DosBox + Watcom C

Written by Black White

iceman@zju.edu.cn

一、关于Watcom C

Watcom C是一个开源的32位编译器，它的DOS版编译器生成的exe工作在32位保护模式下，指针为32位，完全克服了Turbo C 16位指针只能访问64K内存空间的局限性。

Watcom C的官网为http://www.openwatcom.com

二、把Turbo C图形库移植到Watcom C

Turbo C的图形处理函数工作在低分辨率模式下，不仅显示效果差，而且显卡内存很难用程序控制。因此，本次移植工作不仅是换编译器，更重要的是要在DOS平台下实

现 VESA 图像模式，可以做到在高分辨率模式如 1024x768x24bit 下方便地对显存进行控制。

三、 VESA 图形库支持的文本显示模式

DosBox 刚运行时显卡的工作模式就是文本模式;
要让显卡工作在图形模式的话，必须调用函数 initgraph(); 若要从图形模式返回文本模式，则应该调用函数 closegraph() 或 text_mode() 或 textmode(C80);

```
#include <graphics.h>
#include <stdio.h>
main()
{
   char *p = _vp;
   *p = 'A';
   *(p+1) = RED; /* #define RED 4 */
   *(p+2) = 'B';
   *(p+3) = YELLOW;
   getchar();
}
```

程序 a.c;　在 (0,0) 及 (1,0) 输出字母 A 、B

```
#include <graphics.h>
#include <stdio.h>
main()
{
   char *p = _vp;
   int i;
   for(i=0; i<80*25; i++)
   {
      *p = 'A';
      *(p+1) = RED;
      p+=2;
   }
   getchar();
}
```

程序 b.c;　输出 2000 个 A

四、VESA 图形库支持的显示模式

VESA 图形库一共支持以下显示模式:

VESA_320x200x8bit　以下为图形模式

VESA_320x200x24bit

VESA_320x200x32bit

VESA_640x480x8bit

VESA_640x480x32bit

VESA_800x600x8bit

VESA_800x600x24bit

VESA_800x600x32bit

VESA_1024x768x8bit

VESA_1024x768x24bit

VESA_1024x768x32bit　以上为图形模式

TEXT_80x25x16bit　　文本模式

具体请参考 graphics.h 中 enum MODE_SUPPORTED 枚举类型的定义。

五、VESA 图形库支持的三种颜色模式

1. 8bit 颜色模式

该模式下，每个点对应显卡中的一个字节，每个点的颜色可以达到 2^8。其中前 16 种颜色[0,15]的定义如下：

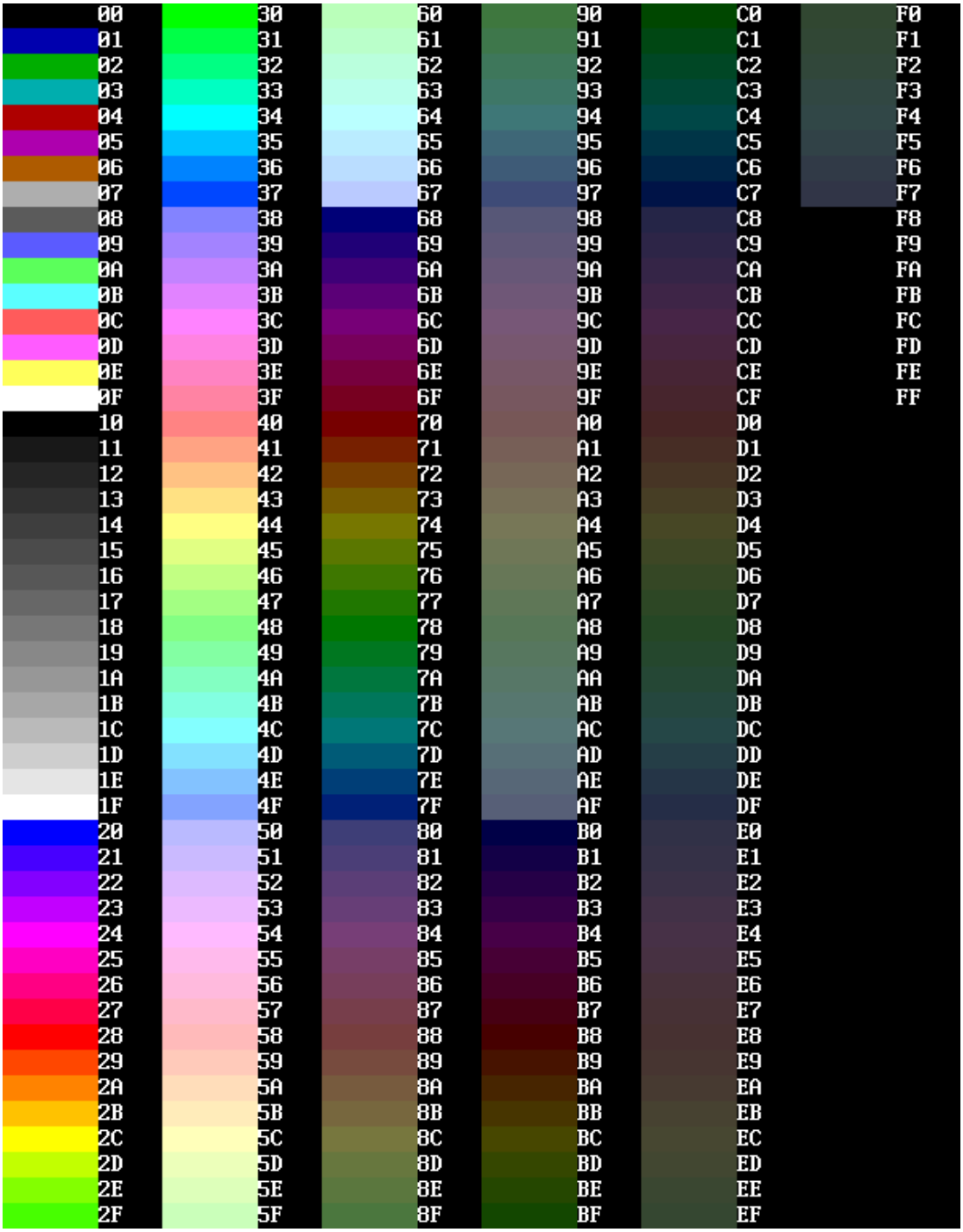
```
enum COLORS
{
   BLACK,        /* dark colors */
   BLUE,
   GREEN,
   CYAN,
   RED,
   MAGENTA,
   BROWN,
```

以

```
#include <graphics.h>
#include <stdio.h>
main()
{
   int driver=0, mode=VESA_1024x768x8bit;
   int i, j, x, y;
   char t[10];
   initgraph(&driver, &mode, "");
   x = 0;
   y = -16;
   for(i=0; i<256; i++)
   {
      setcolor(i);
      y += 16;
      if(y >= _height)
      {
         y = 0;
         x += 100;
      }
      for(j=0; j<16; j++)
      {
         line(x, y+j, x+60-1, y+j);
      }
      setcolor(WHITE);
      sprintf(t, "%02X", i);
      outtextxy(x+60, y, t);
   }
   getchar();
   closegraph();
   return 1;
}
```

程序 256color.c

上述程序的输出结果如下图所示：



8bit 模式下如何访问显存？ 假定要画一条红色水平线

连接坐标(300,100)-(600,100)，则代码如下：

```
#include <graphics.h>
main()
{
   int i, x=300, y=100;
   char *p;
                        int       driver=0,
mode=VESA_1024x768x8bit;
   initgraph(&driver, &mode, "");
   p = _vp + y*_width + x;
   /* _vp 是屏幕左上角坐标(0,0)对应的显存地址,
      _width 是屏幕宽度。这两个变量在调用函数
       initgraph()后会自动初始化。
    */
   for(i=0; i<600-300+1; i++)
      *p++ = 4;  /* 相当于*p++ = RED; */
}
```

