HOPE

TurboC++

图形设计技术与实例

李 伟
熊文杰 编译
申俊杰

北京希望电脑公司

TurboC++ 图形设计技术与实例
北京希望电脑公司

Turbo C++图形设计技术与实例

李 伟
熊文杰 编译
巫俊杰

北京希望公司

目录

前言

如果用户对使用 Turbo C++进行图形设计感兴趣，那么本书是再恰当不过的了．过去，图形编程一直是个困难、费时的工作．幸运的是，由于 Turbo C++ 1.0 版本的公布，用户可以使用面向对象的编程技术(OOP)，并且很容易地使用图形工具 Borland Graphics Interface(BGI)来简化绘制二维和三维个人计算机图形的工作量．

近年来出版了许多关于图形编程方面的书，但是，这些书中的大部分都着重于低级实现方面的描述．即，它们介绍开发低级图形工具的方法，如简单的绘制线和弧的例程，剪裁和填充算法，以及图形硬件驱动程序．但是本书采取了高级的方法，描述如何使用 Turbo C++提供的图形功能来建立现实世界的图形应用．如果用户已经熟悉了 C++编程，并正准备学习图形编程技巧，那么就需要一本书来告诉用户如何简便地开发图形工具和应用程序．为了满足这个要求，《使用 Turbo C++进行图形设计》可以向用户介绍 Turbo C++和功能强大的 BGI．在本书中将讲述如何使用最新的 OOP 技术编写灵活，用途广泛的图形程序．

本书的重点是:

.BGI 函数的深入讨论

.与图形有关的一些特殊问题的讨论，包括 BGI 程序设计技巧

.设计面向对象图形工具的简易方法

.设计基于图形的用户界面的技术

.二维和三维图形编程的基础

.实时动画

.交互式应用程序，包括一个画图和一个绘图程序

为什么要读这本书

如果想知道如何掌握 BGI，开发自己实用的面向对象图形工具库，或编写大型图形应用程序，如计算机辅助设计(CAD)程序，就要用到本书．不论用户是否熟悉图形编程的基础，本书都将逐步讲述如何设计工具和应用程序，包括显示图形，图标编辑，基于图形的用户界面，绘图和 CAD 程序，以及三维图形演示程序．

如果用户没有用 Turbo C++写过图形程序，前三章将帮助熟悉 C++和 BGI．虽然我们会讨论 C++和 OOP 的有关问题，但用户仍需要一本其它的书，如 Bryan Flamig 的《Turbo C++自学指南》(Wiley)或其它有关 C++的书，如《用 Turbo C++进行面向对象程序设计》(Wiley)，来帮助你熟悉 C++和 OOP 技术．

需要些什么

配合本书需要 Turbo C++ 1.0 以及 IBM PC XT, AT, PS / 2, 或能够显示图形的其它兼容计算机系统．开发的所有程序都将工作于虚拟图形陈列，VGA，增强型图形适配器 EGA，彩色图形适配器，CGA 或 Hercules(大力神)图形适配器．由于 BGI 和所给程序的灵活性，它们可以容易地适应用户的任何系统．

我们给出的交互式图形工具，如图标编辑器，弹出式窗口，以及绘图例程都可以工作于 Microsoft 兼容的鼠标上。鼠标是交互图形系统的重要组成部分；实际上，我们用了整整一章来讲述如何支持鼠标。为了运行交互式图形程序(第 7 章至第 13 章)，必须保证计算机上安装了鼠标。

详细说明

本书是从第一个版本《使用 Turbo C 进行图形设计》发展而来，它是为 Turbo C 2.0 编写的。自第一个版本问世后，我们收集了许多建议并把它们加入到本书。我们还重新设计了所有的工具和应用程序来更好地发挥 C++和 OOP 技术的优势和灵活性。根据多年来 C++和图形编程的经验，本书包括了 PC 机程序员最感兴趣的课题。

《使用 Turbo C++进行图形编程》将从图形编程的基础开始，直到一些高级的课题，如建立交互绘图工具和三维图形设计。只要可能我们都运用 C++的特性来写程序，以便可以容易地修改和扩展它们。由于篇幅有限程序的功能也受到限制，但它们仍然是完整、清楚、很有用的--这不同于多数图形编程书中的"玩具"程序。

