);
                   if ( a0==0 )    /* a0=0 is foreground */
                         {
                           area0++;
                           a1= *(Binary_data +(lin-1)*320 +col-1);
                           a2= *(Binary_data + (lin-1)*320 +col);
                           a3= *(Binary_data + (lin-1)*320 +col+1);
                           a4= *(Binary_data +lin*320 +col+1);
                           a5= *(Binary_data +(lin+1)*320 +col+1);
                           a6= *(Binary_data +(lin+1)*320 +col);
                           a7= *(Binary_data +(lin+1)*320 +col-1);
                           a8= *(Binary_data +lin*320 +col-1);
                           if ( a1 || a2 || a3 || a4 || a5 || a6 || a7 || a8 )
```

```
                                p8 ++;
                            if ( a2|| a4 || a6 || a8 )
                                p4 ++;
                        }
                    }
            }
    pi= 12.56637*((float)area0)/((float)p4*(float)p8);
    perimeter = (unsigned short)sqrt(((double)p4)*((double)p8));

}

sample_dst()
{
    double sum_x,sum_sqr,
    sum_x=0;
    for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].diameter;
    mean_sample.diameter = (int)(sum_x/(double)sample_num);
    sum_sqr = 0;
    for (i=0; i<sample_num; i++)
        sum_sqr += (double)( (sample[i].diameter - mean_sample.diameter)*(sample[i].diameter
- mean_sample.diameter));
    std_dev_sample.diameter =  sqrt( sum_sqr/((double)sample_num));
//
    sum_x=0;
    for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].mean_grayscale;
    mean_sample.mean_grayscale = (int)(sum_x/(double)sample_num);
    sum_sqr = 0;
    for (i=0; i<sample_num; i++)
        sum_sqr += (double)( (sample[i].mean_grayscale -
mean_sample.mean_grayscale)*(sample[i].mean_grayscale - mean_sample.mean_grayscale));
    std_dev_sample.mean_grayscale =  sqrt( sum_sqr/((double)sample_num));
//
//
    sum_x=0;
    for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].area;
    mean_sample.area = (unsigned short)(sum_x/(double)sample_num);
    sum_sqr = 0;
    for (i=0; i<sample_num; i++)
        sum_sqr += (double)( (sample[i].area - mean_sample.area)*(sample[i].area -
mean_sample.area));
    std_dev_sample.area =  sqrt( sum_sqr/((double)sample_num));
//
    sum_x=0;
    for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].perimeter;
    mean_sample.perimeter = (unsigned short)(sum_x/(double)sample_num);
    sum_sqr = 0;
    for (i=0; i<sample_num; i++)
        sum_sqr += (double)( (sample[i].perimeter -
mean_sample.perimeter)*(sample[i].perimeter - mean_sample.perimeter));
    std_dev_sample.perimeter =      sqrt( sum_sqr/((double)sample_num));
//
    sum_x=0;
    for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].compactness;
```

```
   mean_sample.compactness = (float)(sum_x/(double)sample_num);
   sum_sqr = 0;
   for (i=0; i<sample_num; i++)
         sum_sqr += (double)( (sample[i].compactness -
mean_sample.compactness)*(sample[i].compactness - mean_sample.compactness));
   std_dev_sample.compactness = sqrt( sum_sqr/((double)sample_num));
//
   sum_x=0;
   for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].ratio1;
   mean_sample.ratio1 = (float)(sum_x/(double)sample_num);
   sum_sqr = 0;
   for (i=0; i<sample_num; i++)
         sum_sqr += (double)( (sample[i].ratio1 - mean_sample.ratio1)*(sample[i].ratio1 -
mean_sample.ratio1));
   std_dev_sample.ratio1 = sqrt( sum_sqr/((double)sample_num));
//
   sum_x=0;
   for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].ratio2;
   mean_sample.ratio2 = (float)(sum_x/(double)sample_num);
   sum_sqr = 0;
   for (i=0; i<sample_num; i++)
         sum_sqr += (double)( (sample[i].ratio2 - mean_sample.ratio2)*(sample[i].ratio2 -
mean_sample.ratio2));
   std_dev_sample.ratio2 = sqrt( sum_sqr/((double)sample_num));
//
   sum_x=0;
   for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].ratio3;
   mean_sample.ratio3 = (float)(sum_x/(double)sample_num);
   sum_sqr = 0;
   for (i=0; i<sample_num; i++)
         sum_sqr += (double)( (sample[i].ratio3 - mean_sample.ratio3)*(sample[i].ratio3 -
mean_sample.ratio3));
   std_dev_sample.ratio3 = sqrt( sum_sqr/((double)sample_num));
//
   sum_x=0;
   for ( i=0; i<sample_num; i++)
            sum_x += (double)sample[i].ratio4;
   mean_sample.ratio4 = (float)(sum_x/(double)sample_num);
   sum_sqr = 0;
   for (i=0; i<sample_num; i++)
         sum_sqr += (double)( (sample[i].ratio4 - mean_sample.ratio4)*(sample[i].ratio4 -
mean_sample.ratio4));
   std_dev_sample.ratio4 = sqrt( sum_sqr/((double)sample_num));
//
}
void menu_dsp()
{
   /*clear screen */
   /* set cursor position        */
   regs.h.ah=2;
   regs.h.bh=0;
   regs.h.dh=13;
   regs.h.dl=0;
   int86(0x10,&regs,&regs);
   printf( "\n  <A>uto threshold; \n  <M>easurement;\n  <T>hreshold ");
```