2. 24bit 颜色模式

该模式下，每个点对应显卡中的 3 个字节，各个点的颜色可以达到 2^{24}。假定要在(0,0)画了一个 RGB 成份为 Red=0x12, Green=0x34, Blue=0x56 的点，程序可以这样写:

```
int driver=0, mode=VESA_1024x768x24bit;
char *p;
initgraph(&driver, &mode, "");
p = _vp;
*p = 0x56;
*(p+1) = 0x34;
*(p+2) = 0x12;
```

请注意，显存中的RGB顺序是逆的。

假定要用上述颜色画一条水平线连接坐标(300,100)-(600,100)，则代码如下：

```
#include <graphics.h>
#include <mem.h>
main()
{
   int i, x=300, y=100;
   char buf[]={0x56,0x34,0x12};
   char *p;
   int driver=0;
   int mode=VESA_1024x768x24bit;
   initgraph(&driver, &mode, "");
```

```
        p = _vp + (y*_width + x) *
            (_color_bits/8);
        /* _vp 是屏幕左上角坐标(0,0)对应的显存地址,
           _width 是屏幕宽度, _color_bits 是每个
点占用内存的 bit 数。这 3 个变量在调用函数
initgraph()后会自动初始化。
         */
        for(i=0; i<600-300+1; i++)
        {
           memcpy(p, buf, sizeof(buf));
           p+=sizeof(buf);
         }
      }
```

3. 32bit 颜色模式

该模式下，每个点对应显卡中的 4 个字节，各个点的颜色可以达到 2^{24}。假定要在(0,0)画一个 RGB 成分为 Red=0x12, Green=0x34, Blue=0x56 的点，程序可以这样写:

```
int driver=0, mode=VESA_1024x768x32bit;
char *p;
initgraph(&driver, &mode, "");
```

```
p = _vp;
*p = 0x56;
*(p+1) = 0x34;
*(p+2) = 0x12;
*(p+3) = 0x00;
```

请注意，4 个字节的最后一个字节都是填 0。点的颜色成份用前 3 个字节表示。

假定要用上述颜色画一条水平线连接坐标(300,100)-(600,100)，则代码如下：

```
#include <graphics.h>
#include <mem.h>
main()
{
   int i, x=300, y=100;
   long color = 0x00123456;
   /* color 在内存中按字节展开后相当于:
   char buf[]={0x56,0x34,0x12,0x00};
    */
   long *p;
```

```
    int driver=0;
    int mode=VESA_1024x768x32bit;
    initgraph(&driver, &mode, "");
    p = (long *)(_vp+(y*_width+x)*
        (_color_bits/8));
    for(i=0; i<600-300+1; i++)
    {
       p[i] = color;
    }
}
```

上述3类颜色模式除了可以用指针进行控制外，均支持画点函数putpixel()、画直线函数line()、画圆函数circle()等。例如，无论在8bit、24bit还是32bit颜色模式下，要画一条水平线连接(300,100)-(600,100)，均可以通过简单地调用line()函数实现：

```
line(300, 100, 600, 100);
```

不过，line()函数的参数中并无画线的颜色值，颜色值需要在调用line()之前先调用函数setcolor()指定。在8bit模式下，可以调用setcolor(RED)或setcolor(4)；24bit以及32bit模式下均可调用

