

· 北京科海培训中心系列教材

Turbo C++ 图形编程技巧

周少柏 查良钊 编



科学技术文献出版社

TP312
2.84

370770

北京科海培训中心系列教材

Turbo C++ 图形编程技巧

周少柏 查良钊 编



科学技术文献出版社

(京)新登字 130 号

内 容 简 介

面向对象的程序设计语言Turbo C++是开发系统软件和实用程序的良好工具。本书通过介绍Borland图形接口,探讨各种实用的高级绘图编程技巧。内容包括三维图形、动画技术、字形构造、图符编辑、上弹窗口、绘图例程和CAD程序等。书中提供经过测试的完整算法源程序,只需配备Turbo C++软件,就可在IBM PC XT, AT, PS/2或其它能显示图形的计算机系统上编译运行。

本书可供从事有关计算机科学研究的技术人员、计算机用户参阅。也可作为大专院校有关专业的参考教材。



Turbo C++图形编程技巧

周少柏 查良钊 编

科学技术文献出版社出版

(北京复兴路15号 邮政编码100038)

河北省蔚县印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

787×1092毫米 16开本 18印张 273千字

1993年5月第1版 1993年5月第1次印刷

印数: 1—5000册

ISBN 7-5023-2010-5/TP·105

定价: 16.00元

前 言

如果你想采用面向对象的程序设计方法,用Turbo C++编写高级的图形程序,本书可能是理想的阅读和参引材料。这里将详细介绍一个称为Borland图形接口的绘图工具包,它允许你充分利用面向对象的编程技术,大大简化创建二、三维个人计算机图形的工作。

众所周知,C是国际上流行的程序设计语言之一,可适用于多种型号的计算机。而最近开发的Turbo C++,继承了C的全部特性,并增添了不少为提高效率、方便用户而设置的新功能。特别是,它支持“面向对象程序设计”的新潮流,从而迅速成为深受用户欢迎的高级语言。

图形技术是计算机应用的一个重要分支,在国民经济、文化教育、军事等领域都有广阔的发展前途。使用Turbo C++提供的图形库函数,你将不难建立真实世界的各种图形应用程序。

值得一提的是,本书所提供的实例程序,都适用于虚拟图形阵列(VGA)、增强型图形适配器(EGA)、彩色图形适配器(CGA)或Hercules图形适配器。正是这种灵活性,使之非常容易移植而运行在各自的计算机——包括IBM PC及其所有兼容机的系统上。

全书共分十四章。前四章详细介绍Turbo C++中的Borland图形接口;第五章开始讨论如何开发能产生高质量表示图的程序,包括饼图、条形图、楔形图等图形显示;第六、七章涉及处理二维图形对象软件的各种技术,并增添动画效果和使用彩色调色板;第八至十四章建立交互式的图形工具,如图符编辑器、上弹窗口、绘画例程等。采用这些工具,配合安装兼容Microsoft的鼠标器,能构造像计算机辅助设计那样的交互式图形系统。

在编译本书的过程中,得到北京科海培训中心华根娣、夏非彼以及中国科学院计算所王玺玉的热情支持和帮助,借此机会,向他们表示衷心的感谢。

目 录

第一章 Turbo C++——更好的C	(1)
1.1 向Turbo C++过渡	(1)
1.2 新的语言特点	(1)
1.2.1 注解	(2)
1.2.2 说明和定义	(2)
1.2.3 类型检查	(4)
1.2.4 参引的变元	(4)
1.2.5 缺省变元值	(5)
1.2.6 直接插入函数	(6)
1.2.7 Const定义	(6)
1.2.8 重载函数	(7)
1.2.9 流	(7)
1.3 面向对象的程序设计	(8)
1.3.1 类程说明	(8)
1.3.2 友元	(9)
1.3.3 派生的类程	(9)
1.3.4 成员	(10)
1.3.5 成员函数	(11)
1.3.6 构造器和消构造器	(13)
第二章 Borland图形接口 (BGI)	(15)
2.1 初始化BGI	(15)
2.2 编写基本的BGI程序	(16)
2.3 错误检查措施	(17)
2.4 使用坐标	(18)
2.5 绘图命令	(19)
2.5.1 像素	(19)
2.5.2 绘制图表	(21)
2.5.3 填充图表	(23)
2.5.4 正文和字形	(25)
2.6 切割成型的风景画	(28)
第三章 BGI绘图函数	(33)
3.1 像素级绘图	(33)
3.1.1 绘制单个像素	(33)
3.1.2 采用各种颜色	(33)
3.1.3 CGA颜色	(35)
3.1.4 EGA和VGA颜色	(36)

3.2 绘图命令综述.....	(36)
3.2.1 画线.....	(36)
3.2.1.1 用绝对坐标画线.....	(36)
3.2.1.2 用相对坐标画线.....	(37)
3.2.1.3 设置线型.....	(38)
3.2.1.4 预定义的线图案.....	(38)
3.2.1.5 确定当前的线型.....	(39)
3.2.1.6 用户定义的线型.....	(39)
3.2.2 画矩形.....	(40)
3.2.3 对多边形工作.....	(40)
3.2.4 弧、圆和椭圆.....	(41)
3.2.4.1 画弧.....	(41)
3.2.4.2 弧的端点.....	(41)
3.2.4.3 圆和椭圆.....	(42)
3.3 动画的基础.....	(43)
3.4 填充区域.....	(45)
3.4.1 设置填充图案.....	(46)
3.4.2 用户定义的填充图案.....	(47)
3.4.3 取填充图案.....	(47)
3.4.4 用用户定义的填充图案试验.....	(48)
3.4.5 箭头键.....	(48)
3.4.6 喷流填充.....	(52)
第四章 BGI字形和正文.....	(54)
4.1 图形模式的正文.....	(54)
4.1.1 位映像字形.....	(54)
4.1.2 四笔画字形.....	(55)
4.1.3 BGI正文函数.....	(55)
4.1.4 把正文写到屏幕上.....	(56)
4.1.5 把正文写到像素位置上.....	(56)
4.1.6 一个正文显示的例子.....	(57)
4.2 Turbo C++如何存取字形.....	(57)
4.2.1 选择和装入字形.....	(58)
4.2.2 装入字形时的错误.....	(59)
4.3 建立定制的字形.....	(59)
4.3.1 使用菜单选项.....	(61)
4.3.2 使用绘图网格.....	(61)
4.3.3 使用正文版面调整.....	(62)
4.3.4 确定当前的正文设置.....	(63)
4.3.5 确定字符的尺寸.....	(64)
4.3.6 关于垂直的字符尺寸的注记.....	(65)
4.4 放大字符.....	(65)
4.4.1 把正文放入方框.....	(67)
4.4.2 有关裁剪正文的注记.....	(69)

4.5 显示字符和数码	(69)
4.6 扩展的正文处理例程	(69)
4.6.1 printf () 的图形版本	(70)
4.6.2 为笔画字形清道	(71)
4.6.3 gprintfxy()函数	(71)
4.7 使用正文输入	(71)
4.7.1 键入字符串	(72)
4.7.2 键入数字值	(72)
第五章 表示图	(77)
5.1 基本的图形类型	(77)
5.1.1 饼图	(77)
5.1.1.1 画饼片	(77)
5.1.1.2 为饼片写标签	(78)
5.1.1.3 使每一饼片不同	(79)
5.1.1.4 建立插图	(79)
5.1.1.5 饼图程序	(80)
5.1.1.6 强调一个饼片	(83)
5.1.2 建立条形图	(85)
5.1.3 三维条形图	(90)
5.1.4 楔形图	(90)
5.2 动画图	(90)
第六章 二维图形技术	(92)
6.1 屏幕坐标	(94)
6.2 屏幕和世界坐标	(94)
6.3 变换	(97)
6.3.1 平移	(97)
6.3.2 变比一个二维多边形	(97)
6.3.3 旋转一个二维多边形	(98)
6.3.4 剪切变换	(100)
6.4 矩阵守护程序	(101)
第七章 动画	(105)
7.1 间隔化	(107)
7.1.1 把一条线动画化	(107)
7.1.2 使用间隔化技术	(108)
7.1.3 使用getimage ()和putimage ()	(108)
7.2 在背景上动画化对象	(110)
7.2.1 动画化多个对象	(115)
7.2.2 getimage ()和putimage ()的限制	(118)
7.3 用调色板动画化	(118)
7.4 使用多重屏幕页	(123)
第八章 创建鼠标工具包	(124)

8.1 使用鼠标	(124)
8.2 鼠标综述	(124)
8.3 访问鼠标驱动程序	(125)
8.4 鼠标函数	(126)
8.4.1 鼠标初始化	(127)
8.4.2 附加的鼠标成员函数	(128)
8.4.3 鼠标光标	(128)
8.4.4 鼠标位置	(130)
8.4.5 鼠标按钮	(131)
8.4.6 在方框中的鼠标	(132)
8.4.7 更多的鼠标控制	(133)
8.5 增添键盘输入	(133)
8.5.1 仿真鼠标	(133)
8.5.2 初始化键盘对象	(134)
8.5.3 仿真鼠标光标	(135)
8.5.4 仿真鼠标位置	(135)
8.5.5 仿真鼠标按钮	(136)
8.6 测试你的鼠标	(146)
第九章 使用图符	(148)
9.1 为什么使用图符?	(148)
9.2 表示图符	(148)
9.3 保存图符	(149)
9.4 读图符文件	(150)
9.5 交互编辑程序	(150)
9.5.1 建立屏幕	(151)
9.5.2 建立放大的图符	(151)
9.5.3 显示原始图符	(153)
9.5.4 与用户进行交互	(153)
9.5.5 转置图符像素	(154)
9.5.6 退出图符编辑程序	(155)
9.5.7 编译此程序	(155)
9.5.8 样本图符	(155)
第十章 图形中的上弹窗口	(162)
10.1 基本方法	(162)
10.1.1 介绍gwindows类程	(162)
10.1.2 上弹窗口	(163)
10.1.3 使用堆栈	(163)
10.1.4 初始化窗口程序包	(165)
10.1.5 上弹例程	(165)
10.1.6 仔细考查gpopup()	(166)
10.1.7 保存屏幕	(167)

10.1.8 建立上弹窗口	(167)
10.1.9 消除上弹窗口	(168)
10.1.10 消除所有窗口	(168)
10.2 使用窗口程序包	(172)
10.3 测试程序	(172)
第十一章 交互式绘图工具	(175)
11.1 交互式图形程序包	(175)
11.1.1 绘图约定	(176)
11.1.2 仔细考查draw.cpp工具	(177)
11.1.3 用笔绘图	(179)
11.2 擦除	(180)
11.3 喷涂效果	(181)
11.4 画线	(182)
11.5 画多边形	(183)
11.6 画矩形	(184)
11.7 画圆	(185)
11.8 画椭圆	(186)
11.9 画弧	(187)
11.10 杂项绘图支援	(188)
第十二章 画画程序	(201)
12.1 画画程序综述	(201)
12.1.1 使用屏幕对象	(204)
12.1.2 建立环境	(205)
12.2 画画函数	(205)
12.3 下拉菜单	(206)
12.4 改变填充类型	(207)
12.5 用户交互作用	(207)
12.6 编译画画程序	(208)
12.7 使用画画程序	(208)
12.8 增强画画程序	(208)
12.9 进行试验的一些想法	(209)
第十三章 CAD程序	(220)
13.1 画画与画图	(220)
13.1.1 设置屏幕	(221)
13.1.2 对象表	(222)
13.2 画各种对象	(224)
13.2.1 画线	(224)
13.2.2 画多边形和圆	(226)
13.2.3 作为图形对象的正文	(226)
13.2.4 显示图形对象	(227)

13.2.5 删除图形对象.....	(227)
13.3 复制函数	(228)
13.4 旋转命令	(228)
13.5 修改绘图次序	(228)
13.6 选择和移动一个对象	(229)
13.7 访问gobjlist中的成员函数	(230)
13.8 扩充CAD程序	(231)
13.9 编译CAD程序	(231)
第十四章 三维图形	(254)
14.1 增加第三维	(254)
14.1.1 使用摄影机模型	(254)
14.1.2 一些三维的对象	(255)
14.2 从世界坐标向眼坐标变换	(256)
14.3 在三维中的裁剪	(258)
14.4 透视投影	(259)
14.5 对象文件	(260)
14.6 显示三维对象	(261)
14.6.1 设置观察参数	(261)
14.6.2 编译3d.cpp程序	(261)
14.6.3 使用三维程序	(262)
14.6.4 一些样板对象	(262)
14.7 扩充三维程序	(264)
附录 BGI函数参考	(275)

第一章 Turbo C++——更好的C

C是美国电话电报公司(AT&T)所属的贝尔实验室(Bell labs)开发的程序设计语言。它以简洁的表达方式、强有力的控制流程、灵活的数据结构以及丰富的操作符而著称,目前已成为国际上流行的编程工具。它的许多优点正被人们逐渐认识,从而应用越来越普遍,受到各方面的关注和重视。

本章概述与C兼容的Turbo C++,它支持面向对象的编程法,提供集成化的程序设计环境,使得应用软件更容易生产,程序更加可靠。人们可以利用预先编制的小型程序(对象软件)来构筑各种大型程序,应用程序已经变成真正的对象——封装式的代码和数据。重复地使用这些程序模块,使得那些过去十分困难和耗时的高级绘图任务,现在变得简单多了。

1.1 向Turbo C++过渡

美国Borland International计算机公司研制成功的Turbo C++,是比C更完善的语言版本。如同它的名字所示,它自然是导自于C。其设计思想是从C向上兼容,所有用C语言编写的程序,只需稍加修改,即可用Turbo C++编译程序处理。但采用Turbo C++,将具备更多的优点。例如,对函数变元和变量赋值的强类型检查,可大大减少编程的错误。显然,它更适合于编写大型的程序。

Turbo C++语言相对复杂一些,但却有助于你写雅致的程序。如果你已经熟悉C,最好用Turbo C++取而代之,它将使你成为更好的C程序员。如果你已学过其它算法语言,如FORTRAN、Pascal等,又要想学习别的语言,请选Turbo C++。

随着个人计算机演变得越来越复杂,编程已不是很容易的任务,需要各种软件工具的帮助。Turbo C++支持面向对象的程序设计,提供丰富的库函数;生产效率高,可移植性强,它最终将成为主流语言之一。不少软件厂商已推荐用它来写系统软件 and 应用程序。

学习Turbo C++并不难,尤其是已有C语言基础的读者。然而,它学习容易精通难。为更快地掌握它,最好的办法是多多读、写有关的程序,并在计算机上反复实践。Turbo C++是针对大型程序的。它使用典型的预处理方式,增加了一些额外开销,编译时间相对长些。但是,这种损失可以从减少错误、方便调试而得到补偿,尤其是在开发大型程序时更能显示出其优越性。

1.2 新的语言特点

对C语言进行了一些必要的扩充,如对函数变元和变量赋值的强类型检查,重载函数等等。它们简化了编程、减少错误并使之更适合于大型程序设计。而大程序中许多严格的接口准则是需要遵守的。下面对一些主要特点分别作介绍。

1.2.1 注解

C++中最寻常而又最受欢迎的新特点之一是双斜线 (//) 注解。这个注解指示符告诉编译程序忽略掉任何跟随的正文，直到该行的末尾为止。仍然支持旧式的C注解，它用/*开始，而以*/结束。//注解的极大优点是，除了比较容易使用之外，还在于它们可以嵌入旧式的注解块中。例如，考虑下面的函数和它旧式的注解：

```
int foo(int a, int b){
    /* here is a comment about the function */
    return a + b;
}
```

如果你想用旧式的注解把整个函数定义注出，那可能做不到，第一个注解行末尾的终止注解记号会提前结束较大的注解信息块。旧式注解是不能嵌套的。过去，为把整个函数或包含注解的大块代码注出，C程序员只好用#ifdef伪指令，如下面的代码所示：

```
#ifdef NEVER
int foo(int a, int b){
    /* here is a comment about the function */
    return a + b;
}
#endif
```

如果用双斜线重写上述的函数，则很容易用旧式注解把整个函数注出，如下所示：

```
/*
int foo(int a, int b){
    // here is a comment about the function
    return a + b;
}
*/
```

1.2.2 说明和定义

C++要求所有函数和变量必须在使用前定义。说明是一种陈述，它告诉编译程序有关变量或函数的信息。对于变量，说明指出它的名字和数据类型。对于函数，说明指定函数的名字、它的返回类型以及函数变元（如果有的话）的数据类型。说明不对变量和函数分配任何空间，它只不过是指定变量或函数接口的一种方法。

而在另一方面，定义则为变量和函数明显分配空间。说明描述接口，定义描述如何实现。

变量的说明和定义往往组合在同一个语句中。例如，下面的语句说明和定义变量。请注意，变量可以用定义语句初始化。

```
int foo = 5;
Rect theRect;
float f = 3.66;
```

为说明一个变量而不实际定义它，你必须把extern关键字放在变量说明之前，如下面的说明语句所示。extern告诉编译程序该变量在另外的地方定义。

```
extern int foo;
extern Rect theRect;
extern float f;
```

函数的说明和定义则经常是分开的。函数通过列出它的名字、返回类型和变元的类型来说明。如下面的语句所示：

```
int foo(int a, int b);
void bar(int, short, char *);
char * batz(void);
```

有关上述的函数说明可提出几点：首先，在函数定义中的变元表可以不包含变元的名字，但必须指出每一变元的类型。所以，对C++而言，下面的函数说明全是等价的：

```
int foo(int, int);
int foo(int a, int);
int foo(int, int b);
int foo(int a, int b);
```

对函数说明的另一提示是返回类型可以为void。void函数不返回任何值。

最后，请注意，变元表也可以为空（void）。它指示该函数不取任何变元，你也能简单地通过使变量表为空白而表明该函数不取任何变元。于是，下面的两个函数说明是等价的：

```
char * batz(void);
char * batz();
```

函数定义提供与函数说明相同的信息，但增加定义该函数实现的代码块，如下面的函数定义所示。当C++遇到函数定义时，它将使用函数块产生实现该函数的真正代码。

```
int foo(int a, int b){
    return a + b;
}
```

那么，什么时候说明和什么时候定义呢？C++程序通常可分解为几个独立的源文件。这些文件被分别编译，然后连接在一起以产生最后的应用程序。把有关的函数和变量放在同一个源文件中，能使你在其它的程序设计方案中重用这个文件。例如，假设你已定义了一组函数，它们用欧姆定律计算电阻、电压和电流的关系，那末可以说明为下面的函数：

```
float GetVolts(float ohms, float current);
float GetCurrent(float volts, float ohms);
float GetOhms(float volts, float current);
```

这些函数的说明放在文件elec.h中。函数的定义则放在文件elec.cp中。实现的文件elec.cp被编译一次以产生目标文件elec.cp.o，即包含该函数的实际目标代码的文件。这个文件必须和其它目标文件连接以形成应用程序。

标题文件elec.h包含函数的说明，可以出现在需要调用电流函数的源文件中。elec.h中的说明告诉C++有关函数的充分信息，以便能在调用这些函数时做所需的类型检查。但它不必知道有关这些函数的定义。当它进行这种类型检查时，函数的真正实现是无关重要的。由于单独的函数说明不会产生任何代码，你可以在若干其它文件中包含标题文件，但不必每次重编译那些函数。

当然，如果你包含了带函数说明的标题文件，就必须连接包括这些函数定义（实现）

的目标文件。否则，连接程序将指出它找不到失去的函数。

同样，对于变量，如果需要有一些全局变量包含在若干其它文件中，那么，你可以在标题文件中把它们说明为extern，然后在各自的实现文件中定义它们。用这种办法，它们只在一个文件中定义，但可以从任何包含该说明标题的文件引用它们。

C++对C的一个重大改进是：在C++中，你可以在任何普通语句能出现的地方定义变量。当在函数块中定义局部变量时，这是最有用的。在C中，函数里的所有变量说明必须出现在函数的首部。如果函数很长，这可能会发生问题，因为你要回到头部检查变量的名字。当在很长的函数定义中删去一段代码时，也可能产生问题，因为你也许忘记返回头部消除被删代码所用的那些变量定义。

C++允许正好在使用变量之前定义它们。于是，在很长的函数定义中，你可以把各自的变量定义和使用它们的代码组合在一起。这使代码更易读和可维护。

这种特色的一个普通用法是在for循环的for子句中定义循环计数器变量，如下面的代码段所示：

```
for(int i = 0; i < maxCount; i++){
```

1.2.3 类型检查

C++检查所有函数变元的数据类型。这是为了在编译时而不是留待运行时查出更多的错误。C++要求在函数说明中有全部的变元表，其目的在于能在调用函数时检查变元表中的全部变元。

强类型检查意味着在把不兼容的变量传送给函数时，C++将在发出警告或拒绝编译这些函数调用。当所传送的变量类型有可能被转换为函数所期待的类型时，C++将进行某些自动的类型转换。例如，当把浮点数传送给期待整数的函数时，C++将在把它传送给函数之前，自动将浮点数截取为整型的成分。当转换会改变该值时（如浮点数向整数转换），C++将在编译时发出警告。

强类型检查是一个很大的优点，但有时会有些限制。假如想把指针变量传送给期待长整数的函数，你清楚知道指针和长整数有同样的尺寸，但编译程序办不到，它没有标准的方法把指针转换为长整数。在要把指针作为函数变元传送时，你可以显式地把指针强制为长整数类型而解决这个问题，如下面的代码段所示：

```
char *p;           // definition of pointer variable
void foo(long l);  // declaration for function with one long arg
foo(p);            // compiler will complain !!!
foo((long)p);      // typecast makes it all OK
```

强制类型是告诉C++你明白你所做的一切的一种方法。实际上，使用C++重载函数的功能，可以定义上述例子所示函数的其它版本，以便能用指针变元或长整数变元调用它，从而消除了使用显式强制类型转换的必要性。有关细节请参见本章“重载函数”一节。

