TURBOC 2.0

图形程序设计方法与实例

- 图形函数描述
- 圆、弧、和线等的画法
- 获取编入文本文档图形结果的方法
- 视口和存储映象联系的Turbo C函数、鼠标输入方法等。

中国科学院希望高级电脑技术公司

TURBO C 2.0
图形程序设计方法与实例

罗　辉
陈　刚
孟　晨

中国科学院希望高级电脑技术公司
一九九一年五月

序

在我以前的一本书——《IBM-PC计算机图形》中，其主要目标是图形例程集的开发，写该书的时候，C编译程序还没提供任何图形功能，所以当时开发并推出了一种图形工具箱，它是基于ROM－BIOS例程、基本输出指令和对图形适配器直接存取1，仅两年后，情况便大不一样，Borland式司的Turbo C（2.0版）提供了大量的图形例程集，而且这种编译程序非常流行，现在人们注意力已从开发基本图形工具转到如何使用Turbo C所提供的这种工具上。这就是本书所要讨论的问题。读了本书后，你也许首先得到这么一个印象：总是在用我自己的图形C函数代替BorLand的图形C函数，然而如果进一步仔细阅读图形函数，就会发现它们包含了对Turbo C图形函数的调用，而且完全依赖于这些调用。我对这些BoLand图形库既不替换，也不补充，而仅仅是使用这些库。所以，Turbo C支持的图形适配器的范围也适用于本书中开发的图形例程。

每个用Turbo C编制图形应用程序的人都会觉得应该有比Turbo C参考手册更多的例子，因而我用到许多的（图解说明的）例子，范围从一个简单三角形到一个完全交互式绘图程序不等。第一章描述一些基本Turbo C图形函数，一个用户坐标系统和转换Turbo C图形驱动程序为目标文件及连接它的方法；第二章讲到特征比例以及圆、弧和任意宽度的线的画法。第三章讲的是如何获得被编入文本文档的图形结果，这跟我写入文本书的一样，演示了Howlett-Packlard图形语言（HP－HL），尤其是同Wordperfect 5.0这样的台式印刷系统相连的用法，该章还给出一个较完善的图形工具箱，即GRASPTC 第四章，要想编写娱乐或艺术方面的图形程序的用户可能对该章感兴趣。最后，第五章讨论了与视口和存储映象联系的Turbo C函数，该章也提供了鼠标输入方法，特别是对要开发用鼠标控制的绘图程序，书中给出了一个SDRAW原代码文本。

目　录

第一章　象素，行和文本

1.1　引言

象任何C程序一样，一个Turbo C图形程序主要由函数组成，因为编写C函数的最好方法最近已经改变，在这一点上我们将避免一切混乱，并且对使用现代(modern) C编译程序的C程序设计者从有趣的题目开始。

本书中我们将不时地用编写函数定义和说明的ANSI风格，例如，函数

```
square (x, y, h) float x, y, h;
 {move(x-h, y-h) ; draw(x+h, y-h) ; qraw (x+h, y+h) ;
  draw (x-h, y+h) ; draw (x-h, h-h) ;
}
```

这是用kernighan和Ritchit原来的风格编写的，现在我们写成

```
void square (float x, float y, float h)
 {move(x-h, y-h) ; draw(x+h, y-h) ; draw (x+h, y+h) ;
  draw (x-h, y+h) ; draw (x-h, h-h) ;
}
```

这是按照ANSI标准编写的。在《Turbo C用户指南》术语里，将不再使用类程风格，而用现代风格所代替，关键字void能使我们自然地编写一个不返回值的函数的说明，例如，

```
void square (float x, float y, float h) ;
```

相反，用类程风格则必须用关键字int代替void，比如，一个说明也叫做一个函数原则。(假设你熟悉函数定义之间的差别，指的是函数本身和一个函数说明仅给出有返回值的信息（用现代风格）和有关任何参数。如果一个函数没有参数，用关键字Void做别的用途，换句话说就是做为一个空参数表，例如，在

```
int readadigit (void)
 { char ch;
  ch=getchar ( ) ;
  return (isdigit (ch) ? ch - '0' : -1) ;
}
```

中如果把这个函数第一行中的关键字删去，Turbo C将把这个函数看作是用类程风格编写的，参数的个数是未定义的，包括这个关键字的话，则相反，我们将告诉编译程序可读数没有参数，以便在对这个函数调用时出现的写出一个或多个自变量的错误时，将得到一个错误信息。