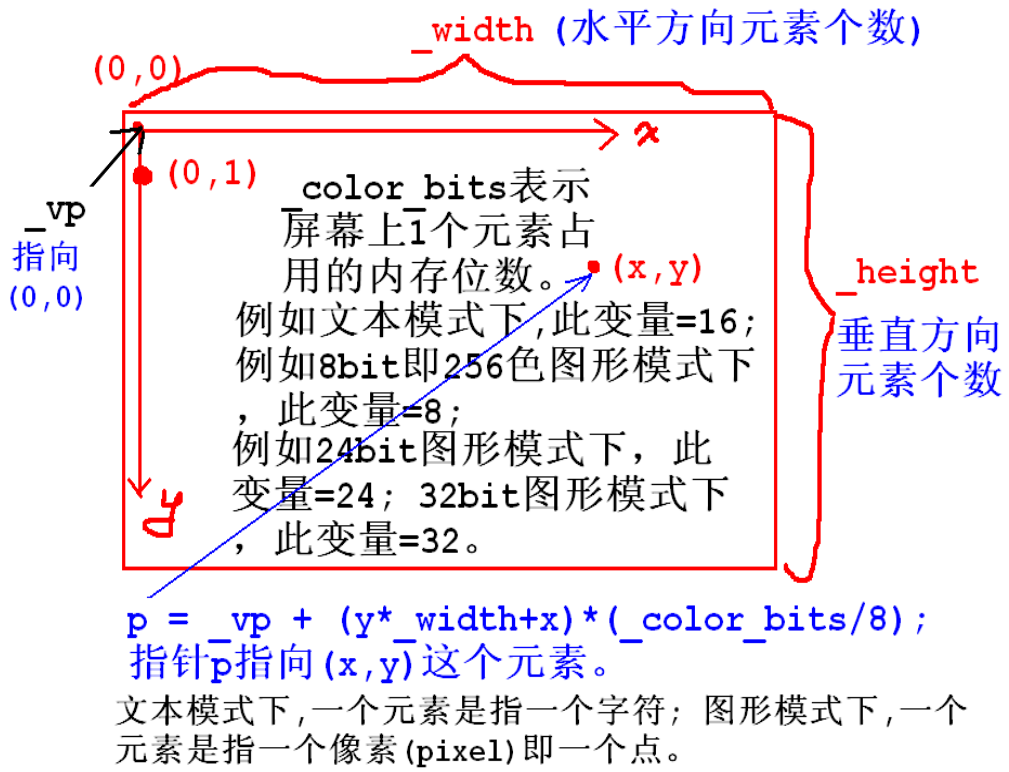
setcolor(0x00FF0000)，其中十六进制数0x00FF0000中的FF表示红色的成份为0xFF即红色的浓度达到最大值。若要在24bit或32bit模式下设定颜色为绿色，则可以调用setcolor(0x0000FF00)；而要设定颜色为蓝色时，则可以调用setcolor(0x000000FF)；setcolor(0x00FFFF00)设定颜色为黄色(红绿的合成色)；setcolor(0x0000FFFF)设定颜色为青色(绿蓝的合成色)；setcolor(0x00FF00FF)设定颜色为紫色(红蓝的合成色)；setcolor(0x00FFFFFF)设定颜色为白色(红绿蓝的合成色)；setcolor(0x00000000)设定颜色为黑色。

六、坐标系统及VESA图形库中定义的全局变量

VESA图形库中定义了如下一些全局变量：

_vp	指向坐标(0,0)的指针
_width	屏幕宽度
_height	屏幕高度
_color_bits	每个像素占用内存的位(bit)数
_visual_page	当前显示画面(前景)对应的页号
_active_page	当前输出画面对应的页号
_back_page	后台画面(背景)对应的页号
_page_gap	后台画面与显示画面的距离
_mode_index	当前视频模式的编号
_mode	当前视频模式的内部编号
_draw_color	前景色
_back_color	背景色
_fill_color	填充前景色
_fill_mask_index	填充掩码
_x	当前 x 坐标
_y	当前 y 坐标

其中 _vp, _width, _height, _color_bits 之间的关系可用下图描述：



_visual_page 以及_active_page 这两个变量可以分别调用以下两个函数设定:

setviualpage()

setactivepage()

请注意_active_page 不一定等于_visual_page，它可以等于_back_page 也可以等于_visual_page。当设

定 _active_page 为 _back_page 时，则 putpixel()、line()、circle()等函数的输出将不会显示在屏幕上，而是隐藏在背景页，接下去再调用 setvisualpage(_back_page)时，原背景页就变成当前的显示页，同时原显示页变成当前的背景页，原先画在背景页的内容将会在屏幕上显示出来，而原先画在显示页的内容则自动隐藏。

