Buletinul Ştiinţific al Universităţii din Baia Mare
Seria B, Matematică-Informatică, vol.IX(1993), 129-138

IMAGE PROCESSING ROUTINES FOR IBM PC FAMILY COMPUTERS

Ovidiu COSMA

One of the most important elements that have contributed to the success of personal computers is computer graphics, because it is a flexible way of communication, more adapted to the human senses.

There are a great number of computer programs capable of producing more and more complex images, but most of them have two major limitations:

-They work properly only on powerful computers;

-There are (excepting WINDOWS applications) very few possibilities of including the generated images in application programs.

The procedures in this article solve the second problem, and they run on almost any IBM PC computer. The images are read from "TIF" files, then transformed and saved into a more flexible graphic format. The TIF format has been developed together with the scanning devices, and unfortunately there are a number of variants for this standard, but the differences between them are unimportant. The next figure shows the black & white TIF files structure.

The header contains a specific signature that includes the number of columns occupied by the screen image. The data area keeps the image, each bit corresponds to a screen pixel. (1 = light, 0 = dark).

-SIGNATURE- Nu. of columns -SIGNATURE-
DATA

The proposed graphic format is illustrated in the next figure.

The first block contains the image dimensions. The second block contains the drawing colors and the third one contains the color palette definitions for the drawing colors. The data area contains the picture, each group of 4 bits correspond to a screen pixel that can be drawn with one of 16 possible colors. The "format" block is used to indicate the type of picture contained in the data area.

Nu. of columns Nu. of lines
Drawing colors
Color palettes
Format
DATA

Thus the data block can hold color images or standard TIF data.

This flexible file structure allows facile changes to the image colors and palettes.

For the conversion process, the TIF images have been divided into 4 x 4 pixel regions. Next, for each of this regions a 4 bits color code is stored in the data area of the new file. The color codes are in direct correspondence with the shade of the region, or the number of dark points in region.

Here is for example a function that does the transformation of a TIF file into the new format:

```
/* TIF FILE CONVERSION */
#define NR_MAX_LIN 350     /* maximum number of lines */
#define LUNG_MAX_LIN 320   /* maximum length of lines */
#define LUNG_ANTET_IN 138  /* TIF file header length */
#define DIM_PIX 4          /* pixel matrix dimmension */
int nr_linii, nr_coloane;  /* nr. of lines, nr. of columns*/
void conv_fis(void)
{
   char intrare[LUNG_MAX_LIN][DIM_PIX]; /* input buffer */
   char iesire[LUNG_MAX_LIN];           /* output buffer */
   char buf[DIM_PIX * LUNG_MAX_LIN];
   int nr_col, i, j, k, bytes;
   char punct1, punct2;
   char *date;
   /* --- open the files */
   open_files();
   /* --- read the TIF file header */
   if(lseek(handle_dest, 40L, SEEK_SET) == -1 ¦¦
   (bytes = _read(handle_sursa, buf, LUNG_ANTET_IN)) == -1 ¦¦
   bytes == 0)
      some_error();
   /* --- read the number of columns from TIF header */
   nr_col = ( *(buf+30) >> 3 & 0x1f ) ¦ *(buf+31) << 5;
   nr_linii = 0;
   nr_coloane = min(LUNG_MAX_LIN, nr_col)/DIM_PIX;
   format = 0xff;
   do{
      /* --- read DIM_PIX lines from the TIF file */
      if ((bytes = _read(handle_sursa, buf, DIM_PIX * nr_col))
      == -1 ¦¦ bytes == 0) {
         printf("\nReady file");
         return;
      }
      /* --- convert the read lines */
      date = buf;
      for( j = 0; j < DIM_PIX; j++ )
         for( i = 0; i < nr_col; intrare[i++][j]=*date++);
      for( i = 0; i < nr_coloane*DIM_PIX; i++){
         punct1 = punct2 = 0;
         for( k = 0; k < DIM_PIX; k++ )
            for( j = 0; j < DIM_PIX; j++ ){
               if(intrare[i][j] & 0x10 && punct1 < 0xf)
                  punct1++;
               if(intrare[i][j] & 0x1 && punct2 < 0xf)
                  punct2++;
               intrare[i][j] >>= 1;
            }
         iesire[i] = punct1 << 4 ¦ punct2 ;
      }
      /* --- write maximum LUNG_MAX_LIN bytes in the new file */
      if ((bytes = _write(handle_dest, iesire,
      nr_coloane*DIM_PIX)) == -1)
         some_error();
      nr_linii++;
   }while(nr_linii < NR_MAX_LIN);

   /* the conversion is ready */
   /* write the new file header */
   if( lseek(handle_dest, 36L, SEEK_SET) == -1L ¦¦
   _write(handle_dest, &format, 1) != 1)
      some_error();
   if( lseek(handle_dest, 0L, SEEK_SET) == -1L ¦¦
   _write(handle_dest, &nr_coloane, 2) != 2 ¦¦
   _write(handle_dest, &nr_linii, 2) != 2)
      some_error();
   for( i = 0; i < 16; i++ ){
      cul[i] = i;/* reset the colors */
      pal[i] = 0xff;
   }
   if( lseek(handle_dest, 4L, SEEK_SET) == -1L ¦¦
   _write(handle_dest, cul, 16) != 16 ¦¦
   _write(handle_dest, pal, 16) != 16)
      some_error();
   close(handle_dest);
   close(handle_sursa);
}
```