一定不要使用那些未经说明的函数，（这里函数定义即说明，那么，如果一个函数仅在被定义之后使用，则不要把函数的说明分开。）只准使用预先定义好的函数的这个规则也适

用于“标准”函数，例如，Printf、幸运的是，对于每一个Turbo C标准函数都有一包含函数说明的头文件（以.H结尾的一个文件名）。由于象这种用现代风格编写的说明我们也可以调用它的函数原型，考虑如下程序：

```
#include <stdio.h>
main ( )
{ puts ("Hello") ;
}
```

尽管极其简单，但也很能说明问题，用类程风格，第一行可以省略，因为编译程序将假定puts返回一int一值（确定是这种情况）。在现代风格中，我们最好包括这行，因为文件STDIO.H包括函数原型。

```
int puts (char *string) ;
```

它使编译程序检测在Puts调用中是否有正确类型的自变量。例如，由于以上程序的第一行，下列调用之一

```
puts ("Hello", "Good Morning") ;
puts (123) ;
```

如果代替Puts（“Hello”），则将引起编译程序显示一错误信息。如果没有说明函数Puts（或者直接地或者包括文件STDIO.H）就已经使用这两行，那么，得到的结果将不是编译时间错误信息，而是难以预料的。

1.2 图形程序

现在将讨论一个简单的程序，该程序用到在Turbo C（1.5版或更高级的版本）上可用的一些重要的图形函数，我们即将使用一矩形坐标系统，原点位于屏幕左上角，坐标是整数，范围从0到最大值，该最大值取决于所用的图形适配器，让我们把变量x__max和Y__max分别作为x_和y_的最大值（这些变量名下的两道线可以避免同x_max和y_max。发生混淆，因为本书中x_max和y_max常用作别的用途。）如图1.1所示。

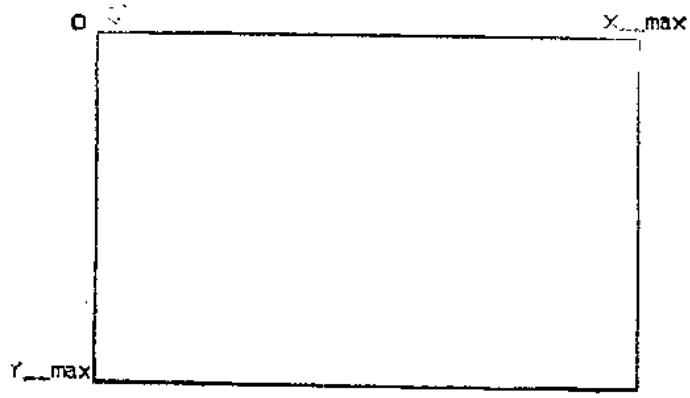

Fig. 1.1. Screen coordinates

程序TRIA绘出屏幕上最大的直角三角形，它的直角位于屏幕的左下角：

```
/* TRIA: A large right-angled triangle.
*/
#include <graphics.h>
#include <conio.h>

main()
{ int gdriver=DETECT, gmode, X_max, Y_max;
  initgraph(&gdriver, &gmode, "\\tc");

  X_max = getmaxx(); Y_max = getmaxy();
  moveto(0, Y_max);          /* Bottom left  */
  lineto(X_max, Y_max);      /* Bottom right */
  lineto(0, 0);              /* Top left     */
  lineto(0, Y_max);          /* Bottom left  */
  getch();
  closegraph();
}
```

假如你希望编译，连接和成功地运行本程序，有两点要特别注意：

1. 一定告诉连接程序检索图形函数的图形库，比如，本程序中initgraph，在TurboC（2.0版）中，做此事的最简单的方法是选择可选项，然后连接程序把可选项图形库转换成on状态（取代缺省的off状态），也不要忘记选择save可选项，以便编译程序下一次能记住你所做的选择。

另外，还可以用投影文件，在这样简单的情况下这种方法没什么吸引力，但如果是较大的程序，或者是分成几个模块的程序，都可以用投影文件，而且也可以在这些文件中包含有文件名GRAPHICS.LIB。在这里，此方法按如下操作。如果本程序的文件名是TREA.C，那么必有一投影文件，即具有如下内容的TRIA.PRJ文件。

```
tria
graphics.lib
```