1.2.4 参引的变元

当在函数的变元表中说明变元时，可以在变元类型名字后面跟一个&符，以指示该变元被参引而不是赋值传送。这意味着C++将把指向变元的指针而不是把变元的副本传送给

函数。于是，你将能改变函数中变元的值，并使这些改变体现在说明为变元的变量中。

参引传送基本上和Pascal的VAR变元特点相同。参引传送使你能在说明变元时指定变量的名字，而不必指定变量的地址。这也意味着你在函数中不用解参引指针以取得变元的内容。对于结构和其它复杂的数据类型，这是十分有用的。

考虑下面的例子。假定已如下定义一个表示矩形的数据结构，

```
struct Rect {
    short top;
    short left;
    short bottom;
    short right;
};
```

现在，考虑一个计算矩形面积的函数。在C中，你应定义该函数取一指向Rect的指针，如下所示：

```
int Area(Rect * r){
    return ((r->bottom - r->top) * (r->right - r->left));
}
```

当调用该函数时，应提供Rect变量的地址作为变元，如下面的代码段所示：

```
Rect theRect;
int theArea = Area(&theRect);
```

使用C++的按参引调用语法，可如下定义面积函数：

```
int Area(Rect& r){
    return ((r.bottom - r.top) * (r.right - r.left));
}
```

当调用这个函数版本时，可以简单地提供Rect变量的名字作为变元，如下面的代码段所示：

```
Rect theRect;
int theArea = Area(theRect);
```

参引传送语法使你避免了典型用于传送指针变元的解参引和取地址操作。参引传送还迫使调用程序分配被调用函数所用的结构。例如，如果你使用取指针变元的面积函数，就不可避免地要分配一个Rect指针变量，而它实际上并不指向有效的Rect结构，因而错误地把指针传送给Area，如下面的代码所示。

```
Rect *pRect;
int theArea = Area(pRect); // DANGER: using uninitialized Pointer!
```

1.2.5 缺省变元值

C++的另一新特色是在函数的说明中把缺省值赋与函数变元的能力。例如，假定你说明一个函数，它以指定的抽样率发音。这个函数取两个变元，一个是指向声频数据的指针，一个是指定抽样率的整数，如下面的说明所示：

```
void PlaySound(char * sound, int sampleRate);
```

进一步假定在你的系统中，一般的声音抽样率为22 000Hz。你可以为该抽样率变元

定义一个缺省值，以便该函数的调用程序不必提供抽样率变元（如果该缺省频率是可接受的）。带缺省赋值的说明如下所示：

```
void PlaySound(char * sound, int sampRate = 22000);
```

C++仅当调用程序不提供变元时才替换缺省变元值。变元表中不能多于一个变元有缺省值，但替换是基于变元位置的，所以在变元表中带缺省值的变元不能跟有非缺省的变元。换句话说，缺省变元必须是变元表中的最后一个。于是，如果像下面那样说明PlaySound函数将是不正确的：

```
void PlaySound(int sampRate = 22000, char * sound); // INCORRECT !!
```

1.2.6 直接插入的函数

C++允许你定义直接插入函数。直接插入函数不是通过普通的函数调用机构引用的，它涉及堆栈处理。这对那些经常调用的或小型的函数可能是低效的，因为它们的函数调用开销可能比函数实际所做的工作还要多。另一方面，直接插入函数体将直接替换到调用直接插入函数处的代码中。某些变元也作类型检查和替换。替换是在编译时进行的；在运行时没有实际的函数调用，只是执行组成该函数的替换代码。

直接插入函数基本上替代了C程序员通常所用的宏功能，避免了短的、频繁调用例程的函数调用开销。直接插入函数比宏功能优越，它对直接插入函数的变元有强类型检查，并返回类型。例如，在C程序中，你可以定义下面的宏以返回32位长整数的高、低16位字：

```
#define HiWrd(aLong) ((short) (((aLong) >> 16) & 0xFFFF))  
#define LoWrd(aLong) ((short) ((aLong) & 0xFFFF))
```

在C++中，你可用直接插入函数代替宏，如下面的直接插入函数定义所示：

```
inline short HiWrd(long aLong)  
{ return (short) (((aLong) >> 16) & 0xFFFF); }  
inline short LoWrd(long aLong)  
{ return (short) ((aLong) & 0xFFFF); }
```

1.2.7 Const 定义

在传统的C程序中。通常用#define语句定义常数，如下所示：

```
#define MAXSHORT 32767  
#define MINUSONE -1
```

C++提供一种更好的机构以定义常数，它允许你指定常数的精确数据类型和其值。在C++中，从前的#define语句将用下面的语句代替：

```
const short MAXSHORT = 32767;  
const long MINUSONE = -1;
```

Const定义主要创建只读变量。Const变量在定义时必须初始化。此后，将不能改变它。Const定义比#define语句优越的是Const变量有定义好的数据类型，并因而在作为函数变元时能进行类型检查。

1.2.8 重载函数

C++中的函数重载允许你定义若干种函数版本，每一版本都有不同的变元表。然后，以同一函数名字引用这几种函数定义。编译程序可通过查看变元的类型而区分不同的版本。例如，再看看本章“类型检查”一节的例子。在那里，必须执行显式的强制类型转换，以便把指针变元传送给期待长整数变元的函数，如下所示：

```
char * p;           // definition of pointer variable
void foo(long l);    // declaration for function with one long arg
foo(p);              // compiler will complain !!!
foo((long)p);        // typecast makes it all OK
```

通过重载foo以便它能接受指针变元，你可以避免必要的强制类型转换，如下所示：

```
char * p;           // definition of pointer variable
void foo(long l);    // declaration for function with one long arg
void foo(char *c);   // OVERLOADED function declaration
foo(p);              // compiler will NOT complain !!!
```

每一重载函数必须有它自己的定义。函数的每一个重载版本实际上都是独立的函数，所有函数仅共享相同的名字。通过匹配说明变元表提供的变元，C++决定调用命名函数的那一个版本。

1.2.9 流

像C一样，C++不包含任何预先定义的输入和输出操作符。所有的I/O都是由库函数处理的。C有一个stdio库，含有质量不高的printf函数。C++支持stdio，但它也定义了一个称为iostream库的新的I/O类型。流提供和stdio库一样的格式化类型和输入输出服务。然而，它还提供更好的类型检查和附加的对用户定义的数据类型的支持。

流是字节的序列。可以从流中抽出数据并通过extraction操作符(>>)把它们送入程序变量中。也可以通过insertion操作符(<<)把数据插入流中。在典型的UNIX环境中，流可以和标准的输入、输出通道相连。例如，标准的输入常常和用户的终端键盘相连，而标准的输出就是终端显示屏。于是，从标准输入流中抽出数据以检索用户在终端键入的字符，把字符插入标准输出流中以发送这些字符到终端显示屏。

iostream库定义了三个流变量——Cin、Cout和Cerr。它们对应于系统的标准输入、标准输出和标准错误输出通道。你可以在程序中使用这三个预定义的变量抽出和插入数据到标准I/O通道。例如，发送字符串到标准输出通道，可以用下面的语句：

```
cout << "This data will go to the output channel \n";
```

更为重要的是，你可以组合插入操作和用insertion操作符的自动格式功能来产生复合输出，如下面的语句所示：

```
int x = 23;
cout << "The value of x is " << x << ".\n";
```

上面的语句将产生如下输出：

The value of x is 23.

你可以看到, insertion操作符知道在把int变量x插入输出流之前, 该怎样把它改为正文表示。同样, extraction操作符知道如何从输入流取出正文表示, 并把它转换为指定的目标数据类型。例如, 你可以从输入流抽出整数, 并用以下语句把它的值放入int变量中。除了插入和抽出预定义的数据类型外, 可以重载流操作符以便对用户定义的类型操作;

```
int x;  
cin >> x;
```

1.3 面向对象的程序设计

面向对象编程法包括定义类程、建立属于类程的对象, 以及基于已有的类程派生新类程等。这些技术能产生奇异的效果。

面向对象这一革新所起的作用, 不亚于早些时候引进的结构化程序设计, 它彻底改变你编程的传统方法。推广应用这种技术, 可大大提高生产率, 使你成为更优秀的程序员。

新编程法关键性的突破在于, 它允许利用许多预先编制的小型程序(对象软件)构筑大程序。对象软件把代码和数据融为一个整体, 从而能完整描述现实世界某一对象的全貌。由于能重复使用这些程序模块, 软件生产变得相对容易, 程序质量也更为可靠。

1.3.1 类程说明

类程像一个数据结构。它有称为成员的数据槽, 有称为成员函数的函数槽。当你编写类程的说明时, 也就指定了属于该类程的对象模板。类程说明告诉编译程序每次建立该类程的对象时需多少空间, 以及如何访问该类程的成员和成员函数。

类程是C语言中结构的扩展, 它给C++增添了面向对象的特征。与传统C的结构不同之处在于, 类程用以建立对象的模板, 而C结构只用于建立数据变量。对象由数据和代码组成, 而普通的变量只存放数据。

类程提供的两个重要性质是: 封装性和继承性。“封装性”是把数据组装在类程中的技术, 以便它们能隐藏或至少是组合在一起; “继承性”是指从已有的类程创建新类程。面向对象的程序设计中继承是重要的, 因为它使人们很容易重用和共享代码。

用于说明类程的基本语法是:

```
class <classname> {  
private:  
    // Private members go here  
protected:  
    // Protected members go here  
public:  
    // Public members go here  
};
```

类程的每一成分称为类程成员, 而成员可以是数据或函数。如上述语法所示, 共有三类成员: 私有的、保护的和公共的。私有的成员类似于函数中的局部变量, 它仅能由类程的其它成员(函数)访问。就像局部变量那样, 仅能在定义它们的函数内部使用。除非另有说明, 对象的所有成员都是私有的。保护的成员可把某些数据隐藏起来, 对于外部世界(也就是类程的外面), 它们是私有的。但是, 在定义它们的类程和某些自原始

类程派生的类程内部，它们是可访问的。公共成员则对任何类程用户都是可访问的。

就这样，C++对类程中的成员和成员函数提供了三级的保护。缺省的保护级是私有。这意味着私有的成员和成员函数不能被不是该类程成员函数的任何函数访问，即所谓数据隐藏。在C++中，数据和函数都能隐藏。下一级是保护，最少限制级是公共，你可以通过在类程说明中插入关键字public、protected或private，再跟一个冒号（:），控制成员和成员函数的保护级。如果没有使用这些关键字，编译程序假定你需要每样东西都私有。但通常你应至少使某些类程成员函数为公共，否则这个类程将无法被程序中的其它部分使用。

在类程说明中，保护关键字可多次出现。例如，你可以从公共转到保护，然后又回到公共。根据你的说明反复进行转换。

图 1.1 概括了用于典型类程定义的不同成分。

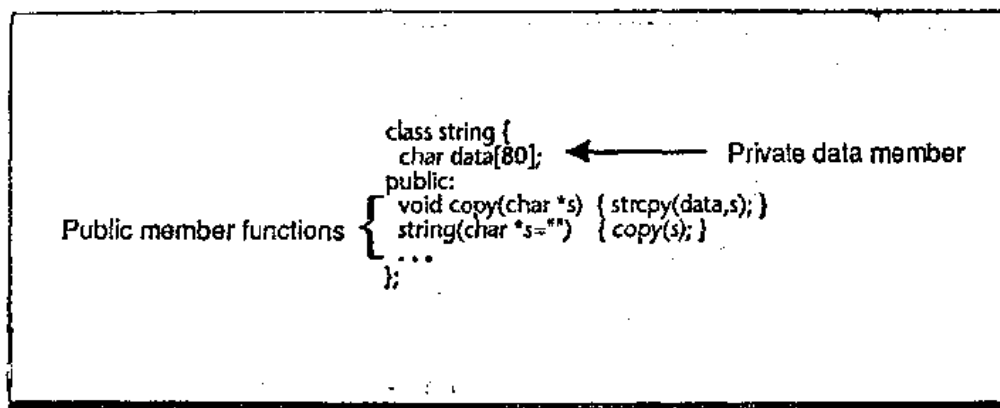


图 1.1 典型的类程定义

1.3.2 友元

在某些情况下，可能需要允许特定的函数或类程能特殊访问保护的或私有的成员。然而，你无须使成员变为公共而完全开放访问，C++提供友元关键字能让你命名特定的函数或类程，使它们有权访问保护的或私有的成员和成员函数。换句话说，这使你能把特殊的访问权限授予特定的函数或类程，但不放弃保护或私有成员享有的正常保护。

访问类程成员和成员函数就像访问数据结构元素一样。

1.3.3 派生的类程

面向对象的程序设计最重要的一个方面是继承的概念。一旦定义了一个类程，你就能从这个类程派出另一个类程。派生的类程继承了原始类程的全部成员和成员函数。原始类程（从它可派生出新的类程）就是双亲或基础类程。

派生的类程具有它的双亲类程的全部特征。典型地，你可以从其它类程派生出一个类程，使得能在派生的类程中增添或改变双亲类程中选出的某些部分。当派生新类程时，有两种方法作改变。第一，可以把新的成员或成员函数增添到新类程中来扩展双亲类程。这些新的成员或成员函数就成了派生类程的一部分。它们是继承自双亲类程的成员或成员函数之外的部分。第二种修改派生类程的方法是重设来自双亲类程的成员函数。当你重设成员函数时，要为函数提供新的定义。C++将对该派生类程使用这个新的函数定义，但原始

的定义对双亲类程仍然有效。

假设我们已经有一个代表矩形的简单类程说明，如下所示。

```
class TRect {
public:
    // these are the data members
    short fTop;
    short fLeft;
    short fBottom;
    short fRight;
    // these are the member functions
    short Area(void);
    Boolean PointInRect(Point thePt)
};
```

这个类程说明给出了类程的名字TRect，然后列出该类程的成员和成员函数。关键字public:指定成员和成员函数的保护级。

现在，你可以用下面的说明从原始的TRect类程派生出圆角的矩形类程：

```
class TRoundRect : public TRect {
protected:
    // add some new members
    short fHOval;
    short fVOval;
public:
    // and override the area member function
    short Area(void);
};
```

TRoundRect是从TRect派生出来的。这用单个的冒号跟随类程名TRoundRect来指示。说明的第一行还包含关键字public，它说明TRoundRect是公共地从TRect派生出来的。正因为TRoundRect是公共地从TRect派生的，在TRoundRect类程中的对象可以像TRect类程中的对象一样处理。即，你可以创建一个TRoundRect对象并使用它调用或访问TRect类程定义的成员函数或成员。如果类程派生的第一行不使用public关键字，则根据缺省，该类程是私有地派生的。这时，派生类程的对象也就不能像它们是双亲类程的对象那样使用。对于私有地派生的类程，只有那些在派生类程中被定义（或重设）的成员和成员函数可以被访问。私有地派生的类程的成员函数可以访问双亲类程的成员和成员函数，但派生类程的对象的户不能使用派生的对象访问双亲类程的成员和成员函数。

TRoundRect还增添了两个新的成员，fHOval和fVOval，用以指定矩形的圆角形状。TRoundRect继承了来自TRect的说明矩形的角的四个原来数据成员。最后，TRoundRect重设Area成员函数，以便更精确地计算圆角矩形的面积。

1.3.4 成员

类程说明描述组成类程的成员。被建立的类程的每一对象都得到一份全部类程成员的副本。即，类程的每一实例都有它自己私有的成员副本。例如，如果你建立了两个TRect对象（如下面的代码段所示），则每一对象将有各自的fTop, fLeft, fBottom和fRight成员的副本。

```

TRect rect1;
TRect rect2;
rect1.fTop = 0;
rect1.fLeft = 0;
rect1.fBottom = 100;
rect1.fRight = 300;
rect2.fTop = 20;
rect2.fLeft = 30;
rect2.fBottom = 400;
rect2.fRight = 100;

```

对象的成员使该对象得以维持它自己的状态，独立于已经建立的同一类程的任何其它对象的状态。

如果你用static关键字说明一个类程成员，那么，这个成员就可被该类程的所有对象共享。这时，只有唯一的一个静态成员的副本。于是，如果一个对象设置了该静态成员的值，该值就反映在这个类程的所有对象中。静态成员类似于全局变量。它如同一般类程成员，也遵守访问和保护规则。

下面的说明把静态成员添加到TRect类程中，

```

class TRect {
public:
    // static member
    static short fNumRects;
    // these are the data members
    short fTop;
    short fLeft;
    short fBottom;
    short fRight;
    // these are the member functions
    short Area(void);
    Boolean PointInRect(Point thePt);

```

静态成员在类程说明中描述。它们必须单独定义。像对全局变量那样定义并初始化静态成员是好的习惯。下面的代码示出怎样为TRect类程定义和初始化fNumRects静态成员：

```

short TRect::fNumRects = 0;

```

一旦用这样的方法初始化之后，静态成员可以被任一TRect的成员函数修改。从类程成员函数中访问静态成员的语法与对非静态成员完全一样。在类程的外面，可以通过在静态成员的名字前写上类程的名字和两个冒号来访问它，如上述定义语句所示那样。当然，从类程外面来访问还与该静态变量的保护级（公共、保护或私有）有关。

1.3.5 成员函数

特定类程的所有对象可共享该类程的成员函数。当每一对象具有各自的成员副本时，如果对每一对象都给出实现该类程的成员函数的代码副本，那就太低效了。于是，当你为对象软件调用成员函数时，将使用定义全类程的成员函数的公用代码。

为对特定类程调用成员函数，必须有一个属于该类程的对象或指向对象的指针。用这个方法把成员函数调用和特定的对象相联系是必要的，因为成员函数常常通过访问该特定对

象的成员而改变对象的状态。

类程的每一成员函数都必须用类程说明的函数原型说明。这个原型列出该成员函数的所有变元。C++把一个额外的变元“this”添加到每一成员函数的变元表头部。这个“this”变元是一个指针，它指向正准备访问该成员函数的单独对象。例如，考虑以下对矩形类程的说明：

```
class TRect {
public:
    // these are the data members
    short fTop;
    short fLeft;
    short fBottom;
    short fRight;
    // these are the member functions
    short Area(void);
    Boolean PointInRect(Point thePt);
};
```

在C++添加了隐含的“this”变元之后，两个成员函数Area和PointInRect实际说明如下：

```
short Area(TRect * this);
Boolean PointInRect(TRect * this, Point thePt);
```

由于“this”变元是隐含的，故不必把它算在你的成员函数变元表中。同样，你也永远不用关心在从C++调用成员函数时怎样提供它，因为在调用成员函数时，编译程序会自动添加此额外的变元。

C++使用“this”来确保：在成员函数中访问类程成员，只影响用以产生成员函数调用的特定对象的成员。你可以在成员函数的定义中用“this”引用对象自身，虽然这通常不是必须的。仅当需要把对对象的引用传送给某些其它函数时，才需要使用“this”。

当说明一个成员函数时，你可以把virtual关键字放在函数说明之前，告诉C++这是一个虚拟函数。你将可能在派生类程中重设它，如以下类程说明所示：

```
class TRect {
public:
    // these are the data members
    short fTop;
    short fLeft;
    short fBottom;
    short fRight;
    // these are the member functions
    virtual short Area(void);
    virtual Boolean PointInRect(Point thePt);
};
```

虚拟成员函数可通过与它的类程关联的跳转表而被调用。这使虚拟函数比直接调用的非虚拟函数略为低效。其实，你用不着关心效率上的差异，尽管你发现正在计算密集的环境中反复调用虚拟函数，可能想使函数转化为非虚拟的。

非虚拟成员函数调用局限于编译时。即，编译程序准确知道哪一个函数对应于该成员函数调用，并省去作直接函数调用的代码。相反，虚拟函数调用局限于运行时。这意味着编译程序不能精确确定应调用哪一个函数。所以，它建立代码为该对象查找跳转表中的函数

指针。在运行时，成员函数调用引用该代码以查找函数指针，然后跳转到适应该类程的真实的虚拟函数。

静态成员函数是指用static关键字说明的成员函数。类程中的静态成员函数仅可存取该类程的静态成员。静态成员函数提供存取静态成员的方法，而不需通过该类程的实际对象。

静态成员函数没有前面描述的隐含“this”变元。这意味着它们不能存取任何该类程的非静态成员。

1.3.6 构造器和消构造器

C++的最佳特点之一是，它提供一种定义函数的方法。这些函数在创建和删除对象软件时自动被调用。在创建对象软件时调用的函数就是构造器。构造器通常用于把对象的成员初始化为它们的缺省状态。在删除对象软件时调用的函数是消构造器。消构造器通常用于消除对象软件所分配的存储。

构造器是一种成员函数，它具有和它的类程相同的名字。消构造器和它的类程有相同的名字，但前面带一个否定号（~）。例如，为了给本章前面描述的TRect类程增添构造器和消构造器，可使用如下的说明：

```
class TRect {
public:
    // these are the data members
    short fTop;
    short fLeft;
    short fBottom;
    short fRight;
    // these are the member functions
    short Area(void);
    Boolean PointInRect(Point thePt);
    // constructor
    TRect(void);
    // destructor
    ~TRect(void);
};
```

如果一个类程的构造器不是公共的，则该类程的友元才能创建这个类程的对象，因为创建对象就如同调用它的构造器。

无论构造器还是消构造器都不能被说明返回某些函数结果值。构造器可说明为取变元的，但消构造器不能带任何变元。把变元传送给构造器，是非常有用的特色，普遍在C++程序中使用。例如。刚才描述的TRect类程的构造器，用下面的方式说明将更加有用：

```
TRect(short top = 0, short left = 0,
      short bottom = 0, short right = 0);
```

当带缺省变元说明时，用户可有以下的选择：或提供矩形的坐标，或不传递变元而接受缺省值。下面的构造器定义示出如何使用变元把对象成员初始化：

```
TRect::TRect(short top, short left, short bottom, short right){
    fTop = top;
    fLeft = left;
    fBottom = bottom;
```

```
fRight = right;
```

```
}
```

类程并不要求构造器和消构造器。你可以对任何类程定义构造器而没有消构造器，或有消构造器而没有构造器，或两者都没有。

当从其它类程派生一个类程时，它继承了双亲类程的构造器和消构造器。双亲构造器在派生构造器之前被引用。消构造器按相反的次序被引用，从派生类程向上通过它的双亲链。

由于构造器不返回任何状态信息，在可能失败的构造器中做任何事情都是不明智的。构造器通常仅用于把成员初始化，不分配存储或读入资源。如果你的类程需要分配存储，则应增加一个单独的初始化成员函数，它在创建对象以后被个别调用。例如，如果TRect类程需要分配存储，你可增加一个成员函数，称为像InitRect或IRect那样的东西。然后，在创建TRect对象之后调用该函数。