```
   printf( "\n  <H>istogram; \n  <G>rayscale analysis>; \n  <Esc> to exit. ");

}

void menu_grayscale()
{
        /*clear screen */
  _fmemset( Screen, 0, 0x9600);
  /* set cursor position        */
  regs.h.ah=2;
  regs.h.bh=0;
  regs.h.dh=10;
  regs.h.dl=0;
  int86(0x10,&regs,&regs);
  printf( "\n Surface Intensity Distribution Analysis \n (1)<Range> select ");
  printf( "\n (2) <C>ut point selection \n <Esc> to exit. ");

}

int image_analysis()
{
  char key;
  clock_t start,finish;
  double  duration;



/* Get Binary image */
  /* Transfer frame buffer to Binary area */
/*
  _frame_to_binary();
  start=clock();
  _pixel_cnt();
  threshold_val=58;
  _threshold();
  remove_noise();
  search_diameter();
  finish=clock();
  duration=( double )(finish-start)/CLOCKS_PER_SEC;
  printf("\n Time spending: %2.1fseconds",duration);
  getch();
  binary_to_bitmap();
  binary_image_dsp();
*/
        /* Display menu */
        menu_dsp();
  while( (key=getch()) != ESCAPE )
  {

        /* Branch to menu choices */
        switch (key)
        {

        case 'A':
                /* Compress Raw data in Frame Buffer into Binary_data area*/
                _frame_to_binary();
                _pixel_cnt();
                /* Automaticly choose threshold value */
```

```
                valley(key);
                _threshold();

                remove_noise();
                search_diameter();
                /* Display on screen */
                binary_to_bitmap();
                binary_image_dsp();
                _clearscreen( _GCLEARSCREEN );
                break;
        case 'T':
                /* set cusor position */
                inregs.h.ah=2;
                inregs.h.bh=0;
                inregs.h.dh=22;
                inregs.h.dl=0;
                _int86(0x10,&inregs,&outregs);
                printf("Input threshold value = ");
                scanf( " %d",&threshold_val );

                _frame_to_binary();
                _pixel_cnt();
                _threshold();
                remove_noise();
                search_diameter();
                _fmemset( Screen, 0, 0x9600);
                binary_to_bitmap();
                binary_image_dsp();
                break;

        case 'M':
                _setvideomode( _VRES16COLOR );
                _clearscreen( _GCLEARSCREEN );
                _settextposition(10,0);
                printf(" Input threshold value: ");
                scanf("%d",&threshold_val);
/*              printf("\n Cut point1: ");
                scanf("%d",&cut_point1);
                printf("\n Cut point2: ");
                scanf("%d",&cut_point2);
                printf("\n Cut point3: ");
                scanf("%d",&cut_point3);
*/
                _measurement();

                printf("\n P4=%d   P8=%d",p4,p8);
                printf("\n Perimeter= %hu",perimeter);
                printf("\n Area0=%hu   Area=%hu",area0,area);
                printf("\n PI= %f",pi);
                printf("\n Diameter = %d",diameter);
                printf("\n Surface Mean Grayscale= %d",mean_grayscale);
                printf("\n (Threshold value->cut point 1) Ratio1=%f",ratio1);
                printf("\n (Threshold value->cut point 2) Ratio2=%f",ratio2);
                printf("\n (Threshold value->cut point 3) Ratio3=%f",ratio3);
                printf("\n (Cut point1-> cut_point3) Ratio4=%f",ratio4);
                printf("\n Threshold value= %d",threshold_val);
                printf("\n Componantof Red= %d Green= %d Blue= %d",red,green,blue);
                getch();
```

```
                    _setvideomode( _DEFAULTMODE );
                    break;
            case 'G':
                    _setvideomode( _VRES16COLOR );
                    _clearscreen( _GCLEARSCREEN );
                    _settextposition(10,0);
                    printf(" Input threshold value: ");
                    scanf("%d",&threshold_val);
                    grayscale_analysis();
                    _setvideomode( _DEFAULTMODE );
                    break;
            case 'H':
                  /* Compress Raw data in Frame Buffer into Binary_data area*/
                    _frame_to_binary();
                    _pixel_cnt();
                    _clearscreen( _GCLEARSCREEN );
                    histogram_dsp();
                    break;
            }
            /* Display menu */
            menu_dsp();
    }
    _setvideomode( _DEFAULTMODE);
}