这里是各章的介绍

第一章：BGI 入门　该章介绍如何使用 BGI 工具和 C++访问图形硬件，并书写完整、独立的图形程序。同时，将讨论 Turbo C++的一些 OOP 特性。在建立完基础之后，将介绍如何开发一个绘图程序。

第二章：BGI 绘图函数　讲解如何使用主要的 BGI 绘图命令。还将介绍使用预置和用户定义的填充方式调整颜色和调色板，以及填充区域。

第三章：BGI 字形和文本　介绍在图形方式下显示正文的工具。用户将学习如何使用位图和字形(stroke fonts)，以及如何放大和剪裁字符。我们还要建立一些支持图形方式下的文本输入例程。

第四章：显示图形　介绍如何开发显示高质量图形的程序。在本章中，将学习把 BGI 字形和绘图例结合在一起显示饼图，直方图，图标，和动画图形。

第五章：二维图形技术　讨论控制二维图形变形的多种技术。

第六章：动画　介绍如何在二维图形中加入动画效果。将学习使用 getimage()和 putimage()函数来移动图形，并使用调色板来模拟动画。

第七章：建立鼠标工具　全面讨论如何建立一个工具库来控制 Microsoft 兼容鼠标。在本章中，将使用鼠标工具支持绘图程序。

第八章：图标　介绍如何建立一个有用的图标编辑器。该图标编辑器将被用于建立图标，我们在第十一章和十二章将用它来支持交互式绘图程序。

第九章：弹出式图形窗口　介绍在图形方式下支持弹出式窗口的工具库。

第十章：交互式绘图工具　介绍如何建立一套有用的交互式绘图例程。

第十一章：一个绘图程序　提供一个强大的绘图程序，包括鼠标支持的用户界面。

第十二章：一个 CAD 程序　给出一个二维 CAD 程序。

第十三章：三维图形　介绍三维图形编程的基础。在本章中将学到剪裁，投影，三维表示，以及更多的东西。

以上就是本书的主要内容，下面我们将从第一章开始。

第一章　BGI 入门

无论任何时候，图形在 PC 机世界起着越来越重要的作用．实际上，在本书完成的同一星期，Microsoft 就推出了最新的 DOS 下的图形接口环境，Windows 3.0，给计算机界带来了很大的冲击．基于文本的程序已经日薄夕山，因为 PC 机用户要求更加动感、更加容易使用的程序．建立这些程序的工具，如 Turbo C++和 Borland 图形接口(BGI)已经可以得到了．

为了开始编写基于图形的程序，我们将看一下 BGI 提供的基本工具．但是记住，这里只是讨论 BGI 的一些主要特性．我们还将介绍 C++的一些特性，如类(classes)和对象(objects)．在以后的章节中，将详细讨论 BGI 工具和面向对象的程序设计(OOP)特性，以及如何使用它们开发应用程序，如图标编辑器，弹出式菜单，CAD 和绘图程序．其中我们还将介绍二维和三维图形编程的重要概念．

我们将首先讨论 Turbo C++图形程序的基本结构以及如何使用几个 BGI 图形函数．这些例程有象素绘制函数，画线命令，画多边形的例程，和图形方式下的正文控制例程．

在本章的末尾我们将编写一个有趣的产生分割的程序．通过这个程序把已经学到的东西结合起来，看一看计算机是如何产生美丽的图形的．

1.1　起步

书写每个 C++图形程序时都要遵循一定的步骤．例如，在每个图形程序开始时都要把屏幕显示硬件设置为图形方式．另外，在每个程序结束时都要把监视器恢复成原来的显示模式．在本节中，将讲解一个典型的 Turbo C++绘图程序的各组成部分．

1.1.1　初始化 BGI

绘图程序的第一步包括初始化图形硬件．这里的硬件是指为了显示图形所必备的显示器适配器．现在有多种不同的 PC 机图形适配卡(每个都有自己不同的图形方式)．幸运的是，BGI 支持这些图形标准中的大多数．(第二章给出了它支持的图形适配卡和模式的列表．)这里先不讨论所有的图形适配卡和模式，只使用 BGI 初始化过程中的缺省模式，它允许我们把现在安装的图形适配卡设置成最高分辨率模式．

用于设置计算机图形方式的函数叫作 initgraph()，它的原型如下：

```
void far initgraph(int far *gdriver,int far *gmode,char far *driver_path);
```

initgraph()的函数原型在 graphics.h 文件中，它必须包含 [illegible] 图形程序的开始处．该文件包括了所有 BGI 组成成分的数据结构，常量，和函数原型定义．