另一方面，消构造器特别适合于消除已经被该对象分配的存储。

对于将用作为派生类程的基础类程的类程而言，最好采用一个虚拟消构造器。

第二章 Borland 图形接口 (BGI)

在个人计算机 (PC) 世界中, 图形技术起着越来越大的作用。如果你已经有一定的 C++ 程序设计经验, 并准备深入开发图形软件, 就会发现新型的 Turbo C++ 将是理想的工具。它提供大量的图形函数以使用户建立真实世界的图形应用。Borland 公司开发的图形接口已成熟可用。Microsoft 发表的新的 DOS 图形接口环境 (Windows 3.0) 在计算机工业界也引起重大反响。计算机用户要求更多可视性强、容易修改、使用方便的程序。传统的基于正文的程序寿命不会太长了。

为了尽快掌握并着手编写基于图形的程序, 在本章我们先熟悉一下 BGI 提供的基本工具, 简略地了解 BGI 的主要特色。将从讨论 Turbo C++ 图形程序的基本结构开始, 介绍如何使用 BGI 图形函数, 并提供一些实用的例程。

在本章的末尾, 我们将开发一个创建 fractal 的有趣程序。通过构造这个程序组合所学的内容, 最终实现能把计算机生成的风景画显示出来的最简单的 fractal 程序。

2.1 初始化 BGI

对于每一个编写的 C++ 图形程序, 通常都遵循一定的步骤。例如, 在每一图形程序的开始, 必须设置视频显示硬件为图形模式。此外, 在程序的末尾, 应该把监视器恢复为正文显示模式。所以, 图形程序中要执行的第一步是初始化图形硬件。这里的所谓“硬件”指的是视频显示适配器卡, 它必须安装在你的 PC 中以便显示图形。在复杂情况下, 可以有好几种不同的图形卡用于 PC (每一个带有不同的图形模式)。幸好, BGI 支持多种这样的图形“标准”。开始, 我们打算涉及所有的图形适配器和模式。我们将使用 BGI 初始化例程的缺省模式, 它置当前安装的图形适配器为它的最高分辨率。

用于配置计算机为图形模式的 BGI 函数称为 `initgraph()`, 它有如下的原型:

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

在文件 `graphics.h` 中包含 `initgraph()` 的函数原型, 你必须在所有的 C++ 图形程序之前包括 `graphics.h`。这个文件包含全部 BGI 成分的数据结构、常数和函数原型定义。

请注意, 在 `initgraph()` 的函数原型中, 前两个变元是指向一些整数的指针, 它们含有分别说明所用的适配器和模式的逻辑值。第三个变元指定存放 Turbo C++ 图形驱动程序文件的路径名。驱动程序文件是随同你的 Turbo C++ 销售盘一起提供的, 它带有 `.bgi` 扩展名。驱动程序文件的路径名可以表示为全路径名, 如:

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

或者, 它可以是部分路径名, 如:

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

在第二个例子中, `initgraph()` 搜索目录 `drivers`, 它是目录 `graphics` 的子目录, 而 `graphics` 则是当前目录的子目录。如果BGI图形驱动程序文件存放在你正执行的程序的一目录中, 那么, 你可以如下方式调用 `initgraph()` :

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

这里, 路径名被说明为0字符串。请注意, 当指定路径名时, 你必须用两条斜线代表一条斜线, 因为Turbo C++把单条斜线解释为专用字符。

现在, 让我们看看怎样用 `initgraph()` 选择图形模式。通过设置 `initgraph()` 的前两个变元为特定值, 你可以令它使用特定的图形驱动程序和模式。另外, 还可让 `initgraph()` 自动配置视频适配器, 我们将使用这个选项。如果第一个指针变元被置为宏常数 `DETECT`, 那么, `initgraph()` 将把图形系统配置为它的最高分辨率模式。宏常数 `DETECT` 在 `graphics.h` 标题文件中说明, 并给BGI发信号, 以便选择它的自动配置功能。

`initgraph()` 中的第二个变元不需要被初始化为任何值, 但你需要包含一个指向整数的指针, 以保存变元的返回值。在调用 `initgraph()` 之后, 你可以考查这个变元的值, 以判断BGI初始化了什么样的图形模式。作为设置PC视频适配器的最后一个要求, 你将需要指定到达BGI驱动程序文件的路径。

当编写图形程序时, `initgraph()` 若没有它的伙伴例程 `closegraph()`, 那将是不完整的。在全部程序的末尾应使用函数 `closegraph()`, 以便关闭BGI系统和恢复监视器为调用 `initgraph()` 前的视频模式。调用 `closegraph()` 是很容易的, 因为它不取任何变元或返回值。

2.2 编写基本的BGI 程序

为概括已经讨论过的初始化步骤, 让我们编写一个简单的图形程序, 它使用了 `initgraph()` 和 `closegraph()` 函数。下面的程序大概是我们能用Turbo C++写的最短图形程序了。在这里例举这个程序是为了强调典型的Turbo C++图形程序的基本结构, 你可以用它作为你的程序的框架

```
#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
}
```

如果你是一个有经验的C程序员, 但却是C++的新手, 那么, 有关这个程序你所看到的第一个不同点是用以列出注解的语法。在C++中, `//` 字符用于告诉编译程序将跟有注解字符串。这些字符扩充了标准C的注解定界符 `/*` 和 `*/`。例如, C的注解语句

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

类似于C++的注解

```
// Exit graphics mode
```

虽然Turbo C++支持两种型式，在本书中我们将使用C++新的注解约定。

如果你不添加任何绘图命令就运行这个程序，则将在屏幕上看到一对斜线，而且仅有这些。把你的PC从它普通的正文模式切换到图形模式，并随即返回到正文模式，将引起屏幕的闪烁。（假定在调用initgraph（）之前屏幕是正文模式。）

让我们仔细看看这个程序。代码从针对graphics.h标题文件的#include语句开始。每一图形程序必须包含这个语句。请记住，文件graphics.h含有各种#define语句和BGI函数的函数原型。如果你没有探索过这个文件，那么，最好能花费几分钟，用Turbo C++的内部编辑程序看看它。

调用initgraph（）将自动配置你的视频适配器，因为DETECT被作为第一变元的值传送。参引视频模式的第二参数没有给出值；在本例中它只作为位置标志符。再说一遍，请注意前两个变元是作为指向整型值的指针传送的。在调用initgraph（）之后，你可以考查gdriver和gmode，看看已装入哪一种视频适配器和初始化了哪一种模式。第三个参数指定保存BGI驱动程序文件的目录，如果你没有指明正确的目录，你的程序将不能初始化为图形模式，并且不能显示任何图形。下一节我们将向你指出，在装载驱动程序文件时如何检查这种和别的潜在错误。

你还应该小心，我们的样本程序不执行有关图形初始化的任何错误检查。因此，如果图形初始化失败，则该程序不处于图形模式，BGI提供的一对错误处理函数。一个这样的函数为graphresult（），它可得到最后调用的图形函数的错误状态。另一个Turbo C++函数是grapherrormsg（），它取graphresult（）返回的值，并显示适当的错误信息。在下一个程序中，我们将使用这些错误检测函数，以构造第一个程序的更健全的版本。

2.3 错误检查措施

当找不到Turbo图形驱动程序时，将引起图形程序中可能出现的最普遍的错误。请记住，initgraph（）的第三个变元指定Turbo C++图形驱动程序文件的路径。如果这条路径不正确，也就找不到图形驱动程序，图形初始化将失败。在我们的第一个样本程序中，图形驱动程序的路径被置为空字符串，它表示BGI驱动程序必须定位在当前目录。如果你的BGI文件不在你正运行的图形程序所在目录中，那么，就需要把路径设置到存放它的那个目录上。

让我们往下修改我们的原始例子，以便能检查出潜在的错误。下面是你应该键入、编译和执行的程序。

```
// 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
}

```

这个程序执行全面的错误检查工作。如果一切顺利，则显示如下信息：

```

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

```

但是，如果出现问题，那么，将显示适当的错误信息，并终止该程序。为确保你的图形环境能正确建立，你应该试运行这个程序。

该程序中我们还没有见过的一个新函数是`outtext()`，它用于以图形模式向屏幕写正文字符串。这是随BGI提供的两个专用图形正文输出例程之一。另一个函数是`outtextxy()`，它允许你在指定的屏幕位置上显示正文字符串。我们将在第三章更为详细地讨论这两个函数。在那里还将提供对BGI字形和正文工作的技术。

2.4 使用坐标

在图形中，我们使用像素坐标来追踪位置。“像素”是你在屏幕上能看到的非常小的点。它们可以使用坐标系统编址，类似于在二维的数组数据结构中访问行与列的方法。这种坐标系统称为“笛卡儿坐标系统”。图2.1解释如何用行和列坐标参引像素。

系统坐标的范围依赖于所用的视频适配器。幸好，有一些标准的技术可处理用于任一BGI支持的视频适配器的像素坐标。首先，PC上所有的图形模式都以坐标(0, 0)开始它们的屏幕左上角。最高的像素坐标是在屏幕的右下角。它的值依赖于你采用的图形模式；但是，Turbo C++包含两个函数`getmaxx()`和`getmaxy()`，你可以调用它们取得这些值。它们一般可用于像下面那样的语句中：

```

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

```

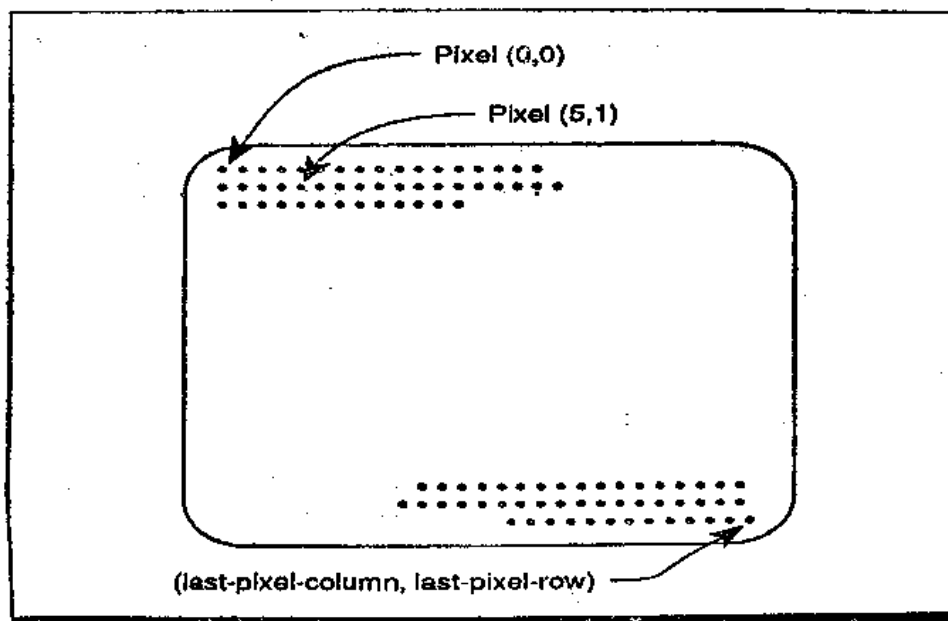


图 2.1 行和列的编址

利用这些函数，并创建能在多种不同图形适配器和模式中同样很好工作的程序，是比较好的程序设计实践。两个函数都不要求向它们传送变元，而它们则分别返回在x方向的像素数（列数）和在y方向的像素数（行数）。因此，右下坐标就由（getmaxx（），getmaxy（））给出。用这个信息，可以告诉BGI把对象画在屏幕的什么地方。现在，让我们观察某些绘图函数。

2.5 绘图命令

Turbo C++支持大量的绘图命令。它们能大大简化应用，使之从像素级的屏幕命令扩展到能画三维条形图的高级函数。

往下，我们将考查一系列BGI的绘图函数。先从BGI中最低级的绘图函数 putpixel（）和 getpixel（）开始。然后，再转到较高级的例程。

2.5.1 像素

putpixel（）和getpixel（）一次只能存取屏幕中的一个像素。putpixel（）函数把在指定位置（坐标）上的像素置为特定的颜色。而getpixel（）函数则用以确定某些屏幕位置的当前像素颜色。这些函数的原型是：

```

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

```

请注意，getpixel（）返回与（x，y）上像素颜色相对应的值。对这些函数的几个样本调用是：

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

为了看看在实际的图形程序中如何应用这些函数，让我们编写一个用putpixel()在屏幕上随机绘出1000个像素点，然后靠getpixel()定位出每一像素，以便使之稀疏的程序。

下面是完整的程序，

```
// randpix1.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;
    int maxcolor;
    int i, x, y, color;

    // Maximum coordinates of screen
    // This mode's max color value

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    randomize(); // Initialize the random number generator
    // 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);
        y = random(maxy+1);
        color = random(maxcolor+1);
        putpixel(x,y,color);
        // Use +1 for these since
        // random returns a value
        // between 0 and num-1
        // 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)
                    putpixel(x,y,BLACK);
                i++;
                // Only reset every
                // other pixel
            }

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

这个程序从使用BGI的自动配置功能初始化图形模式开始。在设置图形模式之后，调用函数getmaxx()和getmaxy()以确定最大的x和y屏幕坐标。此外，类似地调用getmaxcolor()，检索出当前安装的视频适配器可产生的最高颜色值。这个函数向我们提供现用视频模式的全范围有效颜色值，因为所有的颜色就在0和这个数之间。getmaxcolor()函数十分

像getmaxx()和getmaxy()那样工作,但不能用来编写不依赖设备的程序。初始化过程的最后一部分涉及对randomize()的调用。它为我们稍后将用到的随机数生成器提供种子。

请注意,在这个程序中不执行错误检查。虽然检查并确保图形系统已成功初始化是良好的习惯,但为了简单起见,暂时把它忽略。

第一个for循环生成随机绘制的像素,按照随机的位置和颜色在屏幕上画出它们的点。所用的颜色值从0(黑)到由getmaxcolor()返回的值(通常是白)。在这个循环之后,你的屏幕将出现繁星点缀的美丽夜空景象。

下一个for循环将开始在这些星际中遨游。通过重置它们为黑色而擦去每相隔的一个。为了找出随机绘制的星星,进行屏幕像素的搜索,可通过调用getpixel()函数来完成。每隔一次当getpixel()遇到的不是黑色的星星时,就调用putpixel()把它重置为黑色。稀疏过程由变量i来确保,它使putpixel()在每一i的偶出现时绘制它的黑色像素点。屏幕的下标受变量maxx和maxy限制(它们保存着最大的屏幕坐标)。在带有调用getmaxx()和getmaxy()的头部设置这些变量。

2.5.2 绘制图表

现在我们已经知道了怎样绘制和读像素。再进一步看看如何用某些Turbo C++ BGI例程来绘制图表。让我们从建立图表和模型的轮廓的绘图函数开始。它们是arc()、circle()、drawpoly()、ellipse()、line()和rectangle()函数。这六个函数中的每一个都有它自身独特的风格。我们现在不打算马上深入讨论它们,仅仅简单开发一个程序,作些初步的介绍,

这些函数的每一个都有比较直观的名字,而且十分像你所期望地那样操作。例如,arc()画一段弧,而circle()画一个圆。circle()函数要求三个变元:圆心的x和y坐标以及圆的半径。其它函数是类似的。唯一特殊的是drawpoly(),这个例程取x和y点的数组以及数组中点的数目,并绘制出连接这些点的线段。如果在点的表中第一个坐标和最后一个相同,则画出封闭的多边形。

基本的BGI绘图例程画图表的能力很强。你可以控制所画图表的颜色、周边的线型和所用的长宽比(在arc()和circle()函数的情况下)。在第三章,我们将更为详细地探讨每一个绘图函数。

现在让我们编写一个解释如何使用这六个绘图函数的程序。这个程序稍微有点冗长,因为需要保存所有不依赖设备的绘图函数的索引。然而这是通用化的代价。

屏幕分为六部分,在它们当中为每一绘图命令画出图表。此外使用outtextxy()正文显示函数,为每一图表编写标签。下面是该程序:

```
// 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("l"),"ellipse");
line(maxx/2-25,maxy*3/4-25,maxx/2+25,maxy*3/4+25);
outtextxy(maxx/2-textwidth("line")/2,maxy-textheight("l"),"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("l"),
          "rectangle");
getch();
closegraph();
return(0);
}

```

由这个程序产生的显示如图2.2所示。要强调的主绘图函数是如下形式的对绘图例程的调用：

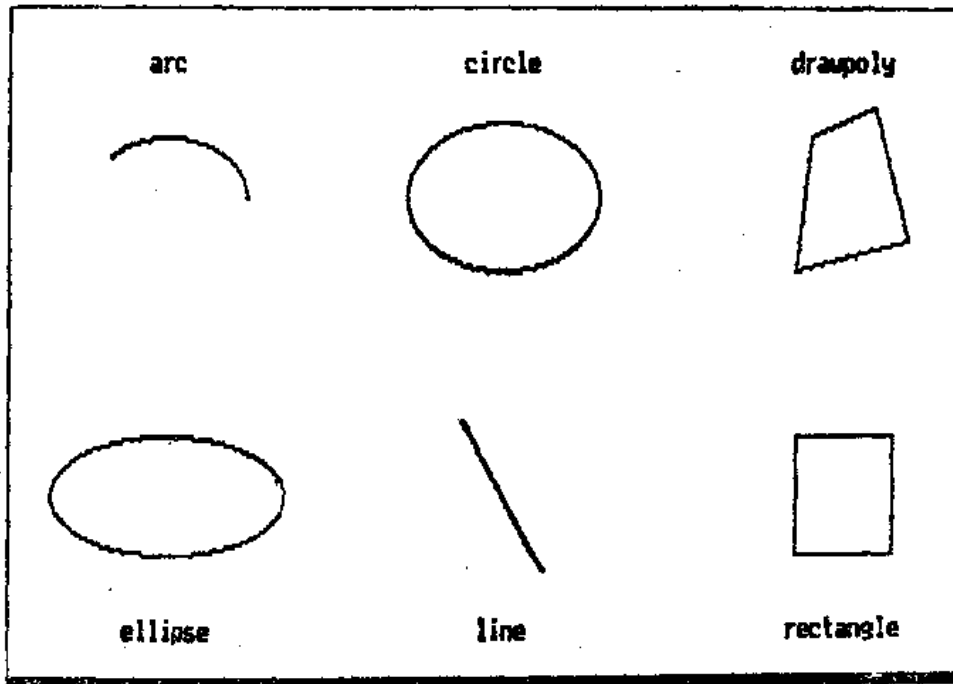


图 2.2 draw.cpp的输出

```

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);

```


程序中包含不同的计算和偏移,以便你能看到用Turbo C++生成不依赖设备的景色,通常需要什么样的代码。特别是,请注意我们正使用textheight()和textwidth()函数以确定被显示正字符串的大小。在第四章将更为详细地考查这两个函数。

请记住,在我们的样本程序中仅示出这些绘图例程的某些功能较强的特色。例如,若干先前提及的函数能允许你设置线型,并能控制用于绘图的颜色,但现在我们避开这些,以便尽快揭示它们的基本特点。

2.5.3 填充图表

在前一节中,我们试验用函数绘制对象的轮廓。Turbo C++还有一组图形函数,它们能绘制和画出对象,或用图案填充对象。这些函数是bar()、bar3d()、fillpoly()、fillellipse()、pieslice()和sector()。像图表绘制例程那样,这些函数也是非常灵活的,大都能支持不同的填充图案、颜色、甚至不同的线型。这些例程的每一个的函数原型是:

```
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);
```

为创建漂亮的图形和框图,函数bar()、bar3d()、fillellipse()、pieslice()和sector()都是有用的例程。在第五章,我们将更详细地探讨这些函数和其它的函数,那时我们还将开发若干代表性的图形应用程序。剩下的函数fillpoly()很像它的伙伴函数drawpoly()那样工作,除了它是用当前的绘图颜色和填充图案绘制和填充多边形之外。

现在,让我们编写一个程序,它使用这些函数中的每一个绘制某些基本的图形对象。该程序把屏幕划分为六个区,并在每一区域绘制不同的图表。图2.3示出由该程序产生的显示。偏移计算使得该代码有些累赘,但再重复一遍,这是通用性的代价。完整的程序是:

```
// showfill.cpp -- This program displays most of the functions
// that use the fill styles provided by the BGI. It draws a bar,
// a 3d bar, and a filled polygon on the top portion of the screen
// and a pieslice, a filled ellipse, and a filled elliptical region
// on the bottom portion.
#include <graphics.h>
#include <conio.h>