learn()
{
    _setvideomode( _VRES16COLOR );
    _clearscreen( _GCLEARSCREEN );
    if ( learn_num == 0)
            {
                    _settextposition(10,0);
                    printf(" Input threshold value: ");
                    scanf("%d",&threshold_val);
/*
                    printf("\n Cut point1: ");
                    scanf("%d",&cut_point1);
                    printf("\n Cut point2: ");
                    scanf("%d",&cut_point2);
                    printf("\n Cut point3: ");
                    scanf("%d",&cut_point3);
*/
            }
/*    Compress frame data into Binary_data  */
    _frame_to_binary();
    _pixel_cnt();
/*  Measurement */
    _measurement();
    sample[learn_num].area=area0;
    sample[learn_num].perimeter=perimeter;
    sample[learn_num].mean_grayscale=mean_grayscale;
    sample[learn_num].compactness = pi;
    sample[learn_num].ratio1 = ratio1;
    sample[learn_num].ratio2 = ratio2;
    sample[learn_num].ratio3 =ratio3;
    sample[learn_num].ratio4 = ratio4;

    remove_noise();
```

```
    search_diameter();
    sample[learn_num].diameter = diameter;

    learn_num++;
    sample_num=learn_num;

    _setvideomode( _DEFAULTMODE);
}

classify()
{
    unsigned short sum_pixel_num;
    long   sum_multi_val_num;
    float  min_u;
/* Read the template */
    if ( classify_flag == 0)
            file_io('t');
_setvideomode( _VRES16COLOR );
_clearscreen( _GCLEARSCREEN );
printf("\n mean_d %d std_dev_d %2.3f mean_r%f std_dev_r %f",mean_sample.diameter,
    std_dev_sample.diameter,mean_sample.ratio4,std_dev_sample.ratio4);
    printf("\n Threshold= %d Delt=%f",threshold_val,x_delt);
    if ( std_dev_sample.diameter ==0)
            {
            _setvideomode( _DEFAULTMODE);
            return(0);
            }
/* Image measurement */
 /* Transfer data */
 _frame_to_binary();
 _pixel_cnt();
    area=0;
    sum_multi_val_num = 0;
    for ( i=threshold_val; i<128;i++)
            {
              area+=(unsigned short)pixel_num[i];
              sum_multi_val_num += ((long)i) * ((long)(pixel_num[i]));
            }
    mean_grayscale = (int) ( sum_multi_val_num/((long)area));
    cut_point1 = (unsigned char) ( (float)mean_grayscale*0.92);
    cut_point2 = (unsigned char) ((float)mean_grayscale*0.95);
    cut_point3 = (unsigned char) ( (float)mean_grayscale*0.98);

    sum_pixel_num=0;
    for ( i=cut_point1; i< cut_point3; i++)
          sum_pixel_num+=pixel_num[i];
    ratio4 =((float) sum_pixel_num)/((float)area);
   /* Threshold then calculate perimeter and area */
    _threshold();
    remove_noise();
    search_diameter();
printf("\n Diameter= %d Ratio=%f",diameter,ratio4);
/* Calculate the membership fuction values */
if ( diameter< (mean_sample.diameter - 3.0*std_dev_sample.diameter))
    u1.diameter = 0;
else
    if ( diameter > ( mean_sample.diameter-std_dev_sample.diameter))
    u1.diameter = 1;
```

```
    else
      u1.diameter = (diameter-mean_sample.diameter+
3.0*std_dev_sample.diameter)/(2.0*std_dev_sample.diameter);
if ( ratio4 > ( mean_sample.ratio4 +  std_dev_sample.ratio4) )
    u1.ratio = 0;
else
    if ( ratio4 < (mean_sample.ratio4-std_dev_sample.ratio4))
    u1.ratio = 1;
    else
      u1.ratio= (mean_sample.ratio4 + std_dev_sample.ratio4-ratio4)/(2.0*std_dev_sample.ratio4);
/* Display results */
    printf("\n membership function u(d)= %2.1f",u1.diameter);
    printf("\n membership function u(r)= %2.1f",u1.ratio);
    printf("\n threshold delt= %2.1f",x_delt);
    min_u = u1.diameter;
    if ( u1.diameter > u1.ratio)
          min_u=u1.ratio;
    if ( min_u >= x_delt)
          printf("\n It belongs to this class");
            else
                  printf("\n It does not belong to this class");
    getch();
    _setvideomode( _DEFAULTMODE);
    classify_flag=1;
}


display_data()
{
    _setvideomode( _VRES16COLOR );
    /* Displaying on the screen */
    _clearscreen( _GCLEARSCREEN );
    _settextposition( 0,0);
    printf("Number of samples: %d",sample_num);
    printf("\nD mean area   P     Pi   ratio1 ratio2 ratio3  ratio4");
    for ( i=0; i<sample_num; i++)
    printf("\n%d %d %hu %hu  %f %f  %f %f %f",
    sample[i].diameter,sample[i].mean_grayscale,sample[i].area,
    sample[i].perimeter,sample[i].compactness,
    sample[i].ratio1,sample[i].ratio2,sample[i].ratio3,sample[i].ratio4);
    sample_dst();
    printf("\n Mean");
    printf("\n%d    %d    %hu   %hu  %f  %f %f %f %f",
    mean_sample.diameter,mean_sample.mean_grayscale,mean_sample.area,
    mean_sample.perimeter,mean_sample.compactness,
    mean_sample.ratio1,mean_sample.ratio2,mean_sample.ratio3,mean_sample.ratio4);
    printf("\n Standard deviation");
    printf("\n%f %f %f %f %f  %f  %f %f %f",
    std_dev_sample.diameter,std_dev_sample.mean_grayscale,std_dev_sample.area,
    std_dev_sample.perimeter,std_dev_sample.compactness,std_dev_sample.ratio1,
    std_dev_sample.ratio2,std_dev_sample.ratio3,std_dev_sample.ratio4);
    printf("\n threshold value: %d",threshold_val);
    printf("\n Cut point1=%d Cut point2=%d Cut point3=%d  ",cut_point1,cut_point2,cut_point3);
getch();