注意在 initgraph()的函数原型中头两个参数是整型指针．它们分别存有示频适配器和模式的逻辑值．第三个参数指出了 Turbo C++的图形驱动程序所存贮的路径名．驱动程序存在于 Turbo C++源盘中，并有扩展名.bgi．指向驱动程序的路径名可以是全程路径名，如：

```
initgraph(&driver,&mode,"\\compilers\\tcpp\\graphics");
```

或者是部分路径名，如：

```
initgraph(&driver,&mode,"graphics\\drivers");
```

在第二个例子中，initgraph()寻找目录 graphics 的子目录 drivers。如果 BGI 驱动程序就存贮在工作目录中，也可以这样调用 initgraph():

```
initgraph(&driver,&mode,"");
```

这里，路径名被定义为空串。注意，当指定路径名时必须使用两条斜线来表示一条斜线，因为 Turbo C++把其中的一条斜线解释成一个特殊字符。现在让我们来看一看如何用 initgraph()来选择图形方式。用户可以通过把前两个参数设置成特定的值，从而告诉 initgraph()使用某个图形驱动程序和模式。另外，也可以让 initgraph()自动配置视频适配器。我们将会用到这个选项。如果把第一个指针参数设置成宏常量 DETECT,initgraph()将把图形系统设置成最高分辨率模式。DETECT 宏是在 graphics.h 头文件中定义的，它告诉 BGI 选择自动配置特性。initgraph()的第二个参数不必设置成任何值，但是必须包括一个指向整数的指针来保存参数的返回值。在调用 initgraph()后，可以检查这个参数的值来确定 BGI 设置了哪种图形方式。最后一个要求是必须指出 BGI 驱动程序的路径。

当书写图形程序时，initgraph()必须和 closegraph()例程一起使用。必须在程序的末尾使用 closegraph()函数，以便关闭 BGI 系统并把监示器的视频方式恢复到调用 initgraph()以前的情况。用户可以很容易地调用它，因为并没有参数，也不返回任何值。

1.1.2 编写基本的 BGI 程序

为了回顾上节的内容，让我们写一个简单的使用 initgraph()和 closegraph()的图形程序。下面这个程序也许是用 Turbo C++写的最短的图形程序 3。

```
#include <graphics.h>                  // Turbo C++ graphics header
main()
{
  int gdriver = DETECT;                // Use autodetection of
                                       // the graphics adapter
  int gmode;                               // For autodetection use a
                                           // variable placeholder
                                           // for the graphics mode
  initgraph(&gdriver,&gmode,"");           // Initialize graphics
  // ... put drawing commands here      ...
  closegraph();                            // Exit graphics mode
}
```

如果用户是一个熟练的 C 程序员，但对 C++却不熟悉的话，这个程序的第一个不同之处就是它进行注释的语法不同。在 C++中字符／／告诉编译器后面紧跟的是注释字符串。这些字符与标准 C 的注释分隔符／＊和＊／不同。例如，C 的注释语句为:

```
/* Exit graphics mode */
```

与 C++的注释效果相同:

```
// Exit graphics mode
```

虽然 Turbo C++支持这两种方式，但本书中将使用 C++的注释规则。

如果不加入任何绘图命令就运行该程序，那么只能看到屏幕闪烁几下就完了。屏幕的闪烁是由于 PC 机从文本方式转换到图形方式，然后又转换到文本方式造成的(假设在调用 initgraph()以前屏幕是文本方式)。

让我们仔细地看一看这个程序。代码首先有一条#include 语句来包含 graphics.h 头文件。每个图形程序必须包括这条语句。记住 graphics.h 文件包含了各种#define 语句和

BGI 函数的函数原型。如果用户还没有接触过这个文件，那么应借助 Turbo C++的内部编辑器看一看它。

调用 initgraph()自动配置视频适配器，因为这里使用了 DETECT 作为第一个参数。指示视频模式的第二个参数没有被赋值；本例中它只是占个位置而已。再次注意前两个参数是作为指针传递给整型值的。在调用了 initgraph()后，可以检查 gdriver 和 gmode 看一看加载了哪种视频适配器，以及初始化成哪种图形方式。第三个参数指出了 BGI 驱动程序所在的目录。如果目录不正确，那么程序将无法初始化图形方式，也就不能显示任何图形。在下一节中我们将讲述加载图形文件时如何检查错误。