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

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    maxx = getmaxx(); maxy = getmaxy();
    bar(maxx/6-20, maxy/4-30, maxx/6+20, maxy/4+20);
    outtextxy(maxx/6-textwidth("bar")/2, 0, "bar");
    bar3d(maxx/2-20, maxy/4-30, maxx/2+20, maxy/4+20, 10, 1);
    outtextxy(maxx/2-textwidth("bar3d")/2, 0, "bar3d");
    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];
fillpoly(5,points);
outtextxy(maxx*5/6-textwidth("fillpoly")/2.0,"fillpoly");
pieslice(maxx/6,maxy*3/4,0,45,75);
outtextxy(maxx/6-textwidth("pieslice")/2,maxy-textheight("l"),
          "pieslice");
fillellipse(maxx/2,maxy*3/4,75,15);
outtextxy(maxx/2-textwidth("fillellipse")/2,
          maxy-textheight("l"),"fillellipse");
sector(maxx*5/6,maxy*3/4,25,295,75,10);
outtextxy(maxx*5/6-textwidth("sector")/2,maxy-textheight("l"),"sector");
getch();
closegraph();
return(0);
}

```

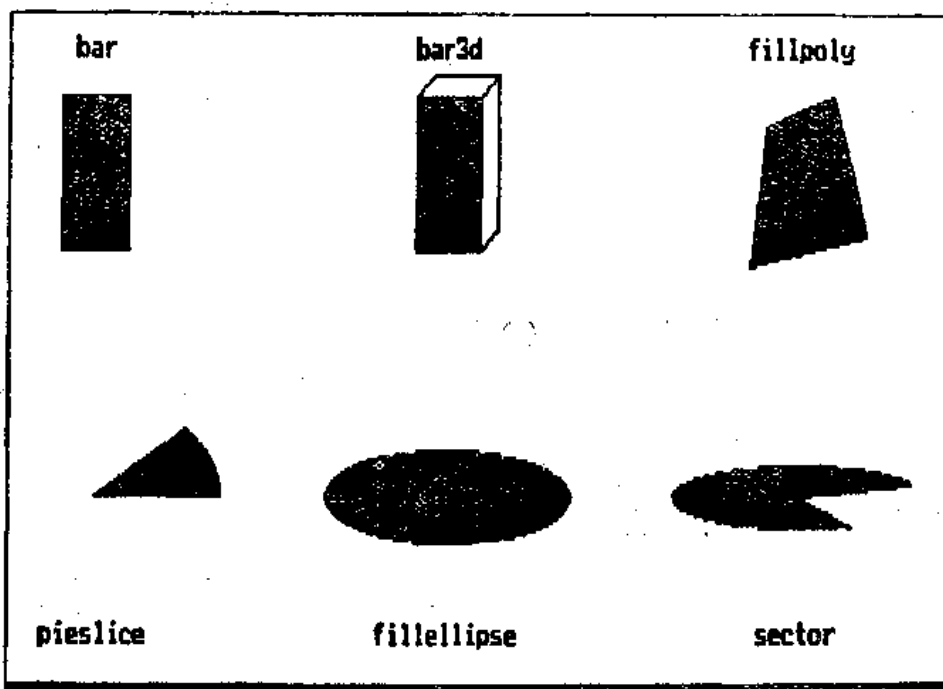


图 2.3 showfill.cpp的输出

现在，你也许会感到惊奇，填充命令怎么知道使用哪种图案呢？原则上，你每次调用填充命令之一时，它使用当前的填充类型和颜色。在我们的程序中，没有明显地设置填充类型，所以程序默认为用视频适配器可达的最高颜色涂写。Turbo C++提供一个称为 `setfillstyle()` 的函数，它可用于改变填充的类型。作为一个实例，如果我们直接在调用 `initgraph()` 之后添加语句

```
setfillstyle(HATCH_FILL,BLUE);
```

那么，我们的程序将用蓝色（EGA或VGA系统）阴影图案绘制所有对象。HATCH_FILL和BLUE的值，以及其它的颜色和填充类型，都是在标题文件 `graphics.h` 中预先定义的。在第三章，我们将介绍怎样能建立你自己的填充图案。

你还可能担心Borland是否已在它们的工具包BGI中提供了足够的填充命令。例如，有没

有圆的填充命令？实际上，若干Turbo C++图形命令都起着双重的作用。在圆的情况下，pieslice()函数可以用于涂画填充的圆。技巧是用范围满360°的起始角到结束角来调用函数pieslice()。然而，fillpoly()函数才是真正有实力的。它有极大的灵活性，你能用它填充任何形状的对象。

2.5.4 正文和字形

当涉及正文输出和字形时，BGI的功能是十分强的。传统上，IBM PC图形模式下的正文输出有点令人失望。例如，在图形模式时，IBM PC只提供固定大小的字符，在某些情况下质量也不好。但是，若使用BGI工具，你可以很容易把高质量的正文添加到你的图形显示中。BGI包含一个笔画字形程序包。它能很好地适应绘制各种高度和宽度的字符。此外，它还能绘制垂直正文。

我们的下一个任务是编写一个按不同比例且用水平和垂直格式显示单行正文的程序。这个程序使用无衬线的笔画字形在屏幕中央显示信息“Turbo C++”。该程序允许你交互地再变比正文的大小。使用s键使正文缩小，g键使正文扩大，空白键使正文在水平和垂直格式之间切换。最后，按q键将退出此程序。

下面的程序稍微偏离本章提供过的其它样本。它使用了称为“类程”的新成分。这个命名为textobj的类收集了许多元素，如变量multx、divx、multy以及函数initstyle()、display()、erase()等。这个C++成分的独特之处在于它允许我们在同一个顶蓬下组合数据和函数。在当前情况下，我们用textobj组合为显示和处理正文字符串所要求的全部数据和函数。

下面是完整的程序：

```
// showtext.cpp -- This program displays a line of text that you can
// scale as you please. The program allows the following user
// interaction:
//          s      - shrink the text
//          g      - grow or enlarge the text
//          space   - switch between horizontal and vertical text format
//          q      - quit the program
#include <graphics.h>
#include <conio.h>
#include <string.h>

const int INC = 10;
class textobj {
    int multx, divx, multy, divy; // Data components
    int newdir, currentdir;
    int cx, cy, sw, sh;
    char *string;
public: // Functions
    textobj(void);
    void initstyle(void);
    void display(void);
    void erase(void);
    char updatesize(void);
};

main()
{
```

```

int gdriver = DETECT, gmode;
textobj text;
char ch;

initgraph(&gdriver,&gmode,"\\tc\\bgi");
text.initstyle();
do {
    text.display();
    ch = text.updateSize();
    text.erase();
} while (ch != 'q');
closegraph();
return(0);
}

// Constructor function for textobj. It initializes the object.
textobj::textobj(void)
{
    multx = 100;          divx = 100;
    multy = 100;          divy = 100;
    currentdir = HORIZ_DIR; newdir = HORIZ_DIR;
    string = "Turbo C++";
}

void textobj::initstyle(void)
{
    setfillstyle(SOLID_FILL, BLACK);
    cx = getmaxx() / 2;   cy = getmaxy() / 2;
    settextjustify(CENTER_TEXT, CENTER_TEXT);
}

// Erase the existing string. Use a solid black bar to
// write over the text and erase it.
void textobj::erase(void)
{
    sw = textwidth(string) / 2;
    sh = textheight(string) * 3 / 2;
    if (currentdir == HORIZ_DIR) {
        if (sw > cx) sw = cx + 1;
        if (sh > cy) sh = cy + 1;
        bar(cx-sw, cy-sh, cx+sw, cy+sh);
    }
    else {
        if (sh > cx) sh = cx + 1;
        if (sw > cy) sw = cy + 1;
        bar(cx-sh, cy-sw, cx+sh, cy+sw);
    }
    currentdir = newdir;
}

// Resize the text based on the character pressed
char textobj::updateSize(void)
{
    char ch;

    ch = getch();
    // Resize the text as long as the scale won't go to zero
    if (ch == 'g') {
        if (divx > INC) {
            divx -= INC; multx += INC;
        }
    }
}

```

```

        if (divy > INC) {
            divy -= INC; multy += INC;
        }
    }
    else if (ch == 's') {
        if (multx > INC) {
            multx -= INC; divx += INC;
        }
        if (multy > INC) {
            multy -= INC; divy += INC;
        }
    }
    else if (ch == ' ')
        newdir = (newdir == HORIZ_DIR)? VERT_DIR : HORIZ_DIR;
    return ch;
}

// Display the text with the new text specifications
void textobj::display(void)
{
    setusercharsize(multx,divx,multy,divy);
    settextrstyle(SANS_SERIF_FONT,currentdir,USER_CHAR_SIZE);
    outtextxy(cx,cy,string);
}

```

这个程序对你来说也许有点陌生，因为我们使用了某些新的C++特色。让我们仔细来看看。请注意，我们定义的第一个函数textobj()有不寻常的外表：

```
textobj::textobj(void)
```

第一部分 (textobj) 说明与函数相关的类名；真正的函数名字跟在双冒号之后。在这种情况下，函数具有与它相关的类名同一的名字textobj。我们错了吗？不。在C++中，这种类型的函数就是“构造器”。它用以把对象初始化。当用类名形式说明C++变量时，如：

```
textobj text;
```

将自动调用具有和类名同一名字的构造器函数。（像text那样的变量称为“对象”，因为它用类名建立的。）再花费点时间考查一个textobj()构造器，你将会看到这个例程为textobj对象的每一数据成分置初始值。

我们的样本程序所包含的其它函数initstyle()、display()、erase()和updatesize()，其操作比较像传统的C函数，有关这些函数唯一不同的两点是调用它们的方式和定义函数标题的方法。

例如，用于调用C++函数的新语法是：

```
text.initstyle();
```

第一个成分 (text) 指定调用函数的对象名字。跟随它的句点把对象名字和函数名字分隔开。在这种情况下，函数名字是initstyle()。

用C++的类编程和传统的C程序设计有显著的不同。这是因为，它有“面向对象的特性”。即，不是编写由主程序或其它函数所作的函数调用组成的程序，而是用对象来控制程序的实现。当对象调用函数时，函数可使用对象的数据以执行计算。在某种意义上，你可

以把对象看成为袖珍的自含程序。

现在, 让我们观察在Turbo C++中怎样不同地说明函数。作为一个例子, 我们再看一下initstyle()的说明:

```
void textobj::initstyle(void)
```

这个说明和标准C函数说明之间的差异在于包含类程名。这个信息告诉编译程序该函数与哪个类程相关。请注意第一个成分void。它指定函数的返回类型。接着是类程名, 最后是函数名和它的变元表。

2.6 切割成型的风景画

让我们再编写一个有趣的程序以结束本章。它用到我们曾经讨论的不少BGI特色。当然, 这个程序还不足以展示在这一章所学过的每一样东西, 但它将向你说明, 使用BGI进行图形程序设计既简易又功能强。

程序使用BGI工具绘画出用切割几何生成的风景画。每一景色都设计成有山峦、海岸和地平线。上面是冉冉升起的一轮红日。山峦的轮廓和海岸线都是用切割成型法建造的。图2.4示出由程序生成的例举景色。我们将简要地讨论切割成型法。但首先让我们考查一

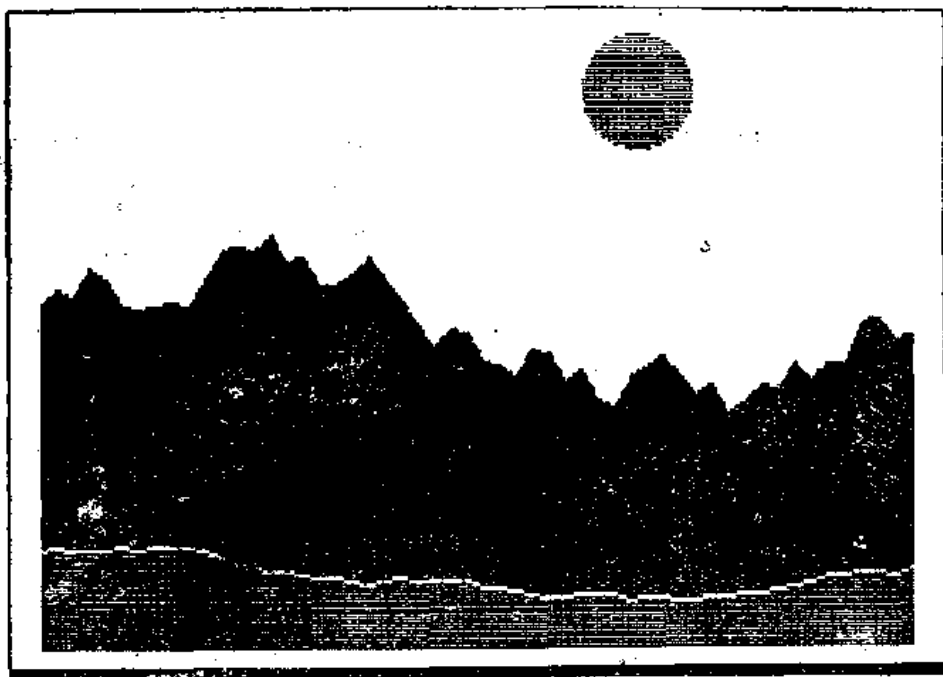


图 2.4 fractal.cpp的样板输出

个在程序中用到但还未讨论过的重要BGI函数。

天空、水和太阳全都是由称为floodfill()的函数涂画的。该函数采用当前的填充设置和绘图颜色填写任意大小或形状的有界区域。floodfill()的原型是:

```
void far floodfill(int x, int y, int border_color);
```

这里, 变元x和y指定所谓的“种子点”。而变元border-color则指示有界区域的边界颜色。

种子点应该是有界区域中某处的坐标点。

现在, 让我们快速地考查一下切割几何。切割成型技术是由数学研究人员创建的。他们试图有效地模型化大自然的复杂性。通过运用数学公式, 这些研究人员发现有可能创建带有树林、湖沼、闪电和许多其它自然景象的逼真的三维山景。

我们的实例程序使用一个fractal例程来生成沿海滨的岸区风景。山峦和海岸的轮廓实际上都是从笔直的水平线开始的, 它们在屏幕上延伸, 再引用一个称为fractal()的函数扭弯这些线段, 以便实现山峦的凹凸不平 and 海岸线的蜿蜒曲折。

fractal()函数调用subdivide(), 它实际执行切割成型。这个函数取一线段, 根据传送给它的两个端点, 计算出线段的中点, 然后使用此点把线段向上向下“弯”。线段是根据随机量来弯的。这个调整过的中点(它在后来对应于在该点画线时要用的y值)被保存在称为fretl的全局数组中。通过取中点左右的线段并在其中点弯曲它们使切割成型继续。它们的小段再被细分并以类似的方式弯曲。重复此过程直到这些小段变成小到不可再分为止。最后所得的线段就是连成一串的在y方向变化的小小段。图2.5示出在一例举的线段切

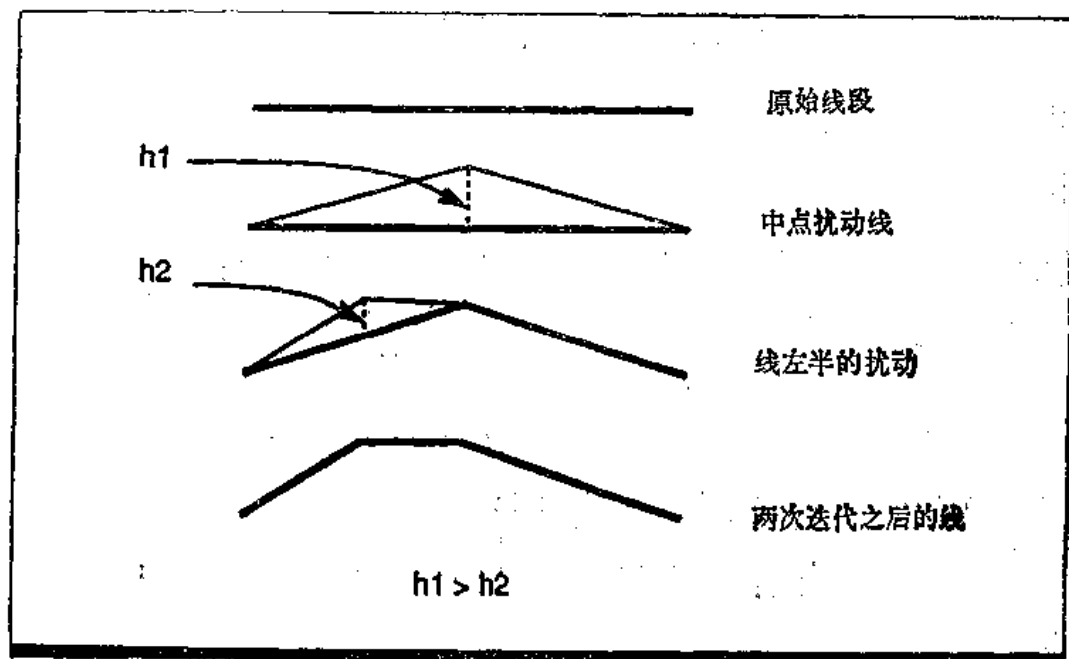


图 2.5 切割一条线的过程

割成型的各阶段。请注意, 线段弯曲量将随着线被细分为较小的片而减小。

中点被弯曲(或被干扰)的量是随机计算的。这给线条以一种合理的自然外观。但是, 有几种变量可用来帮助控制生成线条的方法。让我们从考查fractal()函数使用的变元开始。下面是它的函数原型:

```
void fractal(int y1, int y2, int maxlevel, double h, double scale);
```

fractal()中最后的变元称为scale, 它部分地定义了切割成型每一步的干扰量。此外, 它前面称为h的变元指定一个衰变因子。在每次细分一线段以便对每一越来越小的线缩小scale的大小时, 用它乘以scale中的当前干扰值。换句话说, 开始, 线段变化很大, 但当它们变得较小时, 对它们的干扰也就越来越小。这使大多数景色产生真实感。事实上, 我

们正是用scale和h这两个值的组合来控制切割线的输出的。例如，山峦的轮廓要比海岸线粗糙得多，因为它的h因子被置为0.5（1是最平滑的，而靠近0是最粗糙的）。此外，山应该较大，所以scale初始设置为较大的值50。海岸线假定十分平滑，所以它的h因子设为0.9，而scale被赋予值30.0。

切割几何的关键是应用于正在切割成型的线、表面和形状的随机性。在我们的程序中，故意采用非常简单的随机函数。事实上，它仅用称为random()的Turbo C++随机函数以确定给线段以多少干扰。random()函数基于PC的时钟返回一个随机值。结果，在每次运行时景色都生成不同的随机值。因此，我们的程序每次运行时生成不同的风景画。如果你不只一次地运行这个程序，你将明白我们所指的是什么。你可能还想试试为山峦和海岸线向fractal例程传送不同的h和scale值，以便看看这些参数对切割成型的表面将会产生什么样的影响。

```
// fractal.cpp -- This program combines the BGI and a fractal routine
// to create a fractalized landscape scene. The scene generated is
// slightly different each time you run it. Press any key to quit the
// program after the scene is displayed.
#include <graphics.h>

#include <math.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

const int MAXSIZE = 1000; // The fractalized array has this room
const int MAXLEVEL = 6; // Number of times line is cut in half
const int WATER = 1; // Color of the water on CGAC3 = blue
const int SUN = 2; // Color of the sun -- on CGAC3 = red
const int SKY = 3; // Color of the sky -- on CGAC3 = white

double frctl[MAXSIZE]; // Array used to hold fractalized lines

// Function prototypes for the routines in this program
void subdivide(int f1, int f2, double std, double ratio);
void fractal(int y1, int y2, int maxlevel, double h, double scale);
void draw_fractal(void);

main()
{
    int odriver = CGAC3; // Color Adapter -- CGAC3 = VGA
```



```

setfillstyle(SOLID_FILL,SUN);           // Draw a sun by using a
setcolor(SUN);                           // circle floodfilled with
circle(getmaxx()-100,40,20);             // the color SUN
floodfill(getmaxx()-100,50,SUN);
getch();                                 // Hold scene until key
closegraph();                             // press. Exit graphics mode.
return(0);
}

// This is the main fractal routine. It fractalizes a line in one
// dimension only. In this case, the y dimension. The fractalized
// line is put into the global array frctl. The parameter maxlevel
// specifies how much to break up the line, and h is a number
// between 0 and 1 that specifies the roughness of the line (1 is
// smoothest). and scale is a scale factor that tells how much to
// perturb each line segment.

void fractal(int y1, int y2, int maxlevel, double h, double scale)
{
    int first, last;
    double ratio, std;

    first = 0;                           // Determine bounds of
    last = (int)pow(2.0,(double)maxlevel); // array to be used
    frctl[first] = y1;                   // Use y1 and y2 as end
    frctl[last] = y2;                    // points of line
    ratio = 1.0 / pow(2.0,h);             // Defines fractalization
    std = scale * ratio;                  // decays at each level
    subdivide(first,last,std,ratio);      // Start fractalization
}

// This is the work-horse routine for the fractalization function. It
// computes the midpoint between the two points: p1 and p2, and then
// perturbs it by a random factor that is scaled by std. Then it
// calls itself to fractalize the line segments to the left and
// right of the midpoint. This process continues until no further
// divisions can be made.
void subdivide(int p1, int p2, double std, double ratio)
{
    int midpnt;
    double stdmid;

    // Break the line at the midpoint
    midpnt = (p1 + p2) / 2;               // of point 1 and point 2
    // If midpoint is unique from point 1 and point 2, then perturb it
    // randomly according to the equation shown
    if (midpnt != p1 && midpnt != p2) {
        frctl[midpnt] = (frctl[p1] + frctl[p2]) / 2 +
            (double)((random(16) - 8)) / 8.0 * std;
        // Then fractalize the line segments to the left and right of the
        // midpoint by calling subdivide again. Note that the scale factor,
        // used to perturb each fractalized point is decreased each call
        // by the amount in ratio.
        stdmid = std * ratio;
        subdivide(p1,midpnt,stdmid,ratio); // Fractalize left segment
        subdivide(midpnt,p2,stdmid,ratio); // Fractalize right segment
    }
}

// This routine displays a fractalized line. The frctl array holds
// y values. The x values are equally spaced across the screen
// depending on the number of levels calculated.

```

```

void draw_fractal(void)
{
    int i, x, xinc, l;

    l = (int)pow(2.0,(double)MAXLEVEL); // Number of points in frctl
    xinc = getmaxx() / l * 3 / 2;        // Calculate the x increment that
    moveto(0, 100);                      // is used to draw each line
    for (i=0, x=0; i<l; i++, x+=xinc)    // Draw one line at a time
        lineto(x, (int)frctl[i]);        // using y values in frctl
}

```

第三章 BGI绘图函数

任何图形系统的核心都是由它的绘图函数组成的，如我们在第二章所见，BGI提供了一组丰富的绘图函数，它们支持多种视频适配器和模式。在本章，我们准备进一步探讨这些基本的绘图函数，并向你展示如何在你的Turbo C++程序中使用它们。

我们将从基本的BGI绘图函数（面向像素的例程）开始，然后，将仔细观察怎样用BGI控制彩色。往下，我们将探讨画矩形、圆形、椭圆形、多边形以及其它形状的例程。在这之后，将简要地介绍getimage()和putimage()函数。它们可以用于动画程序中。最后，将围绕区域填充技术展开讨论并开发一个程序，以帮助我们设计自己的填充图案。

3.1 像素级绘图

如在第二章所见的那样，像素是PC图形的基本构造块。通过组合像素，我们可以绘制线条、图表、正文以及其它图形对象。当然，如果只在像素级构造所有我们的图形景象，我们面临的将是繁重的编程任务。然而，像素和处理它们的函数是图形工具包的基本元素。

BGI包含两个像素函数——putpixel()和getpixel()，它们已在第二章简略地介绍过。为了加深你的记忆，这里再重复一遍，putpixel()是在指定的屏幕位置显示一个像素，而getpixel()是在屏幕上返回某个像素的当前颜色。在下面的章节中，我们将使用这两个函数说明如何存取和显示这些像素。

3.1.1 绘制单个像素

putpixel()函数取x和y坐标，它指定像素在哪里显示，并取一整数颜色码指定其颜色。putpixel()的函数原型是：