    _setvideomode( _DEFAULTMODE);

}
```

File Name: HISTOGRA.C

```
/* HISTOGRA.C provides functions for drawing pixel intensity histogram */

#include   <conio.h>
#include   <stdlib.h>
#include   <graph.h>

/* Macro definition for Histogram Axis */
#define    MAXROW  480
#define    YMAX    300
#define    XMAX    600
#define    MAX_P   128
#define    XO      40
#define    YO      400
#define    XD      4
#define    YD      4

extern unsigned long pixel_num[128];
   unsigned       long     max_pixel_num, max_y_axis;
   short  xx,yy,x,y,x1,y1;
   unsigned       int      hscale,i,yyd,max_pixel_point;
   int    mode = _VRES16COLOR;
   unsigned       char     title[20];

y_ruler()
{
/* plot y-axis ruler */
   yyd = YMAX /10;
   for( i=0; i<11; i++)
   {
         x=XO;
         y=YO-yyd*i;
         _moveto(x,y);
         x=x-XD;
         _lineto(x,y);
   }
/* print max y-axis value */
   _settextposition(6,0);
   _strset( title, ' ');
   _outtext( title );
   _settextposition(6,0);
   sprintf(title,"%lu",max_y_axis);
   _outtext( title );

}
histogram()
{
/* Plot the Histogram of Pixels intensity value */

   for(i=0; i<MAX_P; i++)
   {
         yy=pixel_num[i]/hscale;
         if( yy>YMAX )
            yy=YMAX;
         xx= XD*i;
```

```
            x=XO+xx;
            y=YO-yy;
            x1=XO+XD+xx;
            y1=YO;
            _rectangle( _GFILLINTERIOR,x,y,x1,y1);
    }
}
cls_histogram()
{
    unsigned char mask0[8]={0,0,0,0,0,0,0,0};
    unsigned char mask1[8]={0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};

    x = XO+1;
    y = YO-1;
    x1= XMAX;
    y1= YO-YMAX;
    _setcolor( 0) ;
    _setfillmask( mask1);
    _rectangle( _GFILLINTERIOR,x,y,x1,y1);
    _setcolor(16);
}

histogram_dsp()
{
    char   key;

/* set video mode */
    _setvideomode( mode );

/* Draw axis */
    y=YO-YMAX;
    _moveto(XO,y);
    _lineto( XO,YO );
    x=XO + XMAX;
    _lineto(x,YO);

    for(i=1; i<129; i++)
    {
            x=XO+XD*i;
            y=YO;
            _moveto(x,y);
            x1=x;
            y1=YO+YD;
            _lineto(x1,y1);
    }
/* draw x-axis ruler */
    for(i=0; i<17; i++)
    {
            x = XO + 8*XD*i;
            _moveto(x,YO);
            y = YO + 2*XD;
            _lineto(x,y);
            x = x/8;
            _settextposition(27,x);
            sprintf(title,"%3d", 8*i);
            _outtext( title);
    }
/* print titles */
```

```
   _settextposition(2,35);
   _outtext(" Histogram ");
   _settextposition(28,35);
   _outtext(" Pixel Value ");
   _settextposition(4,0);
   _outtext( " Relative Number of Occurrences of Pixel value");

/* Data */


/* Look for Max value */

   max_pixel_num=pixel_num[0];
   max_pixel_point=0;
   for( i=1; i< MAX_P; i++)
   {
         if(pixel_num[i]> max_pixel_num)
         {
            max_pixel_point=i;
            max_pixel_num = pixel_num[i];
         }
   }
   _settextposition(5,30);
   sprintf(title,"Max Number=%lu",max_pixel_num);
   _outtext( title );
   _settextposition(6,30);
   sprintf(title,"Intensity value=%u",max_pixel_point);
   _outtext( title );

/* Calculate the scale of histogram */
   max_y_axis=max_pixel_num;
   hscale = max_pixel_num/YMAX;


/* plot histogram */
   y_ruler();
   histogram();
   while( (key=getch()) !=27)
   {
         switch(key)
         {
            case '+':
                  hscale= hscale>>1;
                  max_y_axis=max_y_axis>>1;

                  y_ruler();
                  histogram();
                  break;
            case '-':
                  hscale= hscale*2;
                  max_y_axis=max_y_axis*2;

                  y_ruler();
                  cls_histogram();
                  histogram();
                  break;
```