With this elements, is easy to build an application program for controlling the image appearance by modifying the color and palette definitions in the header of the file. I have personally used such a program for coloring the images produced with a black & white scanner.

The number of colors can be extended by enlarging the regions used to divide the TIF file.

The following instructions illustrate a simple method that can be used for displaying the converted images.

```
    int stat;
    init_gr_mode(); /* initialize the graphic mode */
    depl_x = SX;  /* indicate the position of the image on screen */
    depl_y = SY;
    /* ---memory allocation for the header and data areas */
    if( (stat = allocmem(NR_PAR_POZA, &segp) ) != -1 ¦¦
    (stat = allocmem(NR_PAR_ANT, &segp_antet) ) != -1)
       alloc_err(stat);
    }
    nume_fis=FILE_NAME; /* name of the image file */
    afis_poza(); /* display the image on screen*/
```

Here is the source code for afis_poza:

```
_text      SEGMENT public 'CODE'
```

```
assume    CS:_text,DS:_data
mask_cul   db 16 dup(?);color masks
def_cul    db 16 dup(?);color definitions
nr_pixel   db ?;pixel number
nr_col_fi  db ?;total number of columns
col_de_af  db ?;number of columns to display next
nr_car_pag dw 2 dup(?);number of characters / page
nr_pag      db ?;number of pages
format     db ?
depdat      equ 40;bytes to start of picture
;--------------------+
  public _mod_10_ega
;--------------------+
_mod_10_EGA:
     mov  ah,0
     mov  al,10h
     int  10h;set 10 EGA mode
;-------------------+
  public _wr_mode_2
;-------------------+
_wr_mode_2:
     mov  dx,3ceh
     mov  al,5
     out  dx,al
     inc  dx
     mov  al,2
     out  dx,al;set write mode 2 EGA
     ret
;-------------------+
  public _afis_poza
;-------------------+
_afis_poza:
     push ds;context saving
     push es
     push si
     push di
     mov  ah,3dh;code to open file
     mov  al,0;open for read
 extrn _nume_fis:WORD
     mov  dx,offset DGROUP:_nume_fis
     int  21h
     jc   eroare;exit in case of error
     mov  filehl[0],ax;save file handle
 ;read file header
     mov  bx,filehl[0]
 extrn _segp_antet:WORD
     mov  ax,_segp_antet
     mov  ds,ax;header segment address
     mov  ah,3fh
     mov  dx,0
     mov  cx,depdat;nr of header bytes
     int  21h
     jc   eroare
     call init_afis
     test format,2
     jnz  cit_im;read image from file
     mov  ah,3eh
     int  21h;close file, release file handle
     jmp  sfirsit;only palettes setup, no image
cit_im:
extrn _segp:WORD
     mov  ax,_segp
     mov  ds,ax;free memory address
     xor  ah,ah
     mov  nr_pag[0],ah;page number=0
cit_pag:  ;read a 64K page from file
     mov  ah,3fh
     mov  dx,0
     mov  cx,0ffffh;nr. of bytes to read
     int  21h
     jc   eroare; in case of error
     mov  cx,bx;save the file handle
     mov  bh,0
     mov  bl,nr_pag[0]
     sal  bl,1;nr.of characters=word(2 bytes)
     mov  nr_car_pag[bx],ax;save nr. of char in page
     inc  nr_pag[0];increment the number of page
     cmp  ax,0ffffh
     jz   mai_date;64k read => there are more pages
;file read is finished
     mov  ah,3eh
     mov  bx,cx;=file handle