```
void far putpixel(int x, int y, int pixelcolor);
```

显示像素的位置相对于当前观察端口的原点，虽然x和y的有效范围依赖于观察端口的大小，但初始是使用全屏幕。函数getmaxx()和getmaxy()可分别用于判断x和y的最大范围。

pixelcolor变元说明像素的颜色。实际上，pixelcolor变元说明在彩色调色板中的索引位置（我们将在下一节讨论这点），pixelcolor变元像x和y坐标那样，对每一种视频模式各有有限范围的值，范围可以从0扩展到由getmaxcolor函数返回的最大的颜色值。

3.1.2 采用各种颜色

我们已经示出如何能用putpixel()函数设置各个像素的颜色。但是，有许多比这更为灵活的控制颜色的办法。特别是，在表3.1中列出了能处理画彩色的背景和前景的函数。

但是，使用BGI的彩色系统，可能比你初步想象的还要复杂。表3.1列出的函数仅代表一部分。

为使用BGI的彩色绘图和填充功能，你需要知道如何在你的PC屏幕中显示彩色。可用的颜色存放在一个彩色调色板中。这里，“彩色调色板”就像组合一组颜色的表。我们现在来仔细考察一下。

表 3.1 主要的颜色函数

函 数	描 述
setbkcolor ()	设置现用的背景颜色
setcolor ()	设置现用的前景颜色
getbkcolor ()	返回现用的背景颜色
getcolor ()	返回现用的绘图颜色
getmaxcolor ()	返回现用彩色调色板中的最高颜色

当显示像素时，当前选择的绘图颜色就作为彩色调色板的索引。调色板实际存放以选择的颜色显示像素所用的值。这个索引系统在图3.1中示出。请注意，只有列在现用调色

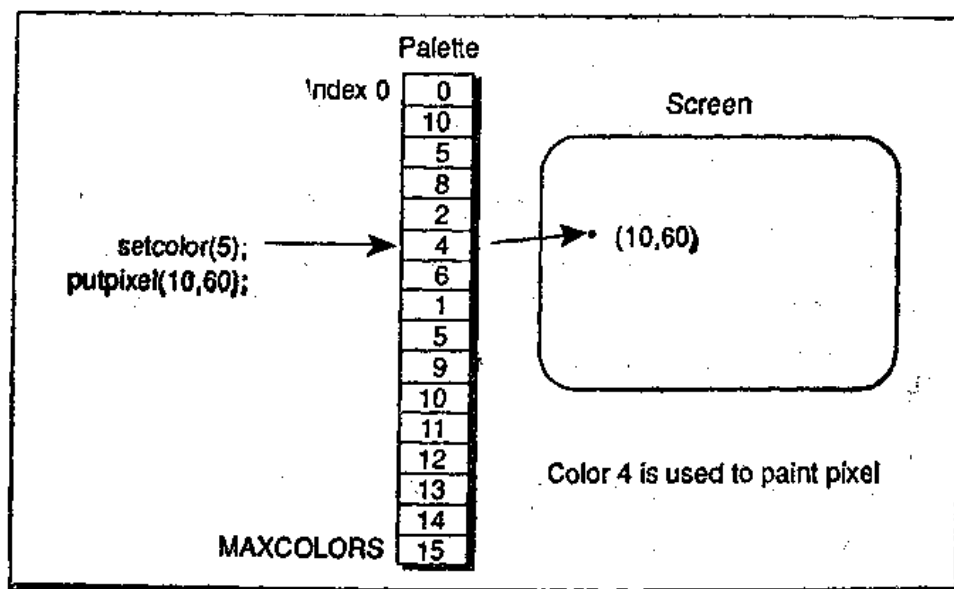


图 3.1 颜色索引系统

表 3.2 访问彩色调色板的函数

函 数	描 述
setpalette ()	改变调色板颜色
setallpalette ()	改变所有的调色板颜色
getpalette ()	返回有关彩色调色板的信息
getdefaultpalette ()	返回当前驱动程序的缺省调色板
getpalettesize ()	返回彩色调色板的大小
setrgbcolor ()	改变RGB型显示的调色板中的颜色
setrgbpalette ()	改变RGB型监视器的调色板

板中的颜色可以被显示。在调色板表中的第一项 (0) 具有特殊的意义。它作为屏幕的背景颜色。为允许你访问彩色调色板。BGI提供7种专用函数，它们在表3.2中列出。同时能被显示的颜色数目由当前的彩色调色板的大小确定。

彩色调色板的大小依赖于使用哪一种图形模式。作为一个例子，CGA适配器的彩色调色板包含4种颜色。EGA适配器的调色板包含16种颜色。由于每一个适配器工作有微小差异，我们将分别考虑每一个。

3.1.3 CGA 颜色

CGA支持低分辨率和高分辨率的模式。在低分辨率模式中，同时可显示4种不同的颜色。可用的颜色列在表3.3 (模式0到3)。在高分辨率模式 (模式4) 中，则只能显示两种颜色，而且背景颜色必须总是黑的。前景颜色可以设置为表3.4中的任一种颜色。

表 3.3 CGA 颜色 (低分辨率模式)

模式	背景颜色 (0)	前景颜色 (1, 2, 3)
0	用户可选择	淡绿、淡红、黄
1	用户可选择	淡青、淡洋红、白
2	用户可选择	绿、红、棕
3	用户可选择	青、洋红、淡灰

表 3.4 可选择的调色板颜色

值	CGA 符号名	值	EGA/VGA 符号名
0	BLACK	0	EGA_BLACK
1	BLUE	1	EGA_BLUE
2	GREEN	2	EGA_GREEN
3	CYAN	3	EGA_CYAN
4	RED	4	EGA_RED
5	MAGENTA	5	EGA_MAGENTA
6	BROWN	7	EGA_LIGHTGRAY
7	LIGHTGRAY	20	EGA_BROWN
8	DARKGRAY	56	EGA_DARKGRAY
9	LIGHTBLUE	57	EGA_LIGHTBLUE
10	LIGHTGREEN	58	EGA_LIGHTGREEN
11	LIGHTCYAN	59	EGA_LIGHTCYAN
12	LIGHTRED	60	EGA_LIGHTRED
13	LIGHTMAGENTA	61	EGA_LIGHTMAGENTA
14	YELLOW	62	EGA_YELLOW
15	WHITE	63	EGA_WHITE

对于低分辨率模式，前景颜色是预定义的。例如，如果你选择如表3.3中所示的模式2，则前景颜色是绿、红和棕。但是，你可以通过使用表3.4中列出的任一颜色选择你自己的背景颜色。

在我们离开CGA颜色这个题目之前，有必要讨论几个关键性的问题。如果你正用低分辨率模式工作，你可以用`setgraphmode()`函数选择4种颜色的集合(模式0到3)中之一。例如，调用

```
setgraphmode(2);
```

将选择模式2，3种前景颜色将是绿红和棕，然后可用`setbkcolor()`选择背景绘图颜色。

如果你使用高分辨率模式，你可用`setbkcolor()`函数设置前景颜色。但是，这可能不大正确，在高分辨率模式中，前景颜色是作为背景颜色处理的。

3.1.4 EGA和VGA颜色

EGA和VGA比CGA在产生彩色时功能更强。

这两个适配器提供可在任何时候改变的调色板。彩色调色板包含16项，根据缺省把它们置为表3.4中列出的颜色。但是，你可以从中选出总共64种颜色。表3.2中列出的函数可用于EGA和VGA，或者用以改变一个或多个现用调色板中的颜色，或者用以确定哪些颜色是当前可用的。

函数`setallpalette()`、`getpalette()`和`getdefaultpalette()`使用`palettetype`结构表示安装的图形适配器的调色板。

```
struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS+1];
};
```

`size`字段给出调色板中的颜色数目，而`colors`字段包含调色板中每一表项的实际颜色值。

3.2 绘图命令综述

虽然你可以用像素绘制差不多所有的东西，但Turbo C++仍提供大量简化此任务的绘图函数。这些绘图函数可以划分为如下范畴：

- 直线
- 多边形
- 弧、圆和椭圆

我们现在仔细观察每一这些范畴。

3.2.1 画线

虽然两点之间的直线总是最短的距离，Turbo C++提供了画线的三种方法。表3.5是一个画线和有关定位函数的表。如表所示，主要有两类画线的函数：绝对的和相对的。但是，当画线时，两者总是使用当前的绘图颜色和线型。

3.2.1.1 用绝对坐标画线

我们从观察两个函数`line()`和`lineto()`开始，讨论画线例程。它们使用绝对坐

表 3.5 画线和定位函数

函 数	描 述
line ()	从 (x1, y1) 到 (x2, y2) 画线
lineto ()	从当前位置到点 (x, y) 画线
linere1 ()	从当前位置相对方向 (dx, dy) 画线
moverel ()	根据 (dx, dy) 说明的相对量移动当前位置
moveto ()	移动当前位置到点 (x, y)

标。先看看函数line()，它取两个x, y坐标时，分别指定所画直线的两个端点。对line()的典型调用是：

```
line(5,30;200,180);
```

这个语句（如果执行的话）将从点（5, 30）到点（200, 180）画一直线。类似地，用下列的代码序列可以画有带有顶点（x1, y1），（x2, y2）和（x3, y3）的三角形。

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

另一画线技术是使用line()和moveto()函数。它们允许你通过移动离散的步画出直线。这两个函数都取一个坐标对作为变元，而且都不返回值。Lineto()函数用于从当前位置到由lineto()指定的坐标画一直线。在调用lineto()之后，当前位置更新为lineto()指定的坐标对。

moveto()函数用于移动当前位置，通常，仅当设置线段的起始点时才需要它。当前位置也可能受别的图形函数影响，例如，当你打开一个观察窗口时，当前位置总是置为（0, 0）。

使用moveto()和lineto()来画我们上述例子中提供的同样三角形，你可以用

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

在某些情况下，lineto()提供绘制直线图表的更为方便的方法。例如，如果你正为图表计算点，你要通过它但又不想保留它，这时，用moveto()和lineto()可能是理想的画线方法。

3.2.1.2 用相对坐标画线

在你的某些应用程序中，可能需要用相对坐标画线。例如，你可能需要相对于其它点或线画线。为此，提供linere1()和moverel()函数。例如，我们的三角形可用以下函数画出：

```
moveto(x1,y1);
linere1(x2-x1,y2-y1);
linere1(x3-x2,y3-y2);
linere1(x1-x3,y1-y3);
```

当然，这个例子是比较差的，因为它导致不少额外的操作。显然它不是这种情况下的最

佳解法。但是，如果我们正沿着一个曲线表面计算我们的路线，则 `linere1()` 和 `moverel()` 例程可能是最优的选择。

3.2.1.3 设置线型

在前一节中，我们探讨了几种用BGI画线的方法。BGI还允许我们指定直线的颜色、它的厚度和它的型。根据缺省，所有直线都用实线画（一个像素宽，使用当前绘图颜色）。但是，我们可以改变这些绘图参数中的每一个。

例如，在第二章我们学习了如何用 `setcolor()` 函数改变当前的绘图颜色。此外，BGI支持4种预定义的线图案和用户定义的线型。在下一节，我们将观察怎样改变线型和它的厚度。

3.2.1.4 预定义的线图案

BGI提供 `setlinestyle()` 函数以改变所有绘制线的线类型和宽度。它的函数原型是：

```
void far setlinestyle(int linestyle, unsigned pattern, int thickness);  
void far ellipse(int x, int y, int stangle, int endangle,  
                 int xradius, int yradius);
```

`linestyle` 变元（它可能被置为下面列出的5个值之一）指定当画线时所用的线类型：

常 数	值
<code>SOLID_LINE</code>	0
<code>DOTTED_LINE</code>	1
<code>CENTER_LINE</code>	2
<code>DASHED_LINE</code>	3
<code>USERBIT_LINE</code>	4

前四个宏常数说明预定义的线图案，并且在图3.2中画出。这些宏常数能使我们画实线、点线和虚线。我们还能用 `USERBIT_LINE` 常数创建我们自己的线型。

	NORM_WIDTH	THICK_WIDTH
<code>SOLID_LINE</code>	—————	—————
<code>DOTTED_LINE</code>
<code>CENTER_LINE</code>	- - - - -	- - - - -
<code>DASHED_LINE</code>	- - - - -	- - - - -

图 3.2 4种预定义的线型

在要求用户定义的线图案时使用 `setlinestyle()` 中的第二个变元。但是，如果你正使用4种预定义的图案之一，那么，应对此变元使用0值。最后的变元（`thickness`）指定所有要画的线的厚度。这里有两种可能：

假定需要用当前的绘图颜色画一系列虚的粗线。为了做到这点，我们必须调用 `setlinestyle()`：

常数	值	像素宽度
NORM_WIDTH	1	1像素厚
THICK_WIDTH	3	3像素厚

```
setlinestyle(DASHED_LINE,0,THICK_WIDTH);
```

一旦执行这个函数，则所有被画出的线都将使用这些线设置。

3.2.1.5 确定当前的线型

现在让我们观察一下，BGI如何判断当前的线设置。当需要保留当前的线设置以便它们能改变并随后恢复时，这种功能特别有用。幸好，BGI提供了函数 `getlinesettings()`，它是专门为此任务而设计的，为了解它如何工作，我们必须引进结构类型 `linesettingstype`，它用于存放当前的线设置。它在 `graphics.h` 中定义为

```
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

这个结构是重要的，因为 `getlinesettings()` 期望把这种类型的结构传送给它，并将在它的成分中返回各种线参数。 `getlinesetting()` 的函数原型是：

```
void far getlinesettings(struct linesettingstype far *lineinfo);
```

因此，为检索当前的线设置，你可以使用下列的一对语句：

```
struct linesettingstype savedlineinfo; // Declare line structure
getlinesettings(&savedlineinfo);      // Save line settings
```

稍后，为恢复这些线设置，你可以使用 `setlinestyle()` 函数：

```
setlinestyle(savedlineinfo.linestyle,savedlineinfo.upattern,
             savedlineinfo.thickness);
```

3.2.1.6 用户定义的线型

除了图3.2所示的4种预定义线型之外，BGI还提供一种定义我们自己的线型的方法。这是用 `setlinestyle()` 函数来完成的，并且涉及对它的三个变元的前两个的设置。

如前所述，`setlinestyle()` 的最左边的变元说明所用的线的类型，在此情况下，必须被置为 `USERBIT_LINE` 或值为4。但是，第二个变元实际上定义线图案，它是16位的二进制模式，用编码说明画线的方法。模式中的每一位等于16像素线段中的1个像素。如果该位被设置，则在线段上它相应位置的所有像素都用当前的绘图颜色显示。如果该位是0，则它的对应像素不画出或不变。因此，定义实线的用户定义线图案可用以下函数调用创建：

```
setlinestyle(USERBIT_LINE,0xFFFF,NORM_WIDTH);
```

16进制值 `0xFFFF` 将设置所有像素并有效地建立实线。类似地，为绘制每隔1个像素设

置的虚线，你只需使用每隔1位置为1的位模式。这可以用以下行来完成：

```
setlinestyle(USERBIT_LINE,0xAAAA,NORM_WIDTH);
```

图3.3示出多种线图案，它们可通过改动用户定义的线图案来生成。

3.2.2 画矩形

本书中我们将建立的所有PC图形都可以用已提供的像素和画线函数来产生。但是BGI还提供多种高级的绘图函数，它们能帮助我们生成图形。这些函数中的第一个是rectangle()，它用于画矩形。在graphics.h中给出的它的函数原型是：

```
void far rectangle(int left, int top, int right, int bottom);
```

四个变元指定矩形的左上角和右下角的像素坐标。当使用rectangle()时，请记住，它仅通过矩形的周边。在本章稍后的节中，我们将看到bar()和bar3d()函数如何能被用于画填充的矩形。像大多数函数那样，rectangle()使用当前的绘图颜色和线型设置，并且是相对于当前观察端口绘制的。

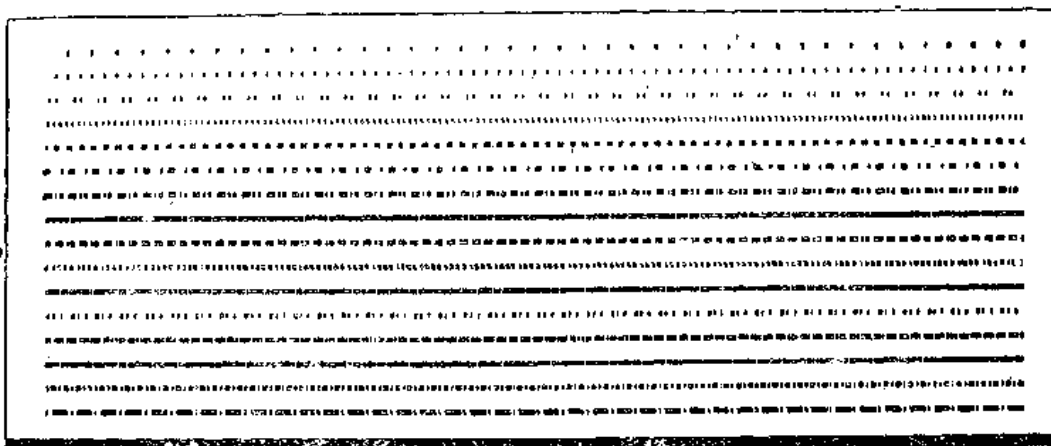


图 3.3 若干用户定义的线图案

3.2.3 对多边形工作

BGI还包含一个画多边形的通用例程，通常称为drawpoly()，它取点的数组，并使用当前线型和绘图颜色画出这些点之间的线段。事实上，drawpoly()类似于进行一系列的对直线例程的调用。它的函数原型是：

```
void far drawpoly(int numpoints, int far *polypoints);
```

第一个变元说明发送给drawpoly()的坐标数目。第二个变元指向线段要连接的交替x和y的坐标数组。

因此，如果你想画一个开口的三面图，那么，应把numpoints置为值3，并且坐标点的数组应包含对应于三条线端点的6个整数。第一个数组位置将保存第一个x坐标，再跟有它匹配的y坐标以及两个另外的x和y坐标对。我们假定这三点是(10, 30)，(300, 30)和

(100, 90)。下面的代码行将为我们画出这个图:

```
int points[] = {10,30, 300,30, 100,90};  
drawpoly(3,points);
```

drawpoly() 函数不能自动封闭多边形。如果你需要一个封闭多边形,那么多边形的最后一点必须和第一点相同。因此,为封闭此例中所提供的多边形,我们可以使用

```
int points[] = {10,30, 300,30, 100,90, 10,30};  
drawpoly(4,points);
```

请注意,点数组的第一个和最后一个坐标对是相同的,并且,为适应这点,numpoints 变元已增加1。

3.2.4 弧、圆和椭圆

至此,我们可能会怀疑BGI是否能做比画线更多的事情。幸好,它能。BGI提供了函数arc(), circle(), ellipse(), pieslice(), fillellipse() 和sector(), 它们中的每一个都可以画曲线的圆。我们将从观察这些函数当中的前三个开始。在本章稍后讨论填充区域时将提供pieslice(), fillellipse() 和sector() 函数。

每一个曲线绘图函数都使用当前的绘图颜色,但它们不完全受线型影响。换句话说,对象的周边总是画实线,然而,它们受线厚度的当前设置影响。此外,像所有至今涉及的函数那样,这些函数的坐标都是相对当前观察端口而取得的。现在让我们观察一下arc() 函数。

3.2.4.1 画弧

arc() 函数画出圆的一部分或一个完整的圆。它的原型是:

```
void far arc(int x, int y, int stangle, int endangle, int radius);
```

前两个变元说明弧的中心点的屏幕坐标。stangle和endangle变元是角度,它们指定弧的范围。这些值都是以度为单位,并且都是从三点钟位置开始按反时针方向测量的。最后的变元表示圆的半径。这个值以像素为单位测量。例如,假定我们需要画一个从水平扩展15度的弧,让我们把弧心放在(200, 100),并给出它的半径为100,产生这段弧的代码行是:

```
arc(200,100,0,15,100);
```

如果沿着这个弧的半径计算像素,你将能证实半径是沿着它的水平的100个像素。

3.2.4.2 弧的端点

如果能找出已经画出的弧的端点在什么地方,这往往是有用的。例如,我们可能经常需要把线和弧的端点连接或把一系列的弧连接在一起。为完成这点,BGI提供了函数getarccoords()。它的函数原型是:

```
void far getarccoords(struct arccoordstype far *arccoords);
```

getarccoords() 的唯一变元是类型结构arccoordstype。这个结构存放最后画出的弧的端点和它的中心位置，它在graphics.h中定义为：

```
struct arccoordstype {
    int x, y;
    int xstart, ystart;
    int xend, yend;
};
```

例如，假定我们需要画半球的周边，为完成这点，可以使用arc() 和line() 函数的组合。为适当地连接弧和线段，我们将使用getarccoords()，以精确确定弧的端点在哪儿和随后该线段应画在哪儿。下面的程序通过在屏幕中心以它的宽度的1/4作为半径，画一个半球来展示这一过程：

```
// hemi.cpp -- Draws a hemisphere at the center of the screen
#include <graphics.h>
#include <conio.h>

main()
{
    int gmode;
    int gdriver = DETECT;
    struct arccoordstype arccoords;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    arc(getmaxx()/2, getmaxy()/2, 0, 180, getmaxx()/4);
    getarccoords(&arccoords);
    line(arccoords.xstart, arccoords.ystart, arccoords.xend, arccoords.yend);
    getch();
    closegraph();
    return(0);
}
```