选择投影文件将按ALT-P键，Enter键和名字TRIA（或者TRIA.PRJ）。

2. 图形适配器的驱动程序必须是在当前驱动的目录\TC或当前目录下，在一个硬盘上按一般方法安装Turbo C（2.0版）之后，前者将是这种情况，但可以在目录/TC下通过查询以.BGI结尾的文件名来检查这一点。记住，以上程序将在运行时装入它需要的图形驱动程序。在1.7节中我们将讨论在程序中包括图形驱动程序的一种更高级的方法。

程序TRIA包含调用一些Turbo C图形函数，头文件GRAPHICS.H包含许多这种函数的说明，这一阶段，考虑如下程序：

```
void far initgraph(int far *graphdriver,
                   int far *graphmode,
                   char far *pathtodriver);
int far getmaxx(void);
int far getmaxy(void);
void far moveto(int x, int y);
void far lineto(int x, int y);
void far line(int x1, int y1, int x2, int y2);
void far closegraph(void);
```

上面关键字far的出现引起编译程序用到长指针格式；如果省略掉far，伴随“小”存储器模型将与指针格式发生冲突。因为你也许知道，可以通过选择可选项、编译程序、最后是存储器模型这几步选择存储器模型。由于常涉及到具有大量数据区的大程序，我总是使用巨型存储器模型，因而我的程序指针格式总是较长，若每人都这么做的话，关键字far就是多余的了。

上面第一个函数initgraph，把计算机从文本方式转换成图形方式，不论何时，要想在屏幕上得到图形，必须得到有这个函数。这个重要的函数有三个参量，三个参量都与程序TREA中的自变量相对应，如下所列：

parameter	Argumnent
graphdriver	&gdriver
graphmode	&gmode
pathtodriver	"\\tc"

若我们用到graphdriver的值DETECT（GRAPHICS.H中定义为0），initgraph将发现图形适配器正在被使用，并且分配一个整数代码，“图形驱动程序常数”与适配器、变量gdriver相对应，适配器的最高分辨率代码由变量gnode指定，由第二个自变量给出。最后，可以发现，图形驱动程序所在的目录，如HERC.BGI由以字符串的形式给出（或者，对一个字符意味着一个指针）。我们的例子中，目录为\tc，注意，如果确想仅得到一个斜杠（\），则在串中必须写出两个紧挨的斜杠（\\），因为单个斜杠将作为一个“扩展符”，例如，\t是标号符的C表示法，若initgraph不能找到由Pathtodriver指定的目录上所要求的图形驱动程序，那将在当前目录上进行检索。假使相应地，如果使initgraph仅在当前目录下检索，可写出空串“”或空指针NULL。

函数Closegraph恰与initgraph相反，它是在程序TRIA末尾被调用，它的作用是又转换为文本方式。

函数getmaxx和getmaxy返回当前图形方式下X和Y的最大值，这些值由我们的图形适配器而定，把这些值赋给变量x__max和y__max主要是为了能多次使用而又不每次调用getmaxx和getmaxy，这两个函数的调用可在调用initgraph后进行。

要绘制三角形需调用moveto和lineto。坐标是用整数表示象素数，是从屏幕的左上角开始算起，左上角第一个为0，调用moveto（X,Y）和Lineto（x，y），就好象从当前位置到点（X，Y）移动一根笔一样，与moveto不同，lineto在当前位置和点（X，Y）之间绘出一条粗直线。

取代

```
moveto (x1, y1) ; lineto (x2, y2) ; lineto (x3, y3) ;
```

可写做

```
line (x1, y1, x2, y2) ;
line (x2, y2, x3, y3) ;
```

可以看出，用Line绘出连接的线段并不好，因为每个连接点坐标不得不指定两次，（如x2和y2）。

1.3 象素和颜色

这节涉及到单个象素可能的情况，对于单色图形适配器，如HGA，屏幕的每个象素要么暗的，要么亮的；对于其它适配器如EGA，有一个可用颜色范围。要区别前景和背景，因为每种选择的颜色都有前台和后台之分，可用color项，也可用象素的暗和亮的区别。

我们将使用GRAPHICS.H中的一些Turbo C函数，说明如下：

```
int far getmaxcolor(void);
void far setcolor(int color);
void far setbkcolor(int color);
int far getcolor(void);
int far getbkcolor(void);
void far putpixel(int x, int y, int pixelcolor);
int far getpixel(int x, int y);
```