用户应该明白本例程对图形初始化并不做任何错误检查。因此，如果图形初始化失败，程序将不进入图形方式。BGI 提供了几个错误处理函数。在这些函数中，graphresult()可以得到最后调用的图形函数的错误状态。Turbo C++的另一个函数 grapherrormsg()截获 graphresult()返回的值然后显示出适当的错误信息。在下面的程序中我们将使用这些错误检测函数，教会你如何写一个更大的程序。

1.2 错误检测

图形程序中最普通的错误是找不到 Turbo 图形驱动程序。记住 initgraph()的第三个参数指出了 Turbo C++图形驱动程序的路径。如果路径错误从而找不到图形驱动程序，就不能进行图形初始化。在第一个例程中，图形驱动程序的路径被设置成为空串，这表明 BGI 驱程序必须在当前目录中。如果 BGI 文件不在运行图形程序的目录中，用户必须设置它们的存贮目录。

让我们修改一下前面的例子来检测可能发生的错误。键入下面的程序，编译并执行它：

```
// errtest.cpp -- Demonstrates how errors can be detected when
// initializing the graphics system.
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

main()
{
  int gmode, gerror, gdriver = DETECT;

  initgraph(&gdriver,&gmode,"\\tc\\bgi");  // Autodetect and initialize
                                           // graphics adapter and mode
  gerror = graphresult();                  // Get error flag value
  if (gerror < 0) {                        // If it's negative then an
                                           // error has occurred. Use
                                           // grapherrormsg() to print an
                                           // appropriate error message.
    printf("Graphics initialization error: ");
    printf("%s \n",grapherrormsg(gerror));
    exit(1);                               // Since error, quit program
  }

  // If no graphics error has occurred, then print a greeting
  outtext("Hello graphics world!      Press any key to exit ...");
// Use getch() to wait for the user to strike a key, otherwise the
// program will immediately execute closegraph() and the screen will
// be cleared and switched to text mode.
```

```
    getch();
    closegraph();                          // Exit graphics mode
    return(0);                             // Return no error value to DOS
}
```

该程序作了全面的错误检测．如果一切正常，显示信息:

```
Hello graphics world!     Press any key to exit ...
```

但如果发生问题，将会显示一个适当的错误信息，程序终止．为了保证正确设置了图形环境，在进入下一节之前要运行此程序．

该程序中另一个新的函数是 outtext()．它在图形方式下向屏幕写一个正文字符串．是 BGI 提供的两个专用图形正文输出函数中的一个．另一个函数是 outtextxy()，允许用户在屏幕的特定位置显示一个正文字符串．在第三章讨论 BGI 字形和正文时，我们将详细地讲解这两个函数．

1.3 坐标

用户在工作时应该知道它在哪里，要到哪儿去．在图形方式下，我们使用象素坐标进行定位．象素是屏幕上看到的小点，它们可以通过一个坐标系统进行定位，该坐标系统与二维数组的行、列访问方式相似．这个坐标系统称为笛卡尔坐标系统．图 1.1 绘出了如何用行列坐标引用象素．

坐标系统的范围取决于所使用的视频适配器．幸运的是，有一些处理象素坐标的标准方法，它可以用于任何 BGI 支持的视频适配器．首先，PC 机上的所有图形方式的坐标都是以屏幕的左上角开始的，坐标为(0,0)．象素坐标的最大值在屏幕的右下角，它的值取决于图形方式；但是 Turbo C++有两个函数，getmaxx()和 getmaxy()，调用它们可以得到这些值．如下面语句:

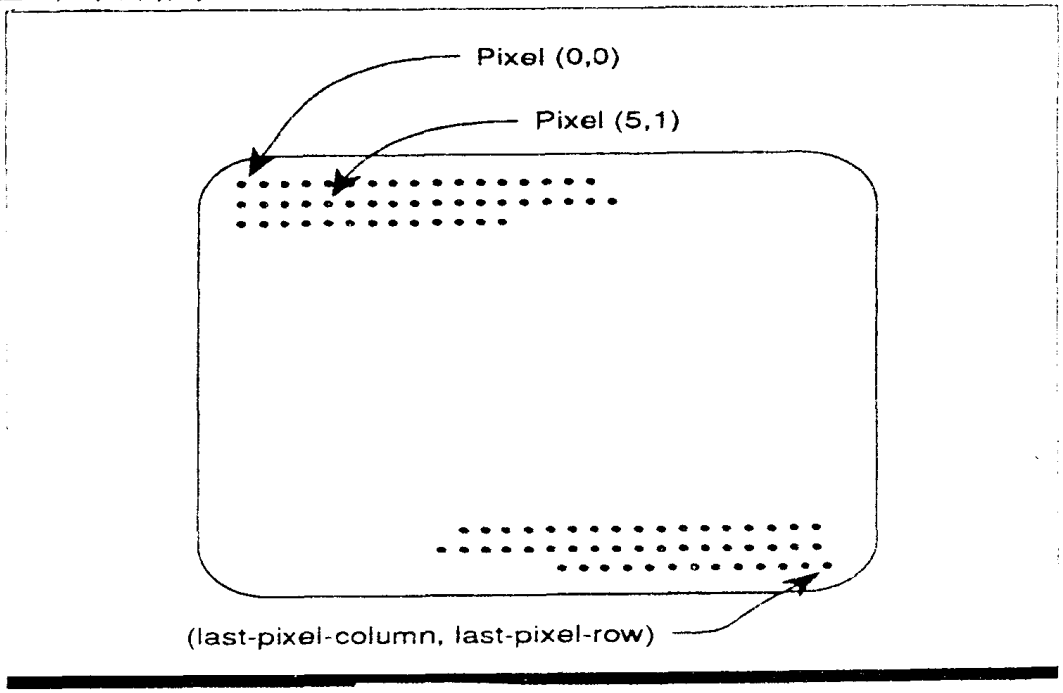

图 1.1　行和列定位

```
max_x_coordinate = getmaxx();
max_y_coordinate = getmaxy();
```

这里有一个好的编程习惯：尽量调用这些函数来编写程序，使它们可以在多种不同的图形适配器和图形方式下工作。这些函数不需要传递任何参数，返回值是 x 方向(行数)和 y 方向(列数)的象素的数目。因此，右下角坐标可以由 getmaxx(), getmaxxy() 给出。有了这个信息，就可以告诉 BGI 把物体画在屏幕的什么位置。现在我们将讲解几个绘图函数。

1.4 绘图命令

Turbo C++支持许多简单易懂的绘图命令，包括象素级的屏幕命令，以及绘制三维图形的高级函数。

在这部分中，我们将讨论一系列 BGI 的绘图函数。先从低级的 putpixel() / getpixel() 函数开始，然后再逐渐到高级例程。

1.4.1 使用象素

putpixel()和 getpixel()在同一时间只能访问屏幕上的一个象素点。putpixel()函数把某个特定位置（坐标）的象素点置为某种颜色。getpixel()函数则用来确定屏幕上任一点的当前颜色。这些函数的原型是：

```
unsigned far getpixel(int x, int y);
void far putpixel(int x, int y, int color);
```

注意 getpixel()返回(x,y)点的对应象素颜色。调用的例子如下：

```
putpixel(100,200,RED);
color = getpixel(20,50);
```

为了弄清楚这些函数是如何用于图形程序中的，让我们使用 putpixel()在屏幕上随机画 1000 个象素点，然后再用 getpixel()定位每一个象素，并把它们清除。完整的程序如下：

```
// randpixl.cpp -- This program randomly plots 1000 pixels on the
// screen and then thins them out by erasing every other pixel.
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

main()
{
  int gdriver = DETECT;
  int gmode;
  int maxx, maxy,                          // Maximum coordinates of screen
  int maxcolor;                            // This mode's max color value
  int i, x, y, color;

  initgraph(&gdriver,&gmode,"\\tc\\bgi");
  randomize();              // Initialize the random number gererator
  // Get the maximum x and y screen coordinates and the largest valid
  // color for this mode
  maxx = getmaxx();
  maxy = getmaxy();
  maxcolor = getmaxcolor();
  for (i=0; i<1000; i++) {
    x = random(maxx-1);                    // Use +1 for these since
    y = random(maxy+1);                    // random returns a value
```

```
    color = random(maxcolor+1);          // between 0 and num-1
    putpixel(x,y,color);                 // Color the pixel
  }
  // Now scan through the screen and thin out the nonblack pixels.
  // This is done by using the getpixel() command to locate the nonblack
  // pixels. Every other pixel that is found is reset to black.
  i = 0;
  for (y=0; y<=maxy; y++)
    for (x=0; x<=maxx; x++)
      if (getpixel(x,y) != BLACK) {
        if (i%2 == 0)                    // Only reset every
          putpixel(x,y,BLACK);           // other pixel
        i++;
      }

  outtextxy(0,0,"Press Any Key to Continue ...");
  getch();
  closegraph();
  return(0);
}
```