3.2.4.3 圆和椭圆

虽然arc() 可以用于画完整的圆（通过指定0度的起始角和360度的终止角），完成这同一任务的更好的方法是使用BGI的circle() 函数。它的原型是：

```
void far circle(int x, int y, int radius);
```

x和y变元指定圆的中心，而最后的变元是它的半径。如同arc() 的情况，圆的半径涉及像素的数目。从圆的中心开始，沿着它的水平轴到它的周边。

请注意，circle() 只画圆的周边。为了画填充的圆，你必须使用pieslice() 或另一种喷流技术填写区域（如我们以后看到的那样）。

BGI支持的另一种曲线形状是椭圆。这种形状由函数ellipse() 绘制，它的操作部分像arc()。因为后者能画出椭圆的所有部分。ellipse() 的函数原型是：

```
void far ellipse(int x, int y, int stangle, int endangle,
                 int xradius, int yradius);
```

这些变元的大部分和前面两个例程的相似。但是，为了得到各式各样的椭圆形状，这个函数允许你在y方向和x方向指定半径。ellipse() 函数不影响所画的每个椭圆的内部。如我们将看到的那样，sector和fillellipse() 函数用于画填充的椭圆区域。

3.3 动画的基础

我们暂停讨论绘图命令，以便能仔细观察一下getimage()和putimage()函数。在下一节，当我们开发一个交互式程序以建立我们自己的填充图案时，需要这些函数。

基本上，getimage()和putimage()用于处理图形屏幕的矩形区域。使用这些函数，可以很容易剪、贴、移动或改变屏幕的区域，而不用关心屏幕的存储地址。因此，我们能用getimage()和putimage()进行拼接，以便形成动画效果的任务、支持上弹窗口，或使得图形对象容易编辑和移动。

getimage()的函数原型是：

```
void far getimage(int left, int top, int right, int bottom,
                  void far *bitmap);
```

函数的前四个变元说明要拷贝的屏幕上矩形区域的左上和右下像素边界。这份拷贝的映像保留在由它的最后变元bitmap所指定的数组中。bitmap数组的大小依赖于正被保存的屏幕映像的大小和当前的图形模式。请回忆一下，每一种模式都支持要求不同存储数量的屏幕分辨率。为了确定屏幕映像的大小，BGI提供imagesize()函数。这个函数取和getimage()一样的像素边界，以计算要保存的屏幕的大小。基于这些边界，函数返回应分配给bitmap数组的字节数。例如，假定我们需要拷贝用(10, 10)和(100, 100)定界的屏幕的区域，那么首先我们需要为bitmap说明和分配空间，如下所示：

```
void *screenimage;
screenimage = malloc(imagesize(10,10,100,100));
```

其次，可以通过调用getimage()把该映像拷贝到screenimage数组，

```
getimage(10,10,100,100,screenimage);
```

现在，我们有一个屏幕映像的副本放在bitmap数组中。我们可以用putimage()函数把它拷贝到不同的屏幕位置。putimage()函数的原型是：

```
void far putimage(int left, int top, void far *bitmap, int op);
```

前两个变元指定传送给函数的映像以此对齐的左上位置。请注意，该函数不要求右下边界。因为这个信息已隐含在bitmap数组的编码自身当中。putimage()的最后一个变元称为op，它指定bitmap数组如何被拷贝到屏幕。它能取到下列值：

常 数	值	描 述
COPY_PUT	0	照原样拷贝位映像到屏幕
XOR_PUT	1	“异或”位映像和屏幕
OR_PUT	2	“或”位映像和屏幕
AND_PUT	3	“与”位映像和屏幕
NOT_PUT	4	拷贝位映像的逆到屏幕

因此，在我们的例子中，如果需要把bitmap映像拷贝到屏幕的位置(110, 10)中，

则可以使用语句

```
putimage(110,10,screenimage,COPY_PUT);
```

类似地，如果需要转换我们早些时候保存的部分映像，则可以使用语句

```
putimage(10,10,screenimage,NOT_PUT);
```

这在强调部分屏幕时是有用的。图3.4示出拷贝上面列出的选项时，每一映像所产生的效果。

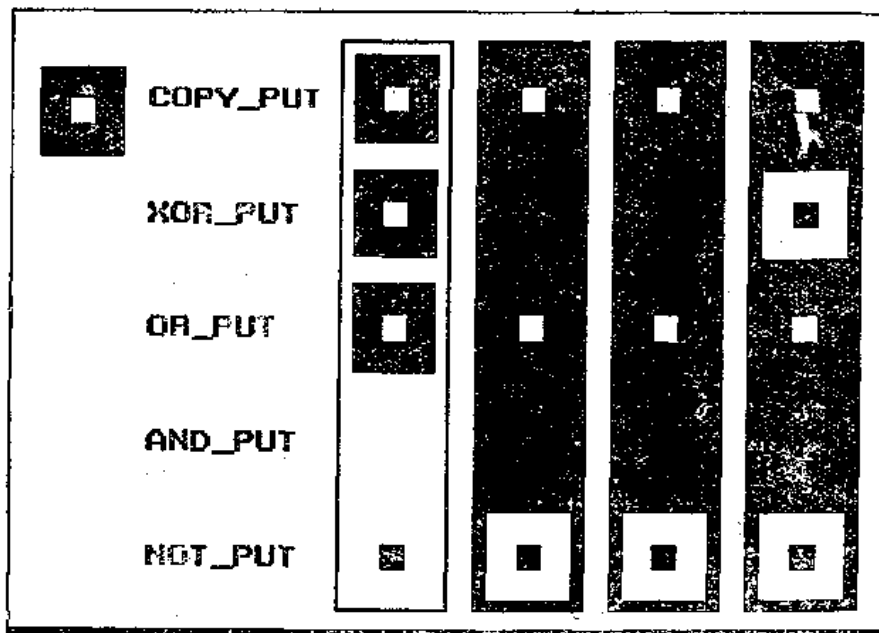


图 3.4 使用putimage()所产生的效果

putimage() 的另一极为有用的可能性是使用XOR_PUT 操作，它把 bitmap 映像和屏幕映像进行“异或”。“异或”操作在二进制一级置屏幕存储中的所有位为1。如果在位映像和屏幕映像两者对应位之一（但不是两者）为1时，如果两者的位都是0或1，那么，将是屏幕映像的相应位为0。使这种特殊的性能非常有用的是拷贝映像自身。在许多情况下，将擦去区域中的某些对象，然后重复此过程，使它重现。这对某些类型的动画可能是有用的。在本章的稍后的一个程序中，我们将使用XOR_PUT 任选项帮助横跨屏幕移动一个光标，而不用关心在移动时修改屏幕。请注意，使用“异或”功能的putimage() 操作的效果将依赖于屏幕上的当前颜色和bitmap数组中的颜色。

putimage() 支持的另外两个操作是AND_PUT和OR_PUT选择。它们可用于取得各种特殊的效果，并可组合以产生任一预定义操作所不能直接得到的其它类型的结果。

最后，当你使用getimage() 和putimage() 时应记住几点。首先，虽然这些函数是相对于当前观察端口坐标定位的，但它们不会像你所期待的那样。受观察端口裁剪的影响，如果bitmap映像的某些部分扩展超出了屏幕的界，那么，整个映像都将被裁剪。这在围绕一个区域移动对象时可能出现问题。但是，大部分时间它是精确地满足你的要求的。

当使用 getimage() 和putimage() 时可能受限的另一方面是对某些 bitmap 数组所

允许的最大尺寸, 这些函数被设计成能接受64K大小或更小些的映像。在某些情况下, 这可以认为已有足够的存储; 但是, 当处理图形屏幕时, 这种存储限制可能很快便成了问题。例如, 在许多640×200的模式中, 我们将不能一次保存整个屏幕, 因为它可能为bitmap数组要求多于64K。

3.4 填充区域

迄今为止, 我们仅观察了画对象轮廓的图形函数。Turbo C++还提供了若干别的函数, 它们能画使用某些预定义图案或用户定义的图案填充的图表。

在继续讨论下去之前, 让我们快速地观察一下列在表3.6中的这些函数。(参考第二章的程序showfill.cpp, 它展示了这些函数如何工作。) bar() 和bar3d() 函数是十分类似的。但是, 有两个主要差异。bar3d() 函数画三维的条形, 而bar() 函数仅简单地画一个填充的矩形区域。我们将在第五章更详细地探讨它们。实际上, bar3d() 还能通过置深度为0画二维条形。但是, 这里还有其它重要的差异, bar3d() 用它当前的绘图颜色画出它的区域的轮廓, 而bar() 没有任何轮廓。这种不同将在本书中多次显出其重要性。你还应注意到, bar3d() 有一个称为topflag的附加变元。如果为非0, 则函数遮蔽掉条形的顶。当你需要把若干个条形彼此在顶上堆放在一起时, 画不带顶的三维条形可能是有用的。

表 3.6 BGI的绘图和填充函数

函 数	描 述
bar ()	画不带轮廓线的填充条形
bar3d ()	画带轮廓线的三维填充条形
fillpoly ()	画填充的多边形
fillellipse ()	画填充的椭圆区
pieslice ()	画饼片并可用于画填充的圆
sector ()	画填充的、椭圆的饼片

函数fillpoly() 也十分像它的对应函数drawpoly(), 两者都将用同样的方法。唯一的差异是fillpoly() 画出传送给它的多边形, 并且填充它的内部。

最后, pieslice() 和sector() 可以分别画填充的饼片和填充的椭圆形状。每一个都有一些变元说明饼片或扇区从哪儿开始到哪儿结束。两者的角度都是相对于图表的中心点(x, y)从三点钟的位置开始。

有关pieslice(), fillellipse() 和sector() 的最后注意事项是次序。请回忆一下, BGI不直接包含能画填充圆的函数。我们建议可用pieslice() 和sector() 函数。想法是用0度的起始角和360度的终止角来画一个完全填充的圆。但是, 这里要注意的是, 这两个函数都用当前的绘图颜色画周边, 它环绕着模型并扩充到它的中心。如果内部和边界为不同颜色, 你将看到有一条线从圆心扩展到圆边界上的三点钟位置。避免这点的唯一方法是使边界和填充图案为同一颜色。另一个方法是使用floodfill() 函数, 如我们将在

这章稍后所讨论的那样。

表 3.7 预定义的填充图案

宏 常 数	值	描 述
EMPTY_FILL	0	用背景颜色填充
SOLID_FILL	1	用填充颜色填
LINE_FILL	2	用水平线填
LTSLASH_FILL	3	用细的左向右斜线填
SLASH_FILL	4	用粗的左向右斜线填
BKSLASH_FILL	5	用粗的右向左斜线填
LTBKSLASH_FILL	6	用细的右向左斜线填
HATCH_FILL	7	用淡的阴影图案填
XHATCH_FILL	8	用深的交叉阴影线填
INTERLEAVE_FILL	9	用间隔线填
WIDE_DOT_FILL	10	用宽空白点填
CLOSE_DOT_FILL	11	用密空白点填

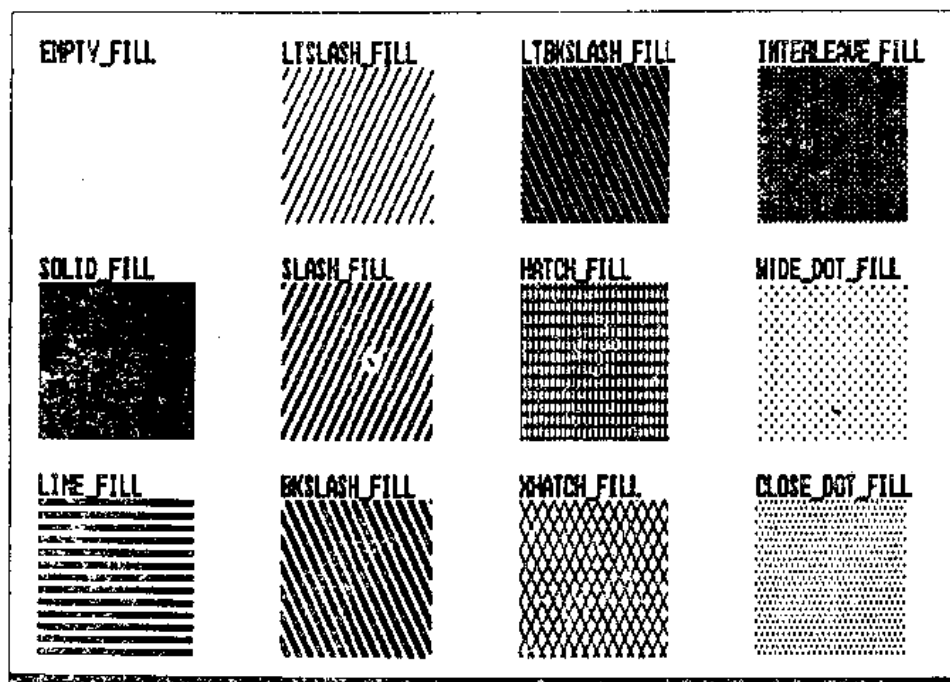


图 3.5 预定义的填充图案

现在，让我们回到填充图案问题上。BGI提供12个预定义的填充图案，表3.6中列出的函数都能使用它们。这些图案枚举在graphics.h并在表3.7中列出。图3.5显示每一这些填充的图案，它们出现在填充的矩形中。除了预定义的填充图案之外，你可以使用USER_FILL以添加你自己的填充图案。我们不久将可看到这点。根据缺省值，所有填充操作都使用SOLID_FILL，而用getmaxcolor()返回的颜色（往往这是白色）涂画内部区域。

3.4.1 设置填充图案

填充图案由setfillstyle() 函数选择。这个例程的函数原型是，


```
void far setfillstyle(int pattern, int color);
```

pattern变元是表3.7中列出的填充类型之一。color变元是画内部的颜色。所有内部中不是图案的部分都用背景颜色涂画。

3.4.2 用户定义的填充图案

不是通过setfillstyle()定义用户定义的填充图案，而是使用函数 setfillpattern()。如下所示，向setfillpattern() 传送8×8二进位的模式，它代表要用的填充图案和绘图的颜色。它的函数原型是：

```
void far setfillpattern(char far *upattern, int color);
```

第一个变元upattern是一个字符数组，它说明在填充操作中使用的图案。它应该是8字节长，其中每一字节代表图案中8个像素的一行。例如，一个实心的填充图案应被定义为：

```
char fillpattern[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
```

类似地，另一个交替设置和清除像素的棋盘图案应说明为：

```
char fillpattern[] = {0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55};
```

为置当前填充图案为这些用户定义的图案之一。你可以使用像下面这样的语句：

```
setfillpattern(fillpattern, getmaxcolor());
```

3.4.3 取填充图案

如同对线型的情况一样，能检索和保存当前的填充设置有时很有用。当你需要临时改变填充设置时，这往往是难能可贵的。存取当前填充设置的函数是：

```
void far getfillsettings(struct fillsettingstype far *fillinfo);
```

正如你能看到的，它使用一个类型结构fillsettingstype返回填充设置。这个结构在graphics.h中定义为

```
struct fillsettingstype {  
    int pattern;  
    int color;  
};
```

对于用户定义的填充图案，检索填充设置稍微有些复杂。如果不使用用户定义的填充图案，你要做的一切就是调用getfillsettings()。而它将返回图案类型和填充颜色。但是，如果当前的填充类型是用户定义的填充图案，则结构中的pattern成分将被置为 USER_FILL。当然，这可用于指出正在使用用户图案，但不能告知是什么样的图案。于是，当需要保存用户定义的图案时，你必须额外调用getfillpattern() 函数以检索用户定义的填充图案。

getfillpattern() 的函数原型是:

```
void far getfillpattern(char far *upattern);
```

它把当前正在使用的用户定义的图案拷贝到传送给它的数组upattern中。你必须确保已为这个数组说明了足够的空间。

总之, 让我们假定, 我们需要保存当前的填充设置 (不管它们是什么), 执行某些操作, 然后恢复这些设置, 为完成这点, 可以使用下面的摘录代码行:

```
char saveuserpattern[8];           // Declare space for the user-  
                                   // defined pattern and the  
struct fillsettingstype savefill;  // fill settings  
getfillsettings(&savefill);        // Retrieve fill settings  
if (savefill.pattern == USER_FILL) // If user-defined fill  
    getfillpattern(saveuserpattern); // pattern, save it too  
// ... code that can change fill settings ...  
if (savefill.pattern == USER_FILL) // Restore fill settings  
    setfillpattern(saveuserpattern, savefill.color);  
else  
    setfillstyle(savefill.pattern, savefill.color);
```

3.4.4 用用户定义的填充图案试验

有许多可能的用户填充图案。您空想象这些图案是比较复杂的事情。所以, 我们将开发一个独立的程序, 它帮助我们交互地探讨不同的填充图案。

这个程序名为userfill.cpp (参见列表3.1), 它有三个主要成分。第一, 有一个画在屏幕右上部分的矩形多边形, 它示出用户定义的填充图案。相邻于它的是一个填充图案的8×8像素区域的分解图。你将能用PC键盘上的箭头键在这个图案上移动光标并选择填充图案中的像素来改动。当你需要设置或清除像素之一时, 仅需简单地配合光标所指之处按一下空白键。这将在你自动改变当前的填充图案, 而且它将更新屏幕上的多边形, 向你展示出新的填充图案。此外, 沿着屏幕的较低部分你将看到一个有效的C++说明。你能在程序中用它来说明你在屏幕上看到的图案。

3.4.5 箭头键

userfill.cpp程序依靠键盘作为用户输入工具。更为特殊地是, 键盘上的箭头键可用于围绕放大的填充图案移动光标。这些键要求专门的处理, 因为每一箭头键生成2字节的序列而不是大多数键盘的键所产生的普通单个字符。结果, 它两次调用getch(), 以便为箭头键取得全键码。于是, 问题变成为, 我们怎能知道在键盘缓冲区中有扩展键码 (必须两次调用getch()) 还是有普通的字符? 幸运的是, 对于我们感兴趣的键, 这不太成问题。原来, 每一个箭头键的头一个字节总是0, 而在键盘上没有别的正规键生成等于0的单个字符。结果, 当我们的程序读到一个0字符时, 它知道在缓冲区中有一个扩展的键码, 而必须从键盘缓冲区再读另一个字节。这第二个字节可用于辨认出按下的是哪种箭头。表3.8示出这些值的表。

在userfill.cpp的main() 中, 根据检查在 while 循环头部getch() 读入0值字符,

swith语句检测扩展的键码，如果这个值为0，它预测按下的是箭头键，并从键盘缓冲区中读出另一个字符。这就是swith语句实际使用的值，它确定按下的是哪一种箭头键 和 应采取什么动作。这些就是在表3.8第三列中列出的值。

表 3.8 箭头键的扩展键码

箭头键	第一字节	第二字节
Home	0x00	0x47
Up	0x00	0x48
PgUp	0x00	0x49
Left	0x00	0x4b
Right	0x00	0x4d
End	0x00	0x4f
Down	0x00	0x50
PgDn	0x00	0x51

•列表3.1 userfill.cpp

```
// userfill.cpp -- This program enables the user to experiment with
// various user-defined fill patterns interactively. The user
// actually manipulates an 8 x 8 enlarged version of the pattern.
// that is drawn on the left side of the screen. On the right
// side of the screen is a rectangle filled with the current fill
// pattern. At the bottom of the screen is the C++ code for an
// array declaration that could be used to generate the current
// fill pattern. The user interaction allowed is:
//     arrow keys on keypad - moves cursor in enlarged fill pattern
//     space bar             - toggles the current pixel under the cursor
//     ESC                   - terminates the program
//
#include <graphics.h>
#include <stdarg.h>
#include <alloc.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

#define PLEFT 20           // Left column of the big pattern
#define PTOP  50           // Top row of the big pattern
#define BIGPIXEL 8         // The big pixels are 8 x 8 in size

// The function prototypes for the functions in this program
void draw_enlarged_pattern(void);
void draw_test_polygon(void);
void show_pattern_code(void);
void init_images(void);
void toggle_bigpixel(int x, int y);
void toggle_pixel(int x, int y);
void toggle_cursor(int x, int y);

// Contains the fill pattern. Initially, it is all set off.
unsigned char fill[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
void *bigpixel;           // Image used to hold a big pixel
```

```

void *cursor; // The cursor image
char bigpatterntitle[] = "User Fill Pattern"; // A few titles
char polytitle[] = "Test Polygon";

main()
{
    char ch;
    int x=0, y=0, gdriver = DETECT, gmode;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    outtextxy(LEFT, PTOP-20, bigpatterntitle); // Write the titles
    outtextxy(400, PTOP-20, polytitle);
    draw_enlarged_pattern(); // Create the enlarged pattern
    draw_test_polygon(); // Draw the test-filled polygon
    show_pattern_code(); // Show the code for the pattern

    while ((ch=getch()) != 0x1b) { // While person doesn't type ESC
        if (ch == ' ') { // If it is a space, then toggle
            toggle_bigpixel(x,y); // the big pixel and update the
            draw_test_polygon(); // polygon and code that shows
            show_pattern_code(); // the pattern
        }
        else if (ch == 0) { // If character was a 0, then it
            ch = getch(); // may be an extended code for
            toggle_cursor(x,y); // an arrow key, get next ch
            // and erase cursor from screen
            // Move the cursor through the big pixel pattern according to the
            // arrow key that is pressed
            switch (ch) {
                case 0x4b : if (x > 0) x--; break; // Left arrow
                case 0x4d : if (x < BIGPIXEL-1) x++; break; // Right arrow
                case 0x48 : if (y > 0) y--; break; // Up arrow
                case 0x50 : if (y < BIGPIXEL-1) y++; break; // Down arrow
                case 0x47 : if (x > 0) x--; // Home key
                           if (y > 0) y--;
                           break;
                case 0x49 : if (x < BIGPIXEL-1) x++; // PGUP key
                           if (y > 0) y--;
                           break;
                case 0x51 : if (x < BIGPIXEL-1) x++; // PGDN key
                           if (y < BIGPIXEL-1) y++;
                           break;
                case 0x4f : if (x > 0) x--; // End key
                           if (y < BIGPIXEL-1) y++;
                           break;
            }
            toggle_cursor(x,y); // Restore cursor to screen
        }
    }
    closegraph(); // Exit graphics mode
    return(0);
}

// Using a series of dotted horizontal and vertical lines, draw an outline
// for the 8 x 8 large pattern on the left side of the screen. When done,
// call init_images() to create the big pixel and cursor images.
void draw_enlarged_pattern(void)
{
    int i, right, bottom;

    setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);