（1.2节中已经简要说明了关键字far，这里不再重述。）

可用的颜色编号为

```
0, 1, 2, …,   getmaxcolor ( )
```

所以用函数getmaxcolor可查询最大颜色数，对于HGA，getmaxcolor返回值为1，则表示“亮”，前景，0表示“暗”，背景。彩色屏幕如EGA，通常只用两色，没必要为颜色数码伤脑筋，因为可以写出来，如：

```
int foregrcolor, backgrcolor, colorsum;
...
foregrcolor = getcolor();   /* Foreground color */
backgrcolor = getbkcolor(); /* Background color */
colorsum = foregrcolor + backgrcolor;
```

注意getcolor和setcolor与前景有关而getbkcolor和Setbkcolor则与背景有关，附加两个颜色代码似乎很奇怪，但可用colorsum的值改变象素状态，如下函数所示：

```
void invertpixel (int x, int y)
 ( putpixel (x, y, colorsum getpixel (x, y) ) ;
}
```

如果getpixel的调用返回值为backgrcolor（后台色），就很容易验证putpixel的第三个自变量值为foregrcolor（前台色），反之亦然。

如果要使用除缺省的前景和背景的任何颜色（显然不是用单色图形这种情况），那么应该有一些可用颜色代码的概念，getmaxcolor ( ) 的值允许我们尽可能这样做，我们可以使用的颜色请见下表。

Numerical value	*Symbolic constant*	*Foreground or background?*
0	*BLACK*	Both
1	*BLUE*	Both
2	*GREEN*	Both
3	*CYAN*	Both
4	*RED*	Both
5	*MAGENTA*	Both
6	*BROWN*	Both
7	*LIGHTGRAY*	Both
8	*DARKGRAY*	Foreground
9	*LIGHTBLUE*	Foreground
10	*LIGHTGREEN*	Foreground
11	*LIGHTCYAN*	Foreground
12	*LIGHTRED*	Foreground
13	*LIGHTMAGENTA*	Foreground
14	*YELLOW*	Foreground
15	*WHITE*	Foreground

上面的符号常数是在头文件GRAPFICS.H中定义的，比如，如果想使后台颜色是绿的，可写成：

```
Setbkcolor (GREEN) ;
```

代替

```
Setbkcolor (2) ;
```

1.4 异或（XOR）写方式

象在讨论getpixel和putpixel时所做的一样，反转象素通常被当作是“在异或方式下编写”，缩写词XOR可当做‘exclusive or’（异或）操作，C中写作^。对于变量x和m，以下的操作

```
x^=m
```

能够使x的一些位变反，条件是与x对应的m的相应位为1。

交互式图形程序中，常常想用到一些由定位输入器或光标指示的可变点，用一图形输入设备如鼠标，我们便可以把定位输入器移到因某些这样那样的原因而要指示的位置。移动输入定位器时，不断擦除一些图形，但不是永久性消除，可由变反象素完成这个工作，在屏幕上移动定位输入器时，使形成图案和输入定位器产生点的象素变反，这样的变反操作进行两次；第一次是在输入定位器移到一点；第二次是在输入定位器移开时。显然，对一个象素变反两次就又恢复到它原来的状态，因此，我们将实际应用一下前一节讨论的invertpixel函数。

既然知道如何使单个象素反转，理论上讲，要使一行中所有象素变反只要对每个象素调用函数invertpixel就行了，在TurboC1.5中必须得这样做，但TurboC2.0提供了一个新函数，说明如下：

```
void far setwritemode (int mode) ;
```

对方式，可以用以下常量：

```
COPY_PUT   (=0)
XOR_PUT    (=1)
```

缺省的写方式是COPY_PUT；该方式下一般绘出行，而不考虑屏幕上其它情况。在XOR

_PUT写方式下则相反，要绘制的行的象素被变反，新函数Setwritemode可以做到这一点，我们也可以用标准Turbo C函数moveto、lineto、和line来做，可是在2.0版本以前，在函数invertpixel基础上，不得不用我们自己的三个函数。因为Turbor C函数画线是非常快的，我们也没必要自己编写。

程序GRHAIRS示范一下函数Setwritemode的用法。如图1.2所示，程序绘出一个矩形并在矩形内绘出许多条线之后，开始调用该函数使之转为XOR_PUT方式，并且绘出一条水平线和一条垂直线，这两条线叫做十字准线（crosshairs）；能指示出交叉点，而且可使我们在同样的x或y坐标上对点做比较。程序CRHAIRS有趣的是通过按四个箭头键就可以使十字准线上下左右移动，同时又不擦除屏幕上任何其它内容。

the four arrow keys we can move either the vertical or the horizontal crosshair without destroying anything else on the screen.