```
        }
    }
        _setvideomode( _DEFAULTMODE );
    return(0);
}
```

File Name: VBLST.C

```
/* VBLST.C  Video Blaster Card Functions                    */
#include<conio.h>
#include<dos.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<graph.h>
#include "myvblst.h"


/* Check if VBLSTDRV is installed and find the interrupt vector */

int FindVblstdrv(char *szDriverSig)
{
        void    far *IpIntrVect;
        int     i,found=0,wSiglen=strlen(szDriverSig);

   /*check from 80h to BFh */
   for(i=0x80;i<0xC0 && !found; i++)
   {
        IpIntrVect=(void far *)_dos_getvect(i);
        FP_OFF(IpIntrVect)=SIG_OFFSET;
        if(!_fstrnicmp(IpIntrVect,(char far *)szDriverSig,wSiglen))
        {
          found=i;
          break;
        }
   }
   return(found);
}

/* Call VBLSTDRV with registers set with appropriate value     */

int CallVblstdrv(union REGS *regs,struct SREGS *segregs)
{
        static char *szErrMessage[]={
                        "Video Blaster driver is busy",
                        "VIDEOBLST environment variable is not present or"
                        "incorrectly set",
                        "Video Blaster is not found at the specified IO"
                        "address",
                        "Interrupt not detected using the specified interrupt"
                        "number",
                        "Function specified is invalid",
                        "Parameter passed to the function is out of range",
                        "Function is invalid in a particular Fit-window mode",
                        "unable to open PCVIDEO.CFG file",
                        "PCVIDEO.CFG file is not a valid Video Blaster"
                        "configuration file"
                        };
```

```
        int     fRetVal=TRUE;

  int86x(wVBIntrNum,regs,regs,segregs);

  if (regs->x.cflag)
  {
        printf("ERROR: %s\n",szErrMessage[regs->x.ax]);
        fRetVal=FALSE;
  }
  return(fRetVal);
}

/* DISALES VIDEO DRIVER AND EXIT                              */

void DisableDriver()
{
  regs.x.bx=VB_TERMINATE;
  CallVblstdrv(&regs,&segregs);
  printf("\nDriver terminated\n");

  regs.x.ax=3;
  int86(VIDEO_INT,&regs,&regs);

  exit(0);
}

/* Input handler                                              */

void    InputHandler()
{
        int     Key;
  while((Key=getch())!=ESCAPE)
  {
        switch(Key)
        {
          case CRET:
                        regs.x.bx=VB_GETDSPSTATUS;
                        if (!CallVblstdrv(&regs,&segregs))
                          DisableDriver();
                        regs.x.ax^=1;       /* OR */
                        regs.x.bx=VB_SETDSPSTATUS;
                        if (!CallVblstdrv(&regs,&segregs))
                          DisableDriver();
                        break;
          case 'F':
                        regs.x.ax=1;
                        regs.x.bx=VB_SETFREEZE;
                        if (!CallVblstdrv(&regs,&segregs))
                          DisableDriver();
                        break;
          case 'U':
                        regs.x.ax=0;
                        regs.x.bx=VB_SETFREEZE;
                        if (!CallVblstdrv(&regs,&segregs))
                          DisableDriver();
                        break;
        }
  }
```

```
}

/* Create a Video Window          */
void DisplayVideoWindow()
{

   unsigned       far *vidPtr=(unsigned far *)0xB8000000L;
                  int x,y;
   /*Draw Window */
   for(y=0;y<WIN_HEIGHT;y++)
         for(x=0;x<WIN_WIDTH;x++)
            *(vidPtr+y*80+x)=0x7020;

   /* Set color key                  */
   regs.x.bx=VB_SETCOLKEY;
   regs.x.ax=7;
   if (!CallVblstdrv(&regs,&segregs))
         DisableDriver();

   /* Set  position */
   regs.x.bx=VB_SETVIDPOS;
   regs.x.ax=VID_XORG;
   regs.x.dx=VID_YORG;
   if (!CallVblstdrv(&regs,&segregs))
         DisableDriver();

   /* Set width and Height    */
   regs.x.bx=VB_SETVIDSIZE;
   regs.x.dx=VID_WIDTH;
   regs.x.ax=VID_HEIGHT;
   if (!CallVblstdrv(&regs,&segregs))
         DisableDriver();

   /* Set Video display on           */
   regs.x.bx=VB_SETDSPSTATUS;
   regs.x.ax=1;
   if (!CallVblstdrv(&regs,&segregs))
         DisableDriver();

}

void DisplayInfo()
{
   /* set cursor position        */
   regs.h.ah=2;
   regs.h.bh=0;
   regs.h.dh=WIN_HEIGHT+1;
   regs.h.dl=0;
   int86(VIDEO_INT,&regs,&regs);


   printf("VBLSTDRV Interrupt Number  :%Xh\n",wVBIntrNum);
   regs.x.bx=VB_GETVERSION;
   if (!CallVblstdrv(&regs,&segregs))
         DisableDriver();
   printf("VBSTDRV Version :%d.%2d\n",regs.h.ah,regs.h.al);
   regs.x.bx=VB_GETSUPPORT;
   if (!CallVblstdrv(&regs,&segregs))
```