```

```

right = 2 * (PLEFT + (BIGPIXEL-1) * (BIGPIXEL + NORM_WIDTH));
bottom = PTOP + 8 * (BIGPIXEL + NORM_WIDTH);

for (i=0; i<=8; i++) { // Draw outline of big pixels
    line(PLEFT,PTOP+i*(BIGPIXEL+NORM_WIDTH),right,
        PTOP+i*(BIGPIXEL+NORM_WIDTH));
    line(PLEFT+2*(i*(BIGPIXEL+NORM_WIDTH)),PTOP,
        PLEFT+2*(i*(BIGPIXEL+NORM_WIDTH)),bottom);
}
init_images(); // Initialize the big pixel and cursor
}

// This routine draws a rectangular region on the right-hand side
// of the screen using the current user-defined fill pattern.
// Before it displays the pattern it erases the existing rectangle
// on the screen by drawing a BLACK rectangle.
void draw_test_polygon(void)
{
    setlinestyle(SOLID_LINE,0,NORM_WIDTH); // Erase the old filled
    setfillstyle(SOLID_FILL,BLACK); // rectangle
    bar3d(400,PTOP,500,PTOP+50,0,0);
    setfillpattern((char *)fill,getmaxcolor()); // Set to the new
    bar3d(400,PTOP,500,PTOP+50,0,0); // fill pattern and show it
}

// Show C++ code that can be used to declare the fill pattern shown
void show_pattern_code(void)
{
    int x=20, y=150;
    char buffer[80];

    // Erase the old text on the screen by drawing a filled rectangle with
    // BLACK. The rectangle extends across the screen and is 8 pixels high.
    // since the default font is used.
    setcolor(BLACK);
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    setfillstyle(SOLID_FILL,BLACK);
    bar3d(x,y,getmaxx(),y+8,0,0);
    // Convert the values to a string that can be printed out by outtextxy
    sprintf(buffer,"char fill[8] = {0x%02x, 0x%02x, 0x%02x, "
        "0x%02x, 0x%02x, 0x%02x, 0x%02x, 0x%02x};",
        fill[0],fill[1],fill[2],fill[3],fill[4],
        fill[5],fill[6],fill[7],fill[8]);
    setcolor(WHITE); // Restore the drawing color to WHITE
    outtextxy(x,y,buffer); // Write out the C++ code of the pattern
}

// This initialization routine is called once to create the images
// that are used for the big pixel and the cursor.
void init_images(void)
{
    int px, py, i, j;

    // Create the image of a big pixel. Do this at the top-left corner of
    // the big pattern. Once it is created, erase it by exclusive-ORing
    // its own image with itself.
    px = PLEFT; py = PTOP;
    for (j=py+1; j<=py+BIGPIXEL; j++) {
        for (i=px+1; i<=px+2*BIGPIXEL; i++) {
            putpixel(i,j,getmaxcolor());
        }
    }
}

```

```

    }
    bigpixel = malloc(imagesize(px+1,py+1,px+2*BIGPIXEL,py+BIGPIXEL));
    getimage(px+1,py+1,px+2*BIGPIXEL,py+BIGPIXEL,bigpixel);
    putimage(px+1,py+1,bigpixel,XOR_PUT); // Erase the big pixel

    // Next, create a small cursor image where the big pixel just was
    px += 3; py += 3;
    for (j=py; j<=py+BIGPIXEL-5; j++) {
        for (i=px; i<=px+2*BIGPIXEL-5; i++) {
            putpixel(i,j,getmaxcolor());
        }
    }
    cursor = malloc(imagesize(px,py,px+BIGPIXEL*2-5,py+BIGPIXEL-5));
    getimage(px,py,px+2*BIGPIXEL-5,py+BIGPIXEL-5,cursor);
}

// This routine should be called each time a pixel in the user-defined
// pattern is toggled. It will toggle the current big pixel by
// exclusive-ORing the current cell with the bigpixel image. It then
// calls toggle_pixel() to toggle the pixel in the pattern array.
void toggle_bigpixel(int x, int y)
{
    int px, py;

    px = PLEFT + x * 2 * (BIGPIXEL + NORM_WIDTH) + 1;
    py = PTOP + y * (BIGPIXEL + NORM_WIDTH) + 1;
    putimage(px,py,bigpixel,XOR_PUT);
    toggle_pixel(x,y);
}

// Toggle the value for the indicated pixel in the user-defined pattern
void toggle_pixel(int x, int y)
{
    unsigned char mask;

    mask = 0x01;
    mask = mask << (8-(x+1));
    fill[y] ^= mask;
}

// Toggle the cursor image on the screen by using the exclusive-OR
// feature of the putimage command
void toggle_cursor(int x, int y)
{
    int px, py;

    px = PLEFT + x * 2 * (BIGPIXEL + NORM_WIDTH) + 3; // Calculate location
    py = PTOP + y * (BIGPIXEL + NORM_WIDTH) + 3; // of cursor
    putimage(px,py,cursor,XOR_PUT); // Toggle cursor
}

```

3.4.6 喷流填充

迄今为止，所有讨论的填充操作都是围绕各种形状的，有时，你可能需要用特定的填充图案去填写一个用一组线条或对象作边界的区域。为完成这点，你可以使用一个称为喷流填充的操作，在其中你指定一个位置开始填写，然后，填写过程喷流一个区域，直到抵达对象的边界为止。

BGI喷流填充操作的函数原型是，

```
void far floodfill(int x, int y, int border);
```

这里，x和y值说明填充操作的起始位置，通常称为“种子点”，border变元指示颜色，floodfill() 用它确定什么时候它已抵达所填充的区域边界。当填充该区域时，函数使用当前的填充设置。

喷流填充迟早有用之处是当你需要画一填充的圆的时候。这可以用 pieslice()，fillellipse()，或sector() 来完成。但是，我们也可以这样来画一个填充的圆，首先画一个圆，然后调用floodfill() 来填充它。下面的代码执行这一系列的操作：

```
setcolor(WHITE);           // Use WHITE for the circle  
circle(100,100,50);        // Draw a circle  
floodfill(100,100,WHITE);   // Fill the circle starting from its center
```

请注意，圆的颜色必须与floodfill()说明的边界颜色相同，以便此过程能正确地工作。

第四章 BGI字形和正文

本章继续深入考查BGI，探索它的正文处理函数。正如我们曾看过的那样，BGI有若干独特而强有力的绘图和填写功能。最不寻常的是它的字符生成能力。

首先，我们将描述两种BGI支持的字符生成格式：位映像字符和笔画字形。然后，将向你展示怎样在典型的图形程序中利用这两种技术。我们将继续讨论完整的一组例程，它们用于指定字形类型，设置字符的放大率，判断正文的尺寸，定义正文的版面调整等等。

在本章，我们将开发一个有用的增强型正文处理工具的程序库。这些新的函数将执行如自动在边界内封入正文、放大缩小正文以适应已有的窗口，以及支持带屏幕回应的正文输入等任务。

4.1 图形模式的正文

BGI提供两种方式的图形模式向屏幕书写字符。缺省的正文方案用位映像字符，任选而更强有力的方法是使用“笔画”字形。你为应用程序选用的方法将依赖于你需书写的正文大小，所需正文的质量以及要求的字形类型。在下面的几节中，将考查显示正文的这两种方法。我们先从位映像字形开始。

4.1.1 位映像字形

位映像字形自动建立在你用BGI编写的每一个程序中。根据缺省，每当你用BGI正文函数之一把正文输出到屏幕时将显示位映像字符。

缺省位映像字形中的每一字符用 8×8 像素图案表示。其中图案中的每一位对应于一个屏幕像素。如果在图案中的一位是1，则在屏幕上以当前颜色显示对应的像素。如果该位是0，则此像素被置为背景的颜色。由于所有的字符都是以同一的格式存放和显示，位映像字符很容易工作并且能快速显示。图4.1提供一个位映像字符的展示视图。

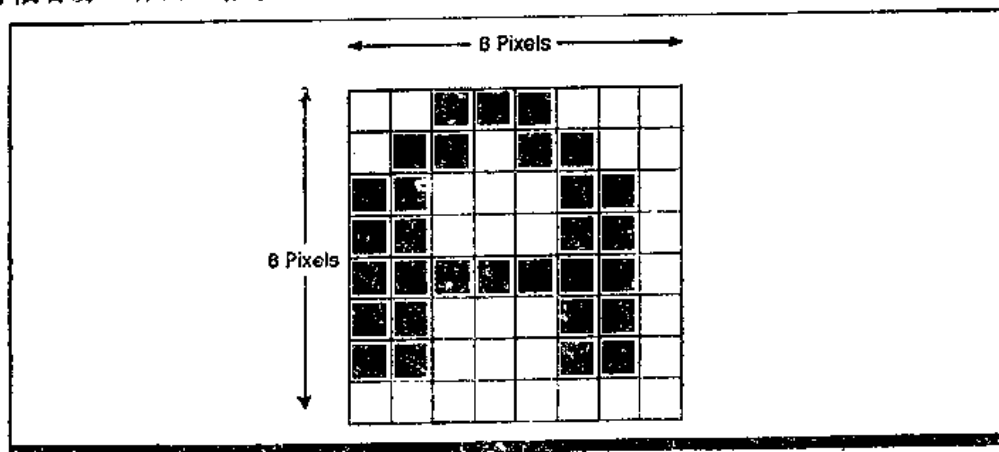


图 4.1 位映像字符的分解图

根据缺省值, 这些位映像图案产生8个像素宽和8个像素高的字符。在稍后的节中, 我们将看到怎样能改变位映像字符的大小和甚至垂直地写它们。

4.1.2 四种笔画字形

虽然位映像字符已适应大多数的应用, 但BGI还提供笔画字形。它们用于显示图形模式的高质量正文输出。笔画字形不是以位模式存放的, 而是每一字符由一系列线段(或“笔画”)来定义(见图4.2), 字符描述的大小和它的复杂性有关。例如, 字符M要比字符T要求更多的笔画。因此, 通常定义带曲线或多线段的字符可能要取很多笔画。

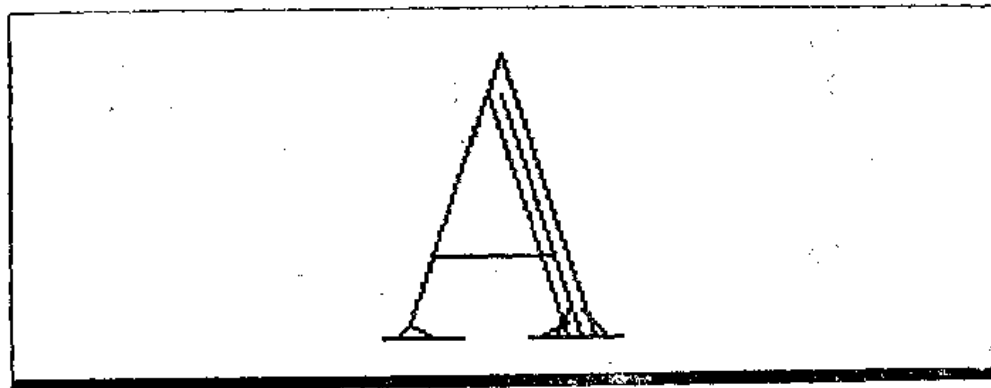


图 4.2 笔画字形字符

与位映像字形相比, 使用笔画字形的主要优点是: 笔画字形可以很容易放大缩小而不会丧失它的高分辨率。BGI包括4种不同的笔画字形, 分别称为小字形、无衬线、三元组和粗黑体。你将会发现, 这些字形对大多数应用提供了足够的灵活性。图4.3给出这些字形的每一样板。Borland还有可利用的定制笔画字形编辑程序, 以便你能建立和编辑自己的字形。在本章的稍后, 我们将仔细考查这个实用程序和讨论怎样构造笔画字形文件。

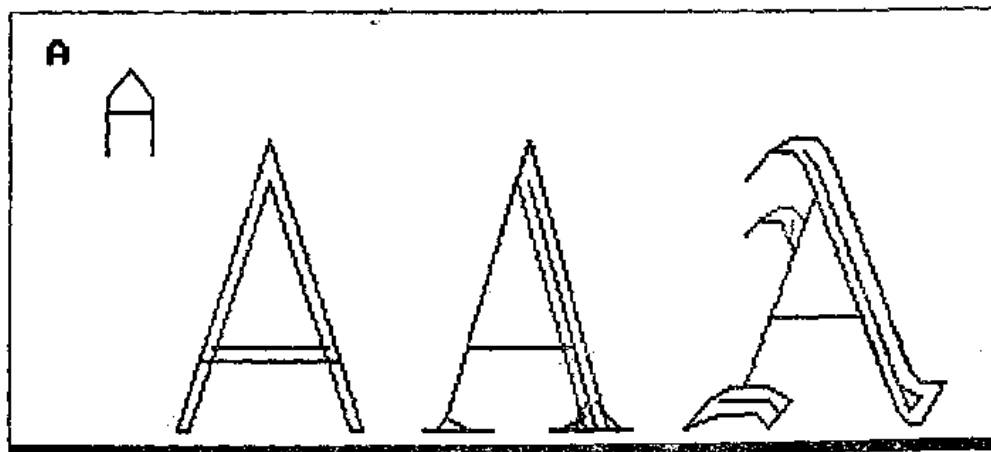


图 4.3 每一BGI字形的字母A

4.1.3 BGI正文函数

除了字形以外, BGI提供9种和正文有关的函数, 它们在表4.1中列出。我们在本章将详细讨论大部分这些函数, 并且将提供使用它们的不同技术。就像你研究该表后可能发现

的那样，正文处理例程为用位映像和笔画字形工作提供了极大的灵活性。请记住，两个函数 `setusercharsize()` 和 `registerbgifont()` 只对笔画字形工作。

表 4.1 BGI中与正文有关的函数

函 数	描 述
<code>gettextsettings()</code>	检索当前字形、方向、大小和正文的版面调整
<code>outtext()</code>	显示当前位置 (CP) 的正文字符串
<code>outtextxy()</code>	显示在位置 (x, y) 的正文字符串
<code>registerbgifont()</code>	用于把字形文件连接到程序的可执行文件中的函数
<code>settextjustify()</code>	定义 <code>outtext()</code> 和 <code>outtextxy()</code> 的对准型
<code>settextstyle()</code>	置字形类型、方向和字符放大率
<code>textheight()</code>	返回按像素计算的字符串高度
<code>textwidth()</code>	返回按像素计算的字符串宽度
<code>setusercharsize()</code>	置笔画字形所用的放大因子

4.1.4 把正文写到屏幕上

由于BGI支持几种不同的视频模式、多种字形类型以及可变大小的正文，它提供它自己的正文函数，可用于显示所有屏幕输出。两个显示正文的函数是 `outtext()` 和 `outtextxy()`。Turbo C++屏幕输出函数如 `printf()` 和 `putch()` 将不能在所有模式中工作，所以在图形模式下应避免使用这些后来的C++库例程。

函数 `outtext()` 在当前观察端口的屏幕位置上显示ASCII字符串。它的函数原型在 `graphics.h`中定义为：

```
void far outtext(char far *textstring);
```

参数 `textstring` 是被显示的字符串。根据缺省值，所有正文都水平地书写，并且相对于当前位置左对齐。当使用缺省设置时，对 `outtext()` 的调用将更新当前位置为正文的最右边。因此，再一次调用 `outtext()` 将把它的字符串直接放在先前显示的字符串的右边。

稍后将会看到，如果你在字形间切换，或使用了不同于先前指定的正文安排，那么，`outtext()` 将不能在每次调用之后自动更新当前位置，你必须自己跟踪它的位置。我们将在有关确定正文的尺寸那一节描述这个过程。

4.1.5 把正文写到像素位置上

`outtext()` 的一伙伴正文输出函数是 `outtextxy()`，像 `outtext()` 一样，它把正文的字符串显示到屏幕上；但是，对 `outtextxy()` 而言，正文被显示在你指定的像素坐标上，`outtextxy()` 函数原型是：

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

前两个变元指定要调准的字符串的像素坐标。像以前那样，`textstring` 是要显示的字符串。当使用 `outtextxy()` 时，请记住，这个函数在已显示正文字符串之后不改变当前

的绘图位置。

4.1.6 一个正文显示的例子

本节所示的程序显示如何用`outtext()`和`outtextxy()`函数显示正文。它从使用Turbo C++的自动探测功能初始化图形模式开始。一旦初始化成功,则在屏幕上显示若干字符串。该程序为所有显示的正文使用缺省的设置。结果,使用的是位映像字符,正文左对齐和水平地写。下面是完整的程序,

```
// texttest.cpp -- Demonstrates the outtext() and outtextxy()
// functions in the BGI
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

main()
{
    int gmode, gdriver = DETECT;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");

    if (gdriver < 0) {
        printf("Graphics initialization failure.\n");
        exit(1);
    }
    // Display text
    outtext("This sentence is printed using outtext ");
    outtext("in the default font. ");
    outtextxy(getmaxx()/2, getmaxy()/2, "outtextxy printed this.");
    outtext("This sentence is also printed using outtext.");
    outtextxy(0, getmaxy()-20, "Press Any Key to Continue ...");
    // Wait for keypress
    getch();
    closegraph();
    return(0);
}
```

前两个正文字符串是用`outtext()`显示的。由于这个程序使用了缺省的版面调整, `outtext()`在每一调用之后更新当前位置。结果,两个字符彼此紧挨着显示。下一个函数调用是`outtextxy()`,它用于在屏幕中部开始显示一字符串。函数`getmaxx()`和`getmaxy()`检索屏幕的宽度,并用于计算屏幕的中点。这是由下面的行完成的:

```
outtextxy(getmaxx()/2, getmaxy()/2, "outtextxy printed this.");
```

下一个是对`outtext()`的调用,由于`outtextxy()`不更新当前的绘图位置,请注意,它的字符串是在`outtext()`印出的上一个字符串而不是在`outtextxy()`印出的字符串之后显示的。最后,该程序调用`outtextxy()`显示屏幕左下角的状态行,通知你按任一键以继续。

4.2 Turbo C++如何存取字形

不像缺省的位映像字符,笔画字形不是自动建立在每一图形程序中的,此外,当运行

你的程序时，通常只有一个笔画字形在存储中立即可用。Turbo C++这样做是为了避免超额使用存储，但增添了若干复杂性。

第一，每一笔画字形被存放在单独的字形文件中，这些文件（列在表4.2）包含在你的Turbo C++盘中，并带有`.chr`扩展名。如果你打算使用笔画字形，则运行时笔画字形文件对你的程序而言必须是可存取的。如果它们不可存取，则将出现错误。此外，如果你使用多于一个笔画字形，则程序必须在每次选择不同的字形类型时访问该笔画文件。

表 4.2 BGI的笔画字形文件

文件名	描 述
<code>goth.chr</code>	笔画的黑体字形
<code>litt.chr</code>	小字符的笔画字形
<code>sans.chr</code>	笔画的无衬线字形
<code>trip.chr</code>	笔画的三元组字形

4.2.1 选择和装入字形

为使用笔画字形之一，你必须显式地装入一个字形文件，这是用`settextstyle()`函数完成的，它的函数原型是：

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

变元`font`是一个数值，它指定要装入的字形文件。为表示字形文件的代码，BGI提供下列一组宏常数，它们在`graphics.h`中定义：

常 数	值
<code>DEFAULT_FONT</code>	0
<code>TRIPLEX_FONT</code>	1
<code>SMALL_FONT</code>	2
<code>SANS_SERIF_FONT</code>	3
<code>GOTHIC_FONT</code>	4

第二个变元`direction`指定正文应水平还是垂直书写。它可被置成两种有效值之一，而且也是由`graphics.h`中定义的宏常数表示的，如：

常 数	值
<code>HORIZ_DIR</code>	0
<code>VERT_DIR</code>	1

最后一个变元`charsize`指定字符的放大因子，它是在用`outtext()`或`outtextxy()`显示正文时使用的。这个变元可以被置为从0到10的任一整数值。我们将简要地讨论`charsize`中的这些不同值的效果。

在离开这个主题之前，让我们看几个如何调用`settextstyle()`的例子。下面的样本语

句装入粗黑体字形文件,定义方向标志以便所有正文水平地书写并置字符的放大因子为4:

```
settextstyle(GOTHIC_FONT,HORIZ_DIR,4);
```

下一个函数调用选择三元组的字形并迫使所有正文垂直地显示,设置字符大小为最大可能的尺寸10:

```
settextstyle(TRIPLEX_FONT,VERT_DIR,10);
```

4.2.2 装入字形时的错误

如果找不到或不能装入适当的字形文件,那么,将初始化错误状态。

在调用settextstyle()之后,你可以调用graphresult()以检查错误。表4.3列出了可能的错误情况。

可以使用下面摘录的代码改变当前使用的字形,如果出现某些错误,则可被查出。如果有问题,该例程调用grapherrormsg()印出错误的原因并使程序终止。请注意,grOK是在graphics.h中定义0值的宏常数,如果在settextstyle()中不出现错误,它将是graphresult()返回的值。

```
void changetextstyle(int font, int direction, int charsize)
{
    int errornum;

    graphresult();
    settextstyle(font,direction,charsize);
    if ((errornum=graphresult()) != grOk) {
        closegraph();
        printf("Graphics Error: %s\n", grapherrormsg(errornum));
        exit(1);
    }
}
```