_draw_color 由函数 setcolor()指定；_back_color 由 setbkcolor()指定；_fill_color 以及 fill_mask_index 由 setfillstyle()指定。_draw_color 影响 putpixel()、line()、circle()、rectangle()、ellipse()、arc()、drawellipse()等函数输出的点、线、边框之颜色；_back_color 影响 bar()、bar3d()、sector()、pieslice()等函数的填充背景色；_fill_color 影响

bar()、bar3d()、sector()、pieslice()、floodfill()等函数的填充前景色；fill_mask_index 则影响 bar()、bar3d()、sector()、pieslice()、floodfill()等函数所填入的填充物形状如交叉线、水平线、斜线等。

line()、lineto()、linerel()、moveto()、moverel()等函数会影响当前点的坐标(_x, _y)。(_x, _y)是上述函数完成后的终点位置。

有关上述函数的具体用法请参考头文件 graphics.h。

七、VESA 图形库演示程序

VESA 图形库 8bit 以及 24bit 演示程序请见 DosBoxWc.rar 压缩包内的两个工程：

① DosBoxWc\Watcom\project\dm8bit

② DosBoxWc\Watcom\project\dm24bit

八、DosBox+WATCOM C 使用指南

1. 安装

从以下链接下载压缩包并解压缩到任意磁盘：

http://10.71.45.100/bhh/DosBoxWc.rar

解压成功后会生成 DosBoxWc 文件夹，里面的 WATCOM 文件夹包含了 Watcom C 编译器，其中 VESA 图形库的头文件 graphics.h 位于 DosBoxWc\WATCOM\h；库文件 graphics.lib 位于 DosBoxWc\WATCOM\LIB386\DOS。

2. 编写源程序

编辑源程序需要用第三方编辑器如 editplus，源程序请保存到 DosBoxWc\WATCOM\project 文件夹内；若源程序有多个文件，请在 project 内新建一个文件夹进行保存。

3. 编译及运行

（1）双击 DosBoxWc\wc.exe 运行 DosBox

（2）假定要编译 DosBoxWc\WATCOM\project\h.c

则输入以下命令：

```
WCL386  h.c
```

若源程序无语法错误，则上述命令会生成 h.exe；若

源程序有错，请查看屏幕输出的编译信息检查错误原因及错误行号，也可以打开 h.err 查看错误信息。

（3）假定上述步骤成功生成 h.exe，则输入此程序的主名 h 运行：

```
h
```

4. 如何 build 多个.c 文件构成的工程

Watcom C 自带一个 make 工具叫 wmake，支持标准的 make 语法。

DosBoxWc\WATCOM\project\4_3 里面是教材带的俄罗斯方块源码，原先的.prj 工程已经改造成 makefile 文件，其内容如下：

```
russia.exe : Timer.c Drawing.c Util.c Russia.c
 wcl386 /fe=russia.exe Timer.c Drawing.c Util.c Russia.c
```

其中 russia.exe 表示想要生成的 exe 文件，冒号后面所跟的是与该 EXE 相关的.c 文件，当这些文件的日期比.exe 更新时，wmake 会做重新编译的动作，编译的命令由

第 2 行指定。DosBox 刚运行时，会自动进入 project 文件夹，现在假定要编译俄罗斯方块源码，则输入以下命令：

```
cd 4_3
wmake
```

5. 调试

假定要调试 DosBoxWc\WATCOM\project\h.c，则需要对此源程序重新编译，编译命令如下：

```
WCL386 /d2 h.c
```

其中/d2 表示在编译时要包含调试信息，若无此编译开头，则调试时将只能看到汇编代码，无法看到源代码。

编译成功后，输入以下调试命令：

```
WD /tr=rsi  h
```

常用调试按键：

F10 单步不进入函数(step over)， F8 单步进入函数(trace into)， F4 查看程序输出结果； 鼠标双击源

代码某行左侧的[]变成[!]就是在此处设一个断点；

Watch 窗口按 insert 键可以添加对某个变量的观察。

6. 双机调试

Watcom C 支持双机调试，比如 A 电脑打开调试器，B 电脑可以看到程序的输出结果。这种方式特别适合调试图形程序。

我们可以用两个 DosBox 模拟两台主机，从而在单机上实现双机调试。步骤如下：

① 计算机->右键->属性->设备管理器->网络适配器(鼠标选中)