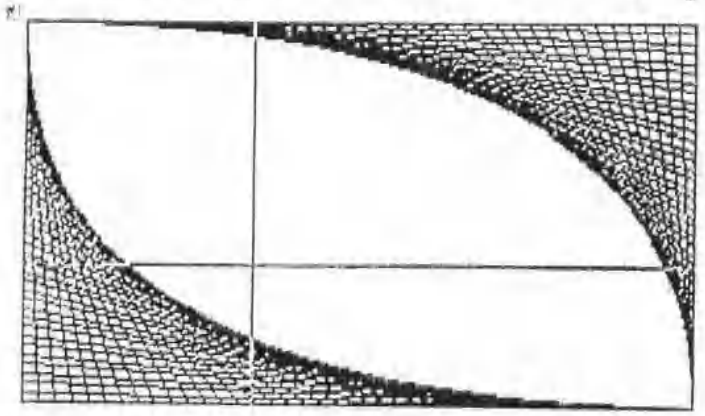

Fig. 1.2. Drawing with crosshairs

```
/* CRHAIRS: Crosshairs.
*/
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#define H 40
#define DX 15
#define DY 6

int Xmax, Ymax;

int round(float x)
 /*  The positive value x is rounded to the nearest integer */
{ return ((int)(x + 0.5));
}

void cross(int X, int Y)
{ static int Xcur=-1, Ycur=-1;
  if (X < 0) X = 0; if (X > Xmax) X = Xmax;
  if (Y < 0) Y = 0; if (Y > Ymax) Y = Ymax;
  if (X != Xcur)
  { if (Xcur >= 0) line(Xcur, 0, Xcur, Ymax);
                          /* Remove old line */
    line(X, 0, X, Ymax);    /* Draw new line   */
```

```
  }
  if (Y != Ycur)
  { if (Ycur >= 0) line(0, Ycur, Xmax, Ycur);
                           /* Remove old line  */
    line(0, Y, Xmax, Y);    /* Draw new line    */
  }
  Xcur = X; Ycur = Y;
}

main()
{ int gdriver=DETECT, gmode, i, OK, X, Y;
  char ch;
  float dX, dY;
  float redfact;
  /* We first draw some picture, in which later
     crosshairs will move without destroying it.
  */
  printf("Reduction factor (not greater than 1): ");
  scanf("%f", &redfact);
  printf("When in graphics mode, you can move the crosshairs "
         "by pressing any\nof the four arrow keys.\n");
  printf("\nPressing any other key will "
         "then cause the program to terminate.\n\n");
  printf("First, press any key to switch to the graphics mode ...");
  getch();
  initgraph(&gdriver, &gmode, "\\tc");
  Xmax = round(getmaxx()*redfact);
  Ymax = round(getmaxy()*redfact);
  dX = (float)Xmax / N; dY = (float)Ymax / N;
  for (i=0; i<=N; i++)
  { line(0, round(i * dY), round(i * dX), Ymax);
    line(Xmax, Ymax - round(i * dY), Xmax - round(i * dX), 0);
  }
  /* Crosshairs through the center of the screen:
  */
  setwritemode(XOR_PUT);
  X = Xmax/2; Y = Ymax/2;
  cross(X, Y);
  /* The crosshairs can now be moved:
  */
  OK = 1;
  while (OK && getch() == 0)
  { ch = getch();
    switch (ch)
    { case 72: cross(X, Y -= DY); break; /* Up    */
      case 75: cross(X -= DX, Y); break; /* Left  */
      case 77: cross(X += DX, Y); break; /* Right */
      case 80: cross(X, Y += DY); break; /* Down  */

        default: OK = 0;  /* Wrong key */
      }
    }
    closegraph();
  }
```