     int  21h;close file, release file handle
     jc   eroare; in case of error
     mov  ax,_segp;restore DS
     mov  ds,ax;free memory address
     mov  nr_pag[0],0;page number = 0
     jmp  desenare
eroare: ;read error
     mov  ah,2
     mov  dl,40h
     int  21h
     jmp  sfirsit
mai_date:
     dec  nr_car_pag[bx]
     mov  ax,ds
     add  ax,1000h
     mov  ds,ax;next segment
     mov  bx,cx;restore file
     jmp  cit_pag
desenare:
;start drawing, DS:0=start of image
     mov  al,5h;80 -> 4
extrn _depl_y:BYTE
     mul  _depl_y;AX=80*y, y=position on screen
     add  ax,0a000h;video buffer start
     mov  es,ax
extrn _depl_x:BYTE
```

```
          mov  al,_depl_x
          xor  ah,ah
          mov  di,ax;di=nr.of points, ES:DI=screen start
          mov  si,0;DS:SI=image start
          test format,1;test picture format
          jnz  nu_e_graf
          call grafica; graphics display
          jmp  sfirsit
nu_e_graf:
          call poza; picture display
          jmp  sfirsit
;---------+
   grafica:
;---------+
          mov  cl,def_cul[1]
          mov  ch,nr_col_fi[0]
          mov  bx,nr_car_pag[0]
afis_urm:
          mov  ah,[si]
          not  ah
          call masc_pix
          mov  ah,es:[di]
          mov  es:[di],cl;display
          inc  si;next point in buffer
          dec  bx
          jnz  nu_sfi
          ret
nu_sfi:
          inc  di;next point on screen
          dec  ch
          jnz  afis_urm
          call lin_urm_ec; next line on screen
          mov  ch,nr_col_fi[0]
          jmp  afis_urm
          ret
;------+
   poza:
;------+
          mov  bl,00h;start with the first mask
st_mask:
          mov  mask_cul[bx],0
          inc  bl
          cmp  bl,10h
          jnz  st_mask;delete masks
          mov  bh,0;used by testcul and gen_mask routines
          mov  ch,4;nr of bytes for 8 pixels
          mov  nr_pixel[0],80h;first of 8 pixels
trat_8_pix:
          call gen_mask;generate masks for the 2 points
          call pt_urm_fis;go to next point in buffer
          jnc  nu_sfirsit
          ret;the buffer is over
nu_sfirsit:
          dec  ch
```

```
     jnz  trat_8_pix
;8 points are ready, begins the display of colors
     mov  bl,0fh; bh=0
trat_cul:
     mov  ah,mask_cul[bx];al=color mask
     or   ah,ah
     jz   mask_urm;next mask
     call afis;display, ah=mask bl -> color
mask_urm:
     dec  bl
     cmp  bl,0ffh
     jnz  trat_cul;repeat for all 16 colors
     inc  di;go to next point on screen
     dec  col_de_af[0];--nr of columns to display
     jnz  poza; ready line?
     mov  al,nr_col_fi[0]
     mov  col_de_af[0],al;nr of char.=nr.of columns
     call lin_urm_ec
     jmp  poza;never reach end of screen here
;---------+
  sfirsit:
;---------+
     pop  di; restore the context
     pop  si
     pop  es
     pop  ds
     ret;ready picture display
;-----------+
  init_afis:
;-----------+
     mov  si,0;set up display attributes
     mov  al,[si]
     mov  ah,0
     mov  nr_col_fi[0],al;buffer column number
     mov  col_de_af[0],al;columns to display
     mov  al,[si+36d];picture format
     mov  format,al
     mov  si,4;prepare the colors
     mov  ax,cs