```
        DisableDri...();
  printf("Base I/O Port Address  :%Xh\n",regs.x.ax);

  regs.x.bx=VB_GETBASEADDR;
  if (!CallVblstdrv(&regs,&segregs))
        DisableDriver();
  printf("Video Base Address  :%X00000h\n",regs.x.ax);

  regs.x.bx=VB_INTRNUM;
  if (!CallVblstdrv(&regs,&segregs))
        DisableDriver();
  printf("Hardware Interrupt Number: %d\n",regs.x.ax);

  printf("\nPress<F>reeze/<U>nfreeze  <CR> to TURN ON/OFF VIDEO"
        "<ESC> to continue");
}

frame_dsp()
{
        regs.x.bx=0x119;
        regs.x.ax=0;
        regs.x.dx=0;
        if ( !CallVblstdrv(&regs,&segregs))
          DisableDriver();

}

get_image()
{
  int   keys;
  if (wVBIntrNum=FindVblstdrv(DRIVER_SIG))
  {
        /* Clear Screen */
//      regs.x.ax=3;
//      int86(VIDEO_INT,&regs,&regs);


        regs.x.bx=VB_INITIALIZE;
        if (!CallVblstdrv(&regs,&segregs))
          DisableDriver();
        printf("Driver initialzed! \n");

//      while( !ClickOrPress() )
        DisplayVideoWindow();
//      DisplayInfo();
  }
  else
    {
        printf("VBLSTDRV not installed!");
        exit(2);
        }
}

void Freeze()
{
  regs.x.ax=1;
  regs.x.bx=VB_SETFREEZE;
  if (!CallVblstdrv(&regs,&segregs))
```

```
   DisableDriver();
}
void UnFreeze()
{
   regs.x.ax=0;
   regs.x.bx=VB_SETFREEZE;
   if (!CallVbIstdrv(&regs,&segregs))
   DisableDriver();
}
get_color()
{
   regs.x.bx = 0x1B;
   regs.x.ax = 4;
   if (!CallVbIstdrv(&regs,&segregs))
   DisableDriver();
   red = regs.x.ax;
   regs.x.bx = 0x1B;
   regs.x.ax = 5;
   if (!CallVbIstdrv(&regs,&segregs))
   DisableDriver();
   green = regs.x.ax;
   regs.x.bx = 0x1B;
   regs.x.ax = 6;
   if (!CallVbIstdrv(&regs,&segregs))
   DisableDriver();
   blue = regs.x.ax;
}
```

File Name: FILEIO.C

```
/*********************************************/
/*                                           */
/*  FILEIO.C  Read or write data file        */
/*                                           */
/*********************************************/
#include <stdio.h>
#include <dos.h>
#include "mv.h"

file_io( rorw)
   char   rorw;
   {
   FILE *io;
   char *filename;
   char buffer[22] ={ 20};
//   *buffer = 16;
   /* Set Cursor position */
   regs.h.ah=2;
   regs.h.bh=0;
   regs.h.dh=13;
   regs.h.dl=0;
   int86(0x10,&regs,&regs);
   /* Input data file name */
   cprintf("File Name <%c> --> ",rorw);
   filename = _cgets( buffer );
   switch( rorw )
   {
   case 'w' :
```

```
if(access(filename,0)==0)
 {
 cprintf("The file %s already exists. Want to overwrite? (y/n)",filename);
 if(getch()!='y')
       return(5);
 }
if((io=fopen(filename,"w"))==NULL)
 return(6);
/* Write data into data file */

fprintf(io,"%d",sample_num);
for ( i=0;i<sample_num;i++)
{
fprintf(io," %d",sample[i].diameter);
fprintf(io," %d",sample[i].mean_grayscale);
fprintf(io," %hu ",sample[i].area);
fprintf(io," %hu",sample[i].perimeter);
fprintf(io," %f",sample[i].compactness);
fprintf(io," %f",sample[i].ratio1);
fprintf(io," %f",sample[i].ratio2);
fprintf(io," %f",sample[i].ratio3);
fprintf(io," %f",sample[i].ratio4);
}
fclose(io);
break;

case 'r' :
if((io=fopen(filename,"rt"))==NULL)
 return(7);
/* Read data frome data file */
fscanf(io,"%d",&sample_num);
for ( i=0; i<sample_num; i++ )
       {
fscanf(io," %d",&sample[i].diameter);
fscanf(io," %d",&sample[i].mean_grayscale);
fscanf(io," %hu ",&sample[i].area);
fscanf(io," %hu",&sample[i].perimeter);
fscanf(io," %f",&sample[i].compactness);
fscanf(io," %f",&sample[i].ratio1);
fscanf(io," %f",&sample[i].ratio2);
fscanf(io," %f",&sample[i].ratio3);
fscanf(io," %f",&sample[i].ratio4);
       }
fclose(io);
break;
case 't':
       if((io=fopen(filename,"rt"))==NULL)
         return(7);
/* Read template data frome data file */
       fscanf(io,"%d",&mean_sample.diameter);
       fscanf(io,"%lf",&std_dev_sample.diameter);
       fscanf(io,"%f",&mean_sample.ratio4);
       fscanf(io,"%lf",&std_dev_sample.ratio4);
       fscanf(io,"%d",&threshold_val);
       fscanf(io,"%f",&x_delt);
       fclose(io);
       break;
}
```