该程序包含一些有趣的特征，值得解释一下，象在大多数交互程序中一样，通过调用getch我们发现一个从由键盘读入一个字符的循环，如果该字符是空字符（可用0或"\0"表示，但不能以'0'表示）那么一定按了一特殊键，并且下一个字符立即可以使用。在那些特殊键中，我们最感兴趣的是用以下代码的箭头键：

↑	0, 72
←	0, 75
→	0, 77
↓	0, 80

这四种情况之一可使x或y改变且函数cross被调用，因为一次只能按一个箭头键，所以不得不移动水平或垂直准线，用x和y的新值同当前值Xcur和Ycur做比较，检查一下比较的情况。这些变量是"静态的"；所以这些值保存在两个cross函数的调用之间，因为它们的初值为－1，这与第一次调用cross后的x和y都不同，而且，两次的准线都被画出，第一次调用的另一个特征是：这种情况下旧十字准线没有被擦除，主要是因为函数中的两个内部if语句。

1.5 用户坐标

图形程序中我们常常不得不多少解决一些困难的问题，这些问题都是应用程序本身所固有的，而且如果在同一时刻，又不得不处理一些产生于我们已使用的硬盘或软盘的问题，那将很不幸，象这样的一个问题是由正在使用的坐标系统引起的。在许多情况下，特别是涉及到数学计算，我们便可用实数而不用整数，除此之外，y轴朝上将更接近人们习惯而且也很方便，因此我们要建立这样一个较方便的坐标系统，这种坐标系统原点位于屏幕左下角，并且具有实数适配器独立坐标。

我们把屏幕上x轴分为十份，X_max是x最大值，同样，y的最大值为y_max，则有：

```
0≤x≤x_max
0≤y≤y_max
 (x=0, y=0 in the lower-left corner)
```

有时把这种新单位叫'英寸'（inches'），尽管对大多数显视器新单位比一英寸要小。选定x_max＝10之后，还不能完全自由选择，因为计算机屏幕的高比宽要小，y_max必须小于10，在我以前的图形书籍中，通常用的是7.0，即y_max＝7.0。但既然现在用的是Turbo C图形因而就得十分仔细，而且有一套考虑到水平和垂直准数处理问题的方法，我们将在2.1节中讨论这个问题，现在暂时令y_max＝7.0，在第二章时将对它做一些修正。然后我们也可用下面的模块，该模块内也包含有函数invertpixel（在1.3节中讨论）。

```
/* GRASPIC0: Graphics System for Programming in Turbo C
             (Preliminary version)
*/
#include <graphics.h>
#include <conio.h>
int X__max, Y__max, foregrcolor, backgrcolor, colorsum;
float x_max=10.0, y_max=7.0, horfact, vertfact;

void initgr(void)
{ int gdriver=DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc");
  foregrcolor = getcolor(); backgrcolor = getbkcolor();
  colorsum = foregrcolor + backgrcolor;
  X__max = getmaxx(); Y__max = getmaxy();
  horfact = X__max/x_max; vertfact = Y__max/y_max;
}

int IX(float x)
{ return (int) (x * horfact + 0.5);
}

int IY(float y)
{ return (int) Y__max - (int)(y * vertfact + 0.5);
}

void move(float x, float y)
/* User-coordinate version of 'moveto' */


{ moveto(IX(x), IY(y));
}

void draw(float x, float y)
/* User-coordinate version of 'lineto' */
{ lineto(IX(x), IY(y));
}

void line_uc(float x1, float y1, float x2, float y2)
/* User-coordinate version of 'line' */
{ line(IX(x1), IY(y1), IX(x2), IY(y2));
}

void endgr(void)
{ getch(); closegraph();
}

void invertpixel(int X, int Y)
{ putpixel(X, Y, colorsum - getpixel(X, Y));
}
```

模块中心图形程序设计要比上节所描述的容易很多，我们可以不用图形函数调用而设计出许多有趣的图形程序，这一点与下面的不同：

initgr ( ) ;　　初始化，转化为图形方式。

move (x, y) ;　移动一个虚设的笔到点 (x, y) ，该点作为'当前位置' ('Current Position') ，原点0位于左下角，坐标范围$0 \leq x \leq 10$，$0 \leq y \leq 7$。

draw (x, y)：从当前位置到点 (x, y) 画一直线，点 (x, y) 变为新的当前位置。

endgr ( )；　　一直等到按一键，然后转回文本方式。

如我们在1.1节中所见，应该在使用函数之前给它一个说明，因此我们将用如下我们自己的头文件：

```
/* GRASPTCO.H: Header file, to be used in any module that uses the
               functions defined in GRASPTCO.
*/
#include <graphics.h>
extern int X_max, Y_max, foregrcolor, backgrcolor, colorsum;
extern float x_max, y_max, horfact, vertfact;

void initgr(void);
int IX(float x);
int IY(float y);


        void move(float x, float y);
        void draw(float x, float y);
        void line_uc(float x, float y);
        void endgr(void);
        void invertpixel(int X, int Y);
```