程序首先使用 BGI 的自动配置功能初始化图形方式．设置完图形方式后，通过调用 getmaxx()和 getmaxy()确定屏幕 x 和 y 坐标的最大值．另外，调用 getmaxcolor()来确定当前视频适配器所能产生的最高颜色值．因为所有颜色都在 0 和这个值之间，所以该函数告诉了我们当前视频方式的合法颜色值范围．getmaxcolor()的调用过程与 getmaxx()和 getmaxy()相似，对编写独立于硬件的程序有很大用途．在初始化过程的最后还调用了 randomize()，它激活随机数产生器．

注意在这个程序中并没有进行错误检测．虽然进行检测以保证正确初始化图形系统是一个好的编程习惯，但这里为了简单起见把它们省略了．

第一个 for 循环随机产生象素点，并以随机的位置和颜色画在屏幕上．颜色值的范围在 0（黑色）与 getmaxcolor()的返回值（通常是白色）之间．在这个循环之后屏幕上会呈现出夜间繁星的美丽图案．

下一个 for 循环遍历这些繁星并逐个把它们恢复成黑色．为了定位随机画出的点必须搜寻屏幕象素．这可以通过调用 getpixel()实现．每次 getpixel()遇到非黑的象素时，就调用 putpixel()把它置为黑色．清除过程由变量 i 来保证，当 i 为偶数时才允许 putpixel()画一个黑色象素．屏幕索引由变量 maxx 限制，它们存有屏幕坐标的最大值．这些变量要在程序的开始，通过调用 getmaxx()和 getmaxy()进行赋值．

1.5 绘制图形

现在我们知道了如何画和读象素，下一步要学习使用 Turbo C++的一些函数绘制图形．首先看一看画图形和物体外轮廓线的函数．这些函数有 arc(), circle(), drawpoly(), ellipse(), line(), 和 rectangle(). 这六个函数各有各的特点和应该注意的地方，我们将先开发一个程序，对它们进行逐个简单的介绍．

这些函数的名字很直观，并且功能比期望的要强．例如，arc()画一个弧，circle()画一个圆．circle()函数需要三个参数：圆心的 x 和 y 坐标和圆的半径．其它函数也类似．略有不同的是 drawpoly()，它要接受 x 和 y 点的多组值和点的个数，然后依次画线把各点连接

在一起。如果最后一点的坐标与第一个点相同，那么将画出一个封闭的多边形。

在画图形时 BGI 的基本绘图函数可以发挥强大的作用。用户可以控制图形的颜色，线的类型，和纵横比(在 arc()和 circle()函数中)。在第二章中，我们将详细讨论每个绘图函数。

现在写一个程序来说明如何使用这六个函数。为了使程序不依赖于设备，它显得有些冗长，但考虑到通用性，这还是值得的。

屏幕被分成六个区域来分别演示这六个绘图命令。另外，它还用 outtextxy()为每个图形加了标签。程序如下：

```
   draw.cpp -- Shows you most of the simple drawing routines in the BGI.
   These do not include the fill routines or the various line styles
   that are available.
#include <graphics.h>
#include <conio.h>

main()
{
  int gmode, maxx, maxy, gdriver = DETECT;
  int points[10];

  initgraph(&gdriver,&gmode,"\\tc\\bgi");
  maxx = getmaxx();  maxy = getmaxy();
     Draw an arc, circle, and polygon on the top portion of the screen
  // and an ellipse, line, and rectangle on the lower portion
  arc(maxx/6,maxy/4,0,135,50);
  outtextxy(maxx/6-textwidth("arc")/2,0,"arc");
   circle(maxx/2,maxy/4,60);
   outtextxy(maxx/2-textwidth("circle")/2,0,"circle");

   points[0] = maxx*5/6-20;  points[1] = maxy/4-20;
   points[2] = maxx*5/6-30;  points[3] = maxy/4+25;
   points[4] = maxx*5/6+40;  points[5] = maxy/4+15;
   points[6] = maxx*5/6+20;  points[7] = maxy/4-30;
   points[8] = points[0];    points[9] = points[1];
   drawpoly(5,points);
   outtextxy(maxx*5/6-textwidth("drawpoly")/2,0,"drawpoly");
   ellipse(maxx/6,maxy*3/4,0,360,75,20);
   outtextxy(maxx/6-textwidth("ellipse")/2,maxy-textheight("1"),"ellipse");
   line(maxx/2-25,maxy*3/4-25,maxx/2+25,maxy*3/4+25);
   outtextxy(maxx/2-textwidth("line")/2,maxy-textheight("1"),"line");
   rectangle(maxx*5/6-30,maxy*3/4-20,maxx*5/6+30,maxy*3/4+20);
   outtextxy(maxx*5/6-textwidth("rectangle")/2,maxy-textheight("1"),
             "rectangle");
   getch();
   closegraph();
   return 0;
}
```