```
}
```

Head Files: MV.H, VBLST.H,MVMENU.H and MVMOUSE

```
// MV.H   Macros, globlal viarables, structure for MV          //

#define    DELT   10
#define    XWIDTH 320
#define    ESCAPE 0x1B
#define    NSAMPLE 100
// Register
union      REGSregs;
struct     SREGS       segreg

   unsigned long addr;
   union   _REGS inregs.intregs;
   struct  _SREGS sr
// Memory allocatin
   unsigned char __huge *BufSeg;
   unsigned char __far  *BitMap;
   unsigned char __far  *Binary_data;

// Pixel viarables
   long     pixel_num[128],avg_grayscale;
   unsigned char  pixel_val,threshold_val,cut_point1,cut_point2,cut_point3;
   int      j,i,mean_grayscale,diameter;
   unsigned short  area,area0,p4,p8,perimeter;
   float    pi,ratio1,ratio2 ratio4,x_delt;
// image display
   unsigned short  lin,col,l
   int      lin_st,lin_end,    ,col_end;
   unsigned char  col_bit,mask,byte_val;
// Numbers
   int   learn_num,sample_num;
// Flag
   int   classify_flag;

struct GDT
{
   unsigned char    zero1[16];
   unsigned short   s_length;
   unsigned char    s_ad0;
   unsigned char    s_ad1;
   unsigned char    s_ad2;
   unsigned char    s_acc_right;
   unsigned short   zero2;
   unsigned short   d_length;
   unsigned char    d_ad0;
   unsigned char    d_ad1;
   unsigned char    d_ad2;
   unsigned char    d_acc_right;
   unsigned char    zero3[18];
}GDT1,GDT2;

struct DATABASE
{
   int      diameter;
   int      mean_grayscale;
```

```
  unsigned short  area;
  unsigned short  perimeter;
  float     compactness;
  float     ratio1;
  float     ratio2;
  float     ratio3;
  float     ratio4;
} sample[NSAMPLE],mean_sample;

struct DEVIATION
{
  double          diameter;
  double          mean_grayscale;
  double          area;
  double          perimeter;
  double          compactness;
  double          ratio1;
  double          ratio2;
  double          ratio3;
  double          ratio4;
} std_dev_sample;

struct    Membership_function
{
  float  diameter;
  float  ratio;
} membership,u1,u2;

/*****************************************************************/
/* MVBLST.H     Fuction protypes, macros, structure etc for VBLST */
/*****************************************************************/

/* Include only once */
#ifndef MVBLST_H
#define MVBLST_H

#define VIDEO_INT         0x10
#define TRUE              1
#define FALSE             0

#define DRIVER_SIG        "VBLAST"
#define SIG_OFFSET        0x103
#define VB_GETVERSION           0
#define VB_INITIALIZE     0x01
#define VB_TERMINATE            0x02
#define VB_GETIOPORT      0x03
#define VB_GETBASEADDR          0x04
#define VB_GETLOGWIN            0x05
#define VB_INTRNUM        0x06
#define VB_POLLSIGNAL           0x07
#define VB_GETFITWIN      0x08
#define VB_GETCROP        0x09
#define VB_GETFREEZE      0x0A
#define VB_GETSIGNAL      0x0B
#define VB_GETACQADDR           0x0C
#define VB_GETCROPXY            0x0D
#define VB_GETCROPSIZE          0x0E
#define VB_GETSCALE       0x0F
```

```
#define VB_VIDINCOLOR          0x10
#define VB_GETSOURCE           0x11
#define VB_GETDSPSTATUS    0x12
#define VB_GETDSPMODE          0x13
#define VB_GETVIDAREA          0x14
#define VB_GETVIDPOS   0x15
#define VB_GETVIDSIZE          0x16
#define VB_GETCOLKEY           0x17

#define VB_SETBASEADDR         0x104
#define VB_SETLOGWIN           0x105
#define VB_SETFITWIN   0x108
#define VB_SETCROP     0x109
#define VB_SETFREEZE   0x10A
#define VB_SETSINGLE   0x10B
#define VB_SETACQADDR          0x10C
#define VB_SETCROPXY           0x10D
#define VB_SETCROPSIZE         0x10E
#define VB_SETSCALE    0x10F
#define VB_SETSOURCE           0x111
#define VB_SETDSPSTATUS    0x112
#define VB_SETDSPMODE          0x113
#define VB_SETVIDAREA          0x114
#define VB_SETVIDPOS   0X115
#define VB_SETVIDSIZE          0x116
#define VB_SETCOLKEY           0x117