在所有使用新函数的程序中，应该使用下面函数

#include　　"grasptco.h"

我们程序中常常不用常量10和7，而用x_max和y_max，因为这两个变量已经在上面的标题文件中加以说明，没必要再说明。

下面的程序能代替1.2节中所列的程序TRIA：

```
/* TRIA1: This program draws a large triangle.
          (After compilation it is to be linked together
           with GRASPTC0.)
*/
#include "grasptc0.h"

main()
{ initgr();
  move(0.0, 0.0);     /* Lower left  */
  draw(x_max, 0.0);   /* Lower right */
  draw(0.0, y_max);   /* Upper left  */
  draw(0.0, 0.0);     /* Lower left  */
  endgr();
```

现在一定不要忘记把模块GRASPTCO同主程序连接起来，所以用一投影文件，其目录为：

```
trial
grasptc0
graphics.lib
```

（如果已经命令你的编译程序经常查询图形库，可以省略最后一行，尽可能用TurboC

2.0版。）

四个函数initgr、move、draw和endgr用起来比'标准函数initgraph、moveto、lineto和closegraph容易很多，如上面的文件GRASPTCO.C和GRASPTCO.H所示，函数IX、IY、line_uc和invertpixel也可用在应用程序中，全局变量x_max、y_max、x_max、y_max、horfact、Vertfact、foregrcolor、backgrcolor、colorsun，也同样如此。以后会知道这些都是有用的。注意最大象素坐标X_max和Y_max取决于我们的图形适配器，且有助于在调用initgr（）后，它们的值立即可以使用。谈到最大屏幕座标x_max和y_max可能得给它们一个具体值，而不是缺省值10.0和7.0。所以如果你愿意的话可以用不同的长度单位，如毫米。可是我劝你在阅完2.1节之前不要这么做，切记，GRASPTCD只是GRASPTC的初级版本，有关GRASPTC列于第三章末尾。

1.6 图形方式的文本和字体

要求图形输出包含文字这很正常，Turbo C提供了两个函数来完成这种功能，并且在GRAPHICS.H中作如下说明

```
void far outtext (char far *textstring) ;
void far outtextxy (int x, int y, char far *textstring) ;
```

Outtexty被明确给一起始点x，y（用象素坐标），相反outtext用当前位置作为起始点，通常（不用settextjustify）'起始点'是在第一个字符的左上角。

文本方式下，屏幕上字符形状由硬件字符生成器决定，所以我们不能修改这些形状。附带一点，字符形状的专门项是字体。另一方面，图形方式下，字符仅由点集组成：这里我们想要的每一字体，理论上讲是由软件实现的，不久我们就会看到，几种字体和许多尺寸都可在Turbo C上使用，但进一步又允许我们用"缺省"字体和尺寸；这意味着每个字符都是以8×8见方的象素来显示的。图形方式下显示文本时没有这样的功能：在一行填满情况下继续下一行，所以我们不得不检查要显示的文本是否在屏幕边界认内，这时一定不要把字符数同象素数搞混淆，幸运的是，Turbo C提供了两种函数，在GRAPHICS.H中说明如下：

```
int far textwidth (char far *tetstring) ;
int far textheight (char far *textstring) ;
```

这些函数返回由textstring给定的文本分别在水平和垂直方向上的象素数。（这不反对缺省字符形状操作，而且也对其它字体和尺寸操作）如，X_max和y_max作为x和y的最大值，可以在屏幕右下角显示文本"ABC"如下：

```
outtextxy (X_max+1-textwidth ("ABC") ,
           Y_max+1-textheight ("ABC") , "ABC") ;
```

如果你正在用Hercules图形适配器，该调用的第一和第二个自变量分别是：

719+1-3×8=696和
347+1-8=340，

如果对于我们的应用程序，仅一种字体和尺寸不够，可用如下Turbo C函数：

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