该程序所产生的显示如图 1.2。主要的绘图函数调用为：

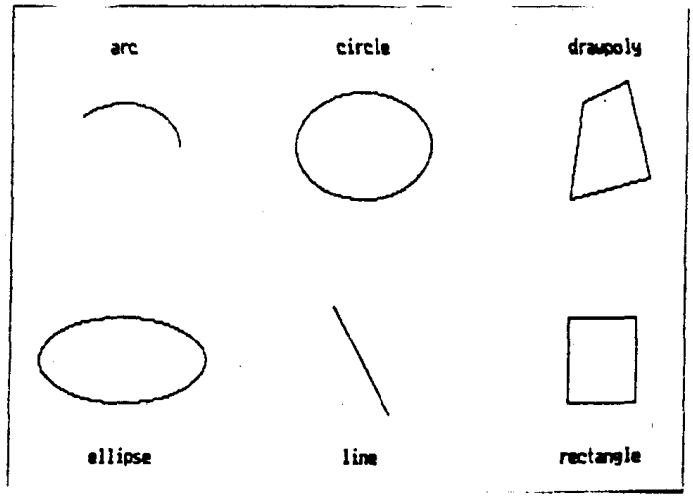

图 1.2　draw.cpp 的屏幕输出

```
arc(x,y,st_angle,end_angle,radius);
circle(x,y,radius);
drawpoly(number_of_points,array_of_xy_points);
ellipse(x,y,st_angle,end_angle,xradius,yradius);
line(x1,y1,x2,y2);
rectangle(left,top,right,bottom);
```

在程序中包含了不同的位移计算方法，它可以告诉用户 Turbo C++中典型的独立于设备代码的写法．特别注意：我们使用了 textheight()和 textwidth()函数来确定正文字符串的大小．在第三章中将仔细讨论这两个函数．

记住在例程中只给出了绘图函数强大功能的一部分．例如，前面的几个函数可以让用户设置线型，而且还可以控制绘图的颜色．但为了说明它们的基本功能，这里没有涉及到更复杂的东西．

1.6　填充图形

前面一节试验了用函数绘制物体的轮廓．Turbo C++还有一套图形函数允许绘制实体并给它们填充颜色．这些函数有 bar()，bar3d()，fillpoly()，fillellipse()，pieslice()，和 sector()．与画图函数一样，这些函数十分灵活．它们中的大多数支持不同的填充形式，颜色，甚至不同的线型．每个过程的函数原型如下：

```
void far bar(int left, int top, int right, int bottom);
void far bar3d(int left, int top, int right, int bottom,
               int depth, int top_flag);
void far fillpoly(int number_of_points, int far *array_of_xy_points);
void far fillellipse(int x, int y, int xradius, int yradius);
void far pieslice(int x,int y, int start_angle, int end_angle, int radius);
void far sector(int x, int y, int start_angle,
               int end_angle, int xradius, int yradius);
```

函数 bar()，bar3d()，fillellipse()，pieslice()，和 sector()可以绘制悦目的图形和表格．在第四章中，我们将开发几个图形显示应用程序，届时再对这些函数作详细的讨论．函数 fillpoly()与 drawpoly()的工作方式类似，所不同的是，它不仅画出多边形，而且还用当前绘图颜色和填充模式填充多边形．

现在让我们写一个程序使用这些函数画一些基本图形实体．程序把屏幕分成六个区