#define CRET                   0x0D
#define ESCAPE                 0x1B
#define SPACE                  0x20

#define WIN_HEIGHT     12
#define WIN_WIDTH      40
#define VID_XORG       4
#define VID_YORG       4
#define VID_HEIGHT     550
#define VID_WIDTH      465

        int             wVBIntrNum;
        union   REGS    regs;
        struct SREGS    segregs;
// Colors
   int    red,green,blue,brightness,saturation,contrast,hue;
/* public functions */
int FindVblstdrv(char *szDriverSig);
int CallVblstdrv(union REGS *regs,struct SREGS *segregs);
void DisableDriver();
void InputHandler();
void Freeze();
void UnFreeze();
void DisplayVideoWindow();
void DisplayInfo();
get_image();
get_color();
#endif /* MVBLST_H */


/* MVMENU.H */
```

APPENDIX B

```
/* Function prototypes, macros, structure, and global variables for
 * Menu and related functions.
 */

/* Include only once */
#ifndef MVMENU_H
#define MVMENU_H

#define TRUE  1
#define FALSE 0

/* Sample key codes for getkey. Additional codes in the same format may
 * be added.
 */
#define U_UP   0x0148    /* Unshifted */
#define U_DN   0x0150
#define U_LT   0x014b
#define U_RT   0x014d
#define S_UP   0x0248    /* Shifted */
#define S_DN   0x0250
#define S_LT   0x024b
#define S_RT   0x024d

#define N_PLUS  0x014e    /* PLUS and MINUS on numeric keypad */
#define N_MINUS 0x014a

#define ENTER  13        /* ASCII */

/* Action codes for getkey */
enum WAITACTION { NO_WAIT, WAIT, CLEAR_WAIT };

/* Text output colors. Note that monochrome can only use _TBLACK,
 * _TWHITE, _TBRIGHTWHITE, and _TUNDERLINE. Graphics black-and-white
 * can only use the first three of these. The first eight colors
 * can be used as background colors (although they may need to be
 * cast to long).
 */
enum TEXTCOLORS
{
   _TBLACK,      _TBLUE,       _TGREEN,      _TCYAN,
   _TRED,        _TMAGENTA,    _TBROWN,      _TWHITE,
   _TGREY,       _TLIGHTBLUE,  _TLIGHTGREEN, _TLIGHTCYAN,
   _TLIGHTRED,   _TLIGHTMAGENTA, _TLIGHTYELLOW, _TBRIGHTWHITE,
   _TUNDERLINE = 1
};

/* Structure and global variable for menu attributes */
typedef struct _MENU
{
   int    fgBorder, fgNormal, fgSelect, fgNormHilite, fgSelHilite;
   long   bgBorder, bgNormal, bgSelect, bgNormHilite, bgSelHilite;
   int    fCentered;
   unsigned char  chNW, chNE, chSE, chSW, chNS, chEW;
} MENU;
extern MENU mnuAtrib;

/* Structure and maximum length for menu items */
#define MAXITEM 20
```

APPENDIX B

```
typedef struct _ITEM
{
   int    iHilite;
   char   achItem[MA\' ' 2M];
} ITEM;

/* Public menu, output, and input functions */
int  Menu( int row, int col, ITEM aItem[], int iCur );
int  EventLoop( int row, int col, ITEM aItem[], int iCur, int cItem,
           int cchItem, int acchItem[], char achHilite[] );
void Bo .( int row, int col, int rowLast, int colLast );
unsigned GetKey( int fWait );
void _outchar( char out );
int  ClickOrPress( void );

#endif /* MVMENU_H */


/* MVMOUSE.H            */
/* Include file for mouse calls. */

/* Mouse events */
#define LEFT_DOWN  0x01      /* 0000 0010 Left button pressed   */
#define RIGHT_DOWN 0x02      /* 0000 1000 Right button pressed  */
#define MIDDLE_DOWN 0x04     /* 0010 0000 Middle button pressed */

/* Mouse event structure */
typedef struct _EVENT
{
   short    x, y;
   unsigned fsBtn;
} EVENT;

/* Mouse pointer shape union containing structures for graphics and text */
typedef union _PTRSHAPE
{
   struct
   {
      unsigned char atScreen;
      unsigned char chScreen;
      unsigned char atCursor;
      unsigned char chCursor;
   } t;
   struct
   {
      unsigned xHot, yHot;
      unsigned afsPtr[32];
   } g;
} PTRSHAPE;

/* Values for SetPtrVis function */
typedef enum _PTRVIS { SHOW = 1, HIDE } PTRVIS;

/* Public mouse functions */
int MouseInit( void );
 int GetMouseEvent( EVENT __far *pEvent );
int GetPtrPos( EVENT __far *pEvent );
int SetPtrPos( short x, short y );
```

APPENDIX B

```
int SetPtrVis( PTRVIS pv );
int SetPtrShape( PTRSHAPE far *ps );
```

Author: Lu, Ning.
Name of thesis: A knowledge-based machine vision system.

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2015

LEGALNOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page)for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.