点菜单"操作"->添加过时硬件->安装我手动从列表选择的硬件->网络适配器->Microsoft->Microsoft Loopback Adapter(Win8 中的名称为"Microsoft KM-TEST 环回适配器"或"Microsoft KM-TEST Loopback Adapter")

② 安装 WinPcap_4_1_3.exe，下载链接如下：

http://10.71.45.100/bhh/winpcap.rar

③ 打开 DosBoxWc\wc.conf 并搜索"realnic"，把它的值改成 list 并保存

④ 双击 wc.exe 观察其输出的网卡列表信息，找到 Microsoft Loopback Adapter 网卡的序号，并把 realnic 的值改成此序号(若找不到，则重启一次电脑)。

⑤ 关闭 DosBox，再双击 wc.exe 重新运行 DosBox

⑥ 打开 DosBoxWc\server\sv.conf，把其中的 realnic 改成④中相同的值。

⑦ 双击 DosBoxWc\server\sv.exe 运行第 2 个 DosBox

⑧ 在第 2 个 DosBox 中输入以下命令：

```
netserv /tr=rsi
```

⑨ 在第 1 个 DosBox 中输入以下命令：

```
wd /tr=net h
```

其中 h 是你想要调试的程序，且 h 必须事先用以下命令编译过：

```
wcl386 /d2 h.c
```

7. Watcom C 参考资料

①DosBoxWc\Watcom\DOCS\CGUIDE.PDF 用户手册

②DosBoxWc\Watcom\DOCS\CLIB.PDF 库函数手册

③DosBoxWc\Watcom\DOCS\PGUIDE.PDF 程序员手册

8. VESA 图形库函数列表

```
/* The following functions are compatible with TC */
void  geninterrupt(int intnum);
int   int86( int, union REGS *inregs, union REGS *outregs);
int   int86x(int intno, union REGS *inregs, union REGS *outregs,
/*==================*/ struct SREGS *segregs);
void  disable(void);
void  enable(void);
void  setvect( unsigned intnum, void (__interrupt __far *handler)() );
void  (__interrupt __far *getvect(unsigned intnum))();
byte  inportb(int port);
```

```
void  outportb(int port, byte value);
int   bioskey(int cmd);
void  randomize(void);
int   random(int n);

void  textmode(int newmode);
void  textbackground(int newcolor);
void  textcolor(int newcolor);
void  clrscr(void);
void  gotoxy(int x, int y);
int   wherex(void);
int   wherey(void);

int   putch(int x);
// This function is a substitute for putch() declared in conio.h,
// because the original putch() does not trace the coordinates of
// the cursor which results in _gettextposition()'s returning
// an incorrect value.

int   gettext(int left, int top, int right, int bottom, void *p);
int   puttext(int left, int top, int right, int bottom, void *p);
void  window(int left, int top, int right, int bottom);

void  initgraph(int *graphdriver, int *graphmode, char *pathtodriver);
void  closegraph(void);
void  cleardevice(void);
int   graphresult(void);
char  *grapherrormsg(int errorcode);

void  putpixel(int x, int y, dword pixelcolor);
dword getpixel(int x, int y);
dword imagesize(int left, int top, int right, int bottom);
void  getimage(int left, int top, int right, int bottom, void *bitmap);
void  putimage(int left, int top, void *bitmap, int op);
void  setactivepage(int page);
void  setvisualpage(int page);
void  setcolor(int color);
void  setbkcolor(int color);
void  setfillstyle(int pattern, int color);
void  settextjustify(int horiz, int vert); /* a dummy function */

void  settextstyle(int font, int direction, int charsize);
```

```
 /* a dummy function */

 void  setlinestyle(int linestyle, unsigned upattern, int thickness);
 /* a dummy func */

 void  setrgbcolor(int colornum, int red, int green, int blue);
 int   getcolor(void);
 int   getbkcolor(void);
 int   getx(void);
 int   gety(void);
 int   getmaxx(void);
 int   getmaxy(void);
 void  outtextxy(int x, int y, char s[]);
 void  getfillsettings(struct fillsettingstype *fillinfo);
 void  setfillpattern(char *upattern, int color);

 void  drawellipse(int x, int y, int xradius, int yradius);