     mov  es,ax;not necessary because es=cs
     mov  di,offset def_cul
     mov  cx,8
 rep  movsw;copy the image colors
     mov  si,0;set up color palettes
 init_pal:
     test BYTE PTR[si+20],80h;valid combination?
     jnz  nuschimb
     call set_pal
 nuschimb:
     inc  si
     cmp  si,16
     jnz  init_pal
     mov  dx,3dah;display enable
     in   al,dx
```

```
    mov  dx,3c0h
    mov  al,20h
    out  dx,al
    ret
;---------+
  set_pal:
;---------+
    mov  dx,3dah
    in   al,dx
    mov  dx,3c0h
    mov  ax,si;culor number, MSB = 0
    out  dx,al
    mov  al,[si+20]
    out  dx,al
    ret
;-----------+   in: BH = 0
  pt_urm_fis:  ;out: DS:SI = next point buffer address
;-----------+        CY = 1 if end of buffer
    mov  bl,nr_pag[0];bl = page number, bh = 0
    sal  bl,1;nr of page characters=word(2bytes)
    mov  ax,nr_car_pag[bx];ax=nr of char. in page
    cmp  ax,si
    jz   gata_pag
    inc  si;page buffer starts at 0, ends at fffe
                   ;ffff=free (ffff will be read next)
    ret
gata_pag:
    cmp  ax,0fffeh;nr of char in page = max?
    jnz  gata_fis;small page => is last
    inc  nr_pag[0];page number++
    mov  ax,ds
    add  ax,1000h
    mov  ds,ax;go to next segment
    mov  si,0
    clc;CY = 0 => the buffer isnot over
    ret
gata_fis:
    stc;CY = 1 => the buffer is over
    ret
;-----------+   in: ES = first byte in line
  lin_urm_ec:  ;out: ES = start of next line on screen
;-----------+        DI = 0
    mov  ax,es
    add  ax,5;80=50h, add 5hto the segment reg.
    mov  es,ax;es += 80;go to next line on screen
    mov  al,_depl_x
    xor  ah,ah
    mov  di,ax;pixel address
    ret
;---------+ in: BH = 0, SI = 2 pixel byte address
  gen_mask:     nrpixel[0] = pixel number
;---------+ generates the color masks
    mov  ah,[si];reads colors of 2 pixels (2x4bits)
    mov  bl,ah;save them
```

```
     mov  cl,4
     ror  bl,cl;first pixel is in MSB
     and  bl,0fh;mask the first 4 bits
     mov  al,nr_pixel[0];pixel nr. (1 to 8)
     or   mask_cul[bx],al;clear bit in color mask
     ror  al,1;go to next pixel
     mov  bl,ah;restore the 2 pixel colors
     and  bl,0fh;the second pixel is in LSB
     or   mask_cul[bx],al
     ror  al,1;prepare for the next pixel
     mov  nr_pixel[0],al;save number of the pixel
     ret
;------+
  afis:  ;in: AH=mask, BL indicates the color
;------+ ;masks the points of different colors
     mov  dx,3ceh
     mov  al,8
     out  dx,al
     inc  dx
     mov  al,ah;al=mask
     out  dx,al
;afiseaza
     mov  al,def_cul[bx];al=color code
     mov  ah,es:[di]
     mov  es:[di],al
     ret
;--------------+
   es_lin_urm:
;--------------+
     mov  ax,es
     add  ax,5;80=50h ->4 = 5
     mov  es,ax
     ret
;------------+
   masc_pix:
;------------+
     mov  dx,3ceh
     mov  al,8
     out  dx,al
     inc  dx
     mov  al,ah
     out  dx,al
     ret
_text       ENDS
DGROUP      GROUP _DATA,_BSS
_DATA       SEGMENT WORD PUBLIC 'DATA'
filehl dw 0
_DATA       ENDS
_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS
     end
```

The assembly language has been chosen for efficiency. This procedure is much faster than the "PUTIMAGE" function in the TURBO