 /*======================== 画椭圆不填充，边界线颜色由 setcolor()设定，TC 无此函数 */
 void  fillellipse(int x, int y, int xradius, int yradius);
 /*======================== 画椭圆并填充，填充物及其前景色由 setfillstyle()设定，
                              背景色由 setbkcolor()设定，无边界线 */
 void   ellipse(int  x,  int  y,  int  stangle,  int  endangle,  int  xradius,  int
yradius);
 /*=================== 画椭圆中的一段弧，圆心与起点及终点均无连线，不填充 */
 void  arc(int x, int y, int stangle, int endangle, int radius);
 /*=================== 画正圆中的一段弧，圆心与起点及终点均无连线，不填充 */
 void  line(int x1, int y1, int x2, int y2);
 void  lineto(int x, int y);
 void  linerel(int dx, int dy);
 void  moveto(int x, int y);
 void  moverel(int dx, int dy);
 void  rectangle(short x1, short y1, short x2, short y2);
 /*======================== 画矩形不填充，边界线颜色由 setcolor()设定 */
 void  bar(short x1, short y1, short x2, short y2);
 /*================== 画矩形并填充，填充物及其前景色由 setfillstyle()设定，
                        背景色由 setbkcolor()设定，无边界线 */
 void   bar3d(int  left,  int  top,  int  right,  int  bottom,  int  depth,  int
topflag);
 /*=================== 画三维矩形并填充，填充物及其前景色由 setfillstyle()设定，
                           背景色由 setbkcolor()设定，边界线颜色由 setcolor()设定
*/
 void  circle(int x, int y, int radius);
```

```
 /*================= 画正圆不填充，边界线颜色由 setcolor()设定；要画有填充的正圆
                       请调用函数 fillellipse() */
 void  sector(int x, int y, int start_angle, int end_angle, int xradius, int
yradius);
 /*================= 画椭圆中的一个扇区并填充，填充物及其前景色由 setfillstyle()设定，
                       背景色由 setbkcolor()设定，边界线颜色由 setcolor()设定；
                       start_angle 及 end_angle 表示起始角度及终止角度 */
 void  pieslice(int x, int y, int start_angle, int end_angle, int radius);
 /*=================== 画正圆中的一个扇区并填充，填充物及其前景色由 setfillstyle()设
                      定，背景色由 setbkcolor()设定，边界线颜色由 setcolor()设定；
                      start_angle 及 end_angle 表示起始角度及终止角度 */
 void  __arc(short fill, short x1, short y1, short x2, short y2,
 /*====================*/ short x3, short y3, short x4, short y4 );
 /*====================== 画椭圆中的一段弧，圆心与起点及终点均无连线，弧线颜色由
                       setcolor()设定。当 fill==_GFILLINTERIOR 时，此段弧对应的
                       扇区会被填充，填充物及其前景色由 setfillstyle()设定，背
                       景色由 setbkcolor()设定；当 fill==_GBORDER 时，此段弧对应
                       的扇区不会被填充。 (x1, y1)及(x2,y2)是指包围此椭圆的矩
                       形的左上角及右下角坐标；(x3, y3)是落在椭圆外的一个点，
                       该点与圆心构成一条直线，此直线与椭圆的交点就是弧的起点；
                       同理，(x4, y4)决定弧的终点。TC 无此函数。*/

 void  __pie(short fill, short x1, short y1, short x2, short y2,
 /*====================*/ short x3, short y3, short x4, short y4 );
 /*====================== 画椭圆中的一个扇区，比__arc()多了圆心与起始点的连线，
                       连线颜色与弧线颜色相同,由 setcolor()设定。TC 无此函数 */
 short floodfill( short x, short y, dword stop_color);


 /* The following functions are not compatible with TC. */
 /* They are included here to assist you in video programming. */
 void text_mode(void);
 int  set_bmp_palette(BMP_PALETTE *palette);
 int  load_8bit_bmp(int x, int y, char *filename);
 void set_timer_frequency(word divisor);
 int  enum_vesa_modes(void);
 void rep_stosd(void *pdest, dword data, dword count);
 /* The functions above are not compatible with TC. */
```