表 4.3 当装入字形文件时可能的错误

返回值	描述
-8	找不到字形文件
-9	没有足够的存储装入字形
-11	一般的错误
-12	图形输入/输出错
-13	无效的字形文件
-14	无效的字形号

4.3 建立定制的字形

现在,我们已涉及如何使用内部笔画字形的基础。让我们转向考查怎样才能建立我们自己定制的笔画字形。创建一个新字形可能比你想象的要容易得多。尤其是,由于Borland已开发了一个称为FE.EXE的专用笔画字形编辑程序。字形编辑程序允许你使用鼠标交

互地编辑和建立.chr字形文件。你也可以在屏幕上预先试演示一些字符。为运行这个实用程序，你将需要确保你安装有兼容Microsoft的鼠标，并且你的系统配备有EGA或VGA显示硬件。在你通过键入文件名FE调用字形编辑程序之后，将显示如图4.4那样的屏幕。这里，你必须键入已有的笔画字形文件的名字（如sans.chr），或新文件的名字。在键入文

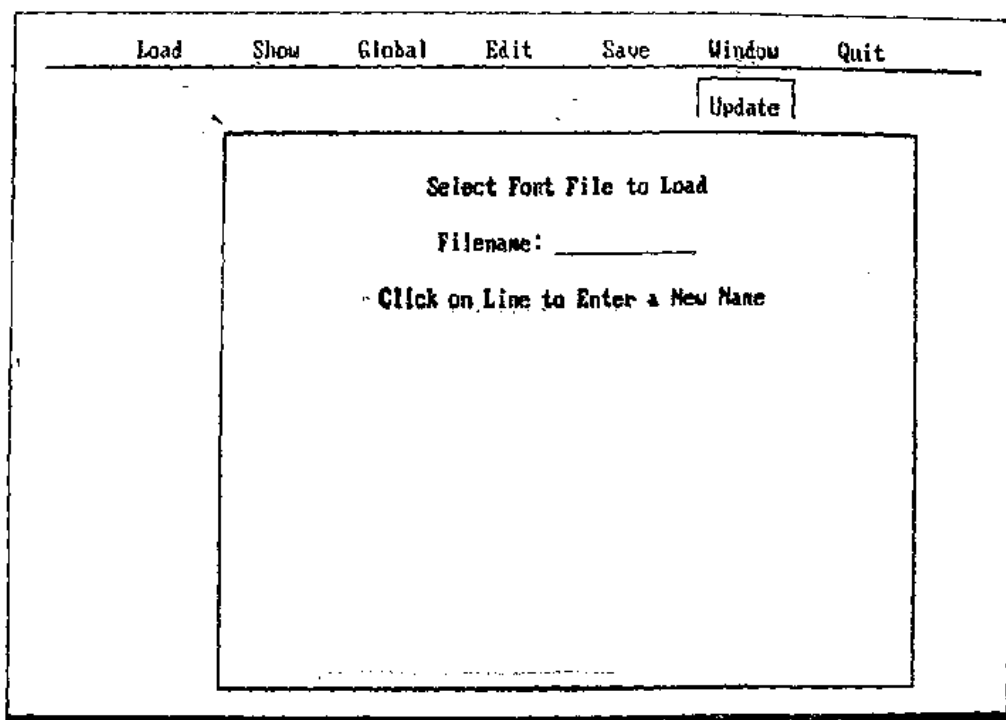


图 4.4 字形编辑程序的开放屏幕

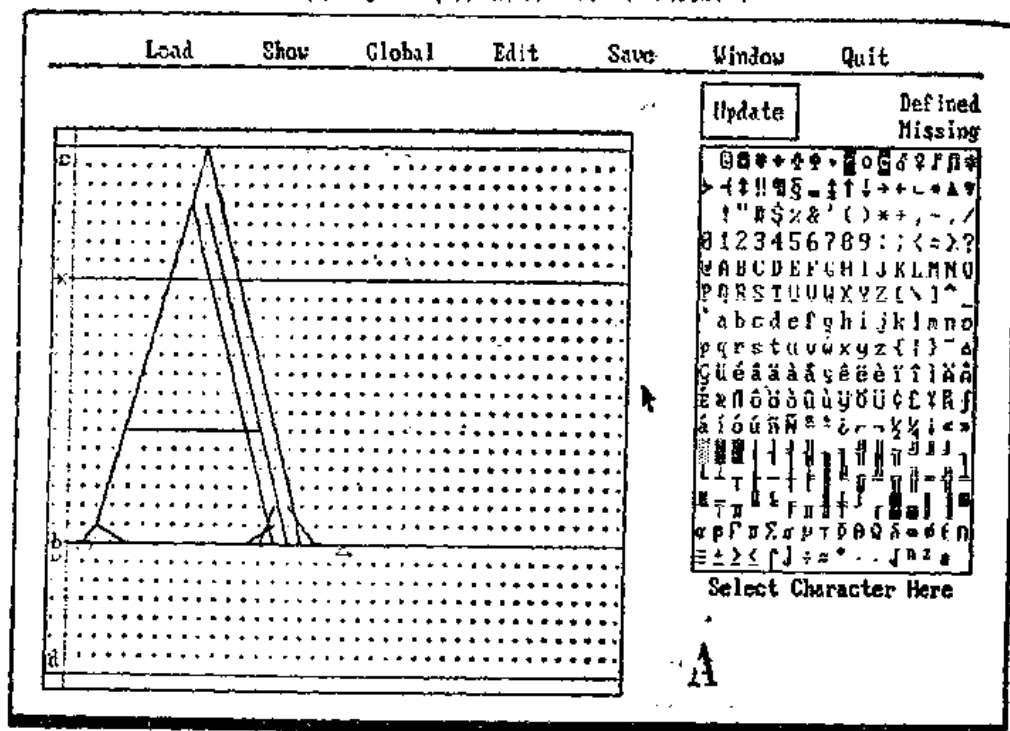


图 4.5 用字形编辑程序编辑一个字形

件名之后，屏幕将改变为图4.5所示的形式。请注意，编辑程序分为三个主要部分：提供基本的编辑和文件管理命令的主菜单条形、字形绘制网格，以及一个为选择已有字形作编辑而使用的字形表。为编辑和创建字符，你只须简单地从表中选出要求的字符，然后用鼠标和绘图网格编辑或建造该字符。为绘制一字符，你必须使用直线段（编辑程序称之为笔画）。编辑程序不支持弧或圆，当在网格中绘制笔画时，请记住，笔画是以放大的格式示出的。你正绘制的字符则以它的实际大小直接在字符选择表下示出。

4.3.1 使用菜单选项

每一个主菜单的选项（Load, Show, Global, Edit, Save, Window, Quit），都可以通过把鼠标光标定位在选项的名字上并按一下右鼠标按钮而选出。某些选项，如Load，仅执行简单的动作，而其它一些选项，将带来一个包含新的一组菜单选项的子菜单。例如，当你选择Edit选项时，菜单条形改变，并显示选项CopyChar, Flip, Shift, ShowAlso, Clipboard和Exit。（请注意，当使用子菜单时，你可以选择Exit选项返回主菜单。）下面是由主菜单选项执行的基本命令一览表：

命令	描 述
Load	装入新的或已有的笔画字形文件
Show	显示所有已经用字形编辑程序创建的字符；你还可以用这个选项绘制你的字符
Global	允许你为整个字符集设置左、右、基线空白；还包括有一个任选项以便从不同的字形文件拷贝字符
Edit	提供一些编辑任选项，包括拷贝、翻转和移位等
Save	把在编辑程序中建立的工作保存到一个.chr笔画字形文件中
Window	提供一组任选项，以控制如何显示字符编辑窗口
Quit	终止该编辑程序

4.3.2 使用绘图网格

建立字符的最富有技巧性的部分是学习如何使用绘图网格。用绘图网格，你可以添加和删除笔画，以及拷贝、翻转和移动字符。在我们深入讨论绘制字符的技术以前，先分析一下绘图网格的成分。

图4.6用它标识的主要成分示出网格。请注意，该网格包含四个代表字符不同高度的水平线。作为原点使用的线称为“基本高度”。这个高度缺省设置为0并代表字符的基础。

某些字符，如字母p（小写），包含“下降斜线”。在这种情况下，下降斜线由构成文字的圆下扩展的柄组成。在绘图网格中，用字母d标识的水平线代表“下降斜线的高度”。直接在“基本高度”之上的是“小写高度”。这条线（由x标记）代表字符集中每一小写字母的高度。最后，再上一条线（用C标记）是大写高度，因为它代表大写字母的高度。

当绘制字符时，应该使用水平高度标记作为你的警界线。例如，如果你想使所有的一组大写字母看起来统一，则应确保使它们从“基本高度”开始并使用“大写高度”作为每

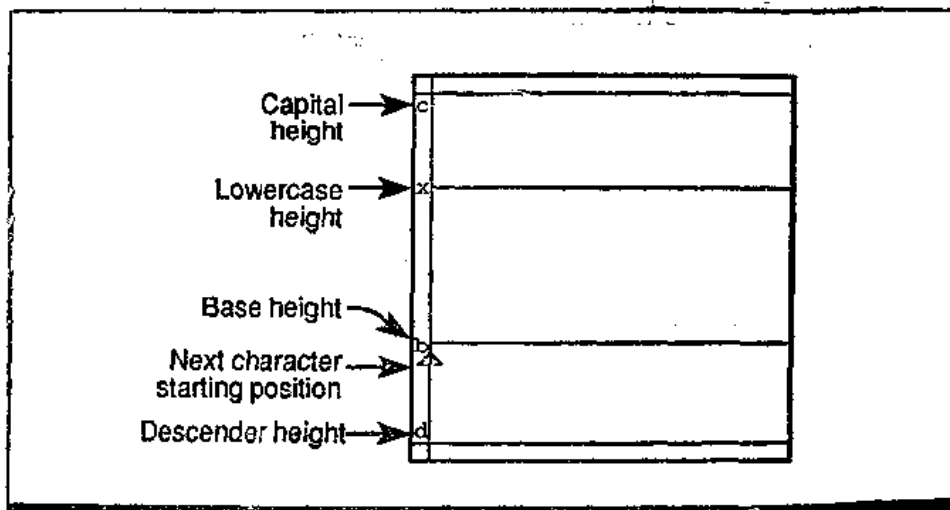


图 4.6 字形编辑程序的字符绘图网格

一字符的顶点。与基本高度交叉的小三角用作标记，以指示下一字符将从什么地方开始。在绘制一个字符之后，你应该确保撤动按钮并把三角拖曳到右边而移动它，以便在绘制的字符或字母表的每一字符之后放置一些空间。

当你启动字形编辑程序时，我们刚才已讨论的4个水平方向被赋予缺省的设置（基本高度 = 0，下降斜线高度 = -7，小写高度 = 20，大写高度 = 40）。由于这里尺寸是用字符M、q和x的基本尺寸确定的，你可以通过调用字形编辑程序绘制出这些字形并保存它们来改变其设置。当下次装入这些字形时，将使用这些新的字符尺寸。

一旦已设置好尺寸，你可以通过撤动按钮和拖曳鼠标画出每一字符的笔画，编辑程序使用“橡皮筋”技术画线。为擦去一条线，只须简单地在线上再画一下就可以了。如果你开始画了一条线，但随后不喜欢它，则把鼠标器的光标移出绘图网格，该线将会消失。

4.3.3 使用正文版面调整

BGI提供少量的正文版面调整设置。它们可用于定制你的图形显示。outtext() 和 outtextxy() 使用这些设置来判断正文该怎样相对于当前绘图位置显示。根据缺省值，正文是左对齐的，并显示在当前位置之上。通过调用settextjustify() 函数可以很容易改变这些缺省设置。它的函数原型是：

```
void far settextjustify(int horiz, int vert);
```

第一个变元horiz设置水平的调准类型，它可被置为下列值之一：

常 数	值
LEFT_TEXT	0
CENTER_TEXT	1
RIGHT_TEXT	2

类似地，变元vert定义垂直的调准，它能用在所有对outtext() 和outtextxy() 的调用中。它的值的范围包括：

常 数	值
BOTTOM_TEXT	0
CENTER_TEXT	1
TOP_TEXT	2

图4.7和图4.8显示一组水平和垂直格式的字符串，使用了调准设置的各种组合。

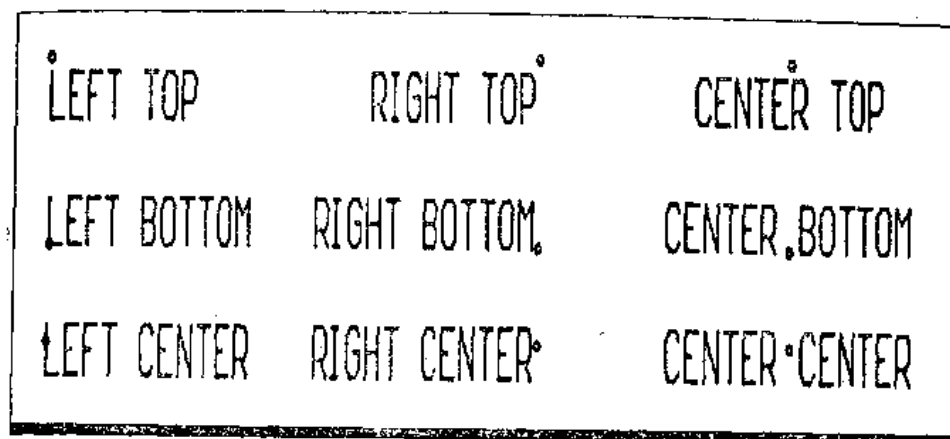


图 4.7 水平正文版面调整

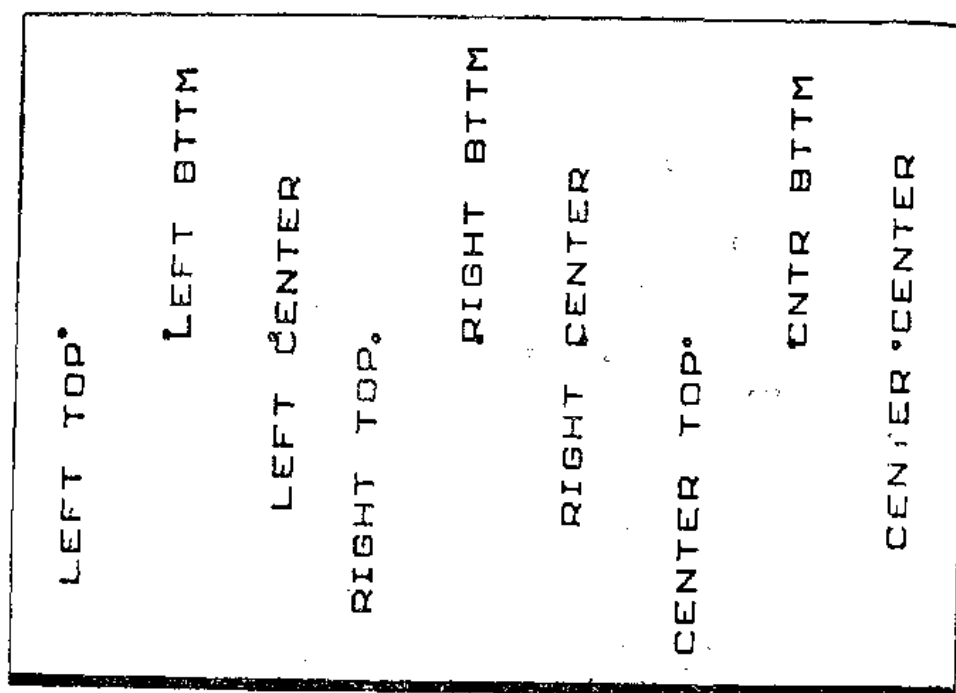


图 4.8 垂直正文版面调整

4.3.4 确定当前的正文设置

Turbo C++提供能检索当前正文设置的函数`gettextsettings()`。在你需要临时改变正文的类型而随后即把它恢复为原来设置的情况下，这个函数很便于使用。`gettextsettings()`的函数原型是：

```
void far gettextsettings(struct textsettingstype far *textinfo);
```

为存放有关正文字形的信息，使用一种命名为textsettingstype的专用数据结构。你将发现，这个结构在文件graphics.h中定义为：

```
struct textsettingstype {
    int font,
    int direction,
    int charsize,
    int horiz,
    int vert,
};
```

成分font包含现用字形类型的数字代码。请记住，Turbo C++提供一个缺省类型和4个笔画字形；于是存放在这个成分中的值的范围是从0到4。第二个成分direction指示当前正文是用水平还是垂直方向显示。沿着表向下看，成分charsize存放用于变比某些被显示正文的放大因子。

比例因子将在稍后详细介绍；简略地说，它的范围是从0到10，其中值1选择位映像字符的标准尺寸（8×8），2指示位映像字符为两倍尺寸（16×16），如此等等。类似地，charsize的较大值建立较大的笔画字形字符，charsize=0只保留给笔画字形，用于选择我们稍后还会看到的字符放大率的补充形式。最后两个成分horiz和vert定义按水平和垂直方向显示正文的调准属性。

如你所看到的那样，调用gettextsettings()将检索字形类型、方向标志、字符放大率以及正文调准设置，并且把这些信息存放在textsettingstype结构。作为一个例子，下面的代码行保存正文字符设置，改变它们，并随后恢复正文的说明。

```
// Declare a structure to save the settings in
struct textsettingstype oldtext;
// Previous code ...
gettextsettings(&oldtext);
// Change the text settings here
settextstyle(oldtext.font,oldtext.direction,oldtext.charsize);
settextjustify(oldtext.horiz,oldtext.vert);
// Rest of the code ...
```

4.3.5 确定字符的尺寸

由于BGI允许字符以不同的尺寸放大，所以有必要确定某些留在屏幕上的正文的实际像素尺寸。当你已试图对齐正文或把正文字符串封入窗口中时，这是特别重要的。BGI提供确定字符串正文尺寸的两个函数。这些函数为：

```
int far textheight(char far *string);
int far textwidth(char far *string);
```

在把字符串显示在屏幕时，它们各自返回字符串的像素的高和宽。这些函数返回的值经常是由字符的面向而不是屏幕的轴确定的。换句话说，textheight()和textwidth()函数返回相同的值，而不管正文是水平还是垂直显示。

位映像字符（请记住，它们是从8×8位模式推导而来的）带有缺省的字符放大率1，有8个像素的正文宽度。类似地，它的高也是8个像素，为理解放大率对有关位映像字符尺寸所起的作用，请参看图4.9。该图以放大率1到10乘缺省的大小示出字母A的尺寸。

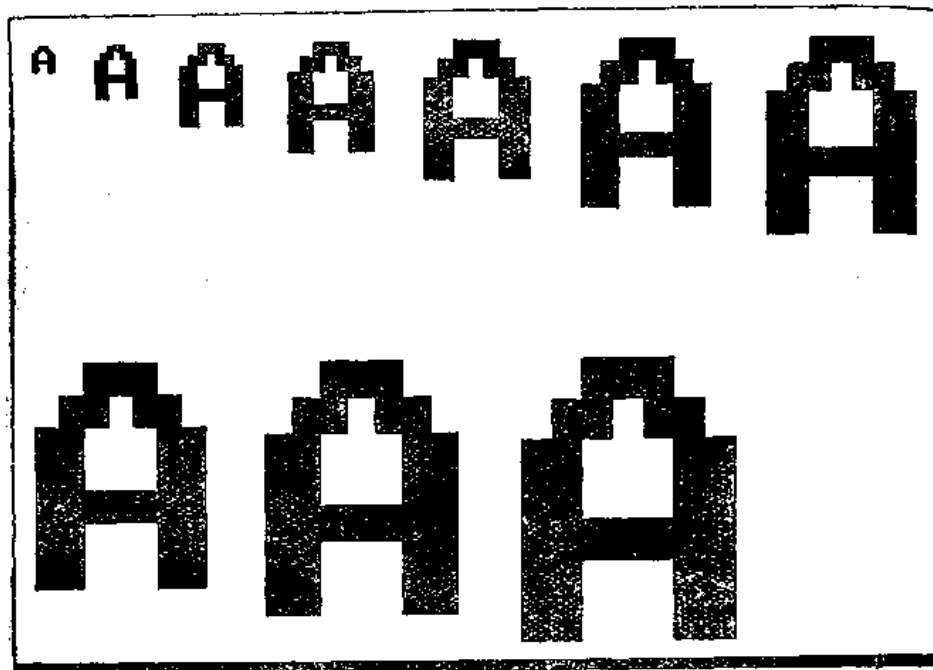


图 4.9 位映像字符的放大率

4.3.6 关于垂直的字符尺寸的注记

虽然`textheight()`和`textwidth()`不管正文是水平还是垂直写而都类似地操作,但在编写代码时它们也可能使你混乱。要记住的是,这些函数返回有关字符串面向的像素尺寸。例如,当水平显示字符串时,`textwidth()`返回正文跨越的像素列的数目。但是,当垂直显示正文时,`textwidth()`返回正文延伸的像素行的数目。当你为设计同等处理水平和垂直正文而编写函数时,需要小心`textwidth()`和`textheight()`。

4.4 放大字符

BGI提供两种方法修改`outtext()`和`outtextxy()`显示的正文大小,所用的方法依赖于正显示的是位映像还是笔画字形。

改变正被显示的正文大小的最简单方法是更改早些时候介绍过的`textsettingstype`结构中的`charsize`字段。可以通过调用函数`settextstyle()`来设置`charsize`成分。请记住,它定义适用于`outtext()`和`outtextxy()`显示的所有正文的比例因子,范围从1到10。如果`charsize`值是在1和10之间,它将影响缺省的位映像字符和笔画字形两者。在位映像字符的情况下,`charsize 1`将产生缺省值大小 8×8 的字符。用`charsize 2`写的位映像正文是缺省值大小的两倍(16×16),如此等等。直到`charsize 10`,它产生 80×80 的尺寸。

`charsize`的较大值还可用作放大笔画字形,虽然它们并不总能生成和位映像字符那样同等大小的字符。图4.10示出每一笔画字形和使用`charsizes 1, 2, 3`和4的缺省字形。

其余的一个`charsize`值0单独保留给笔画字形。如果使用它,则采用通过调用函数`setusercharsize()`指定的第二组比例因子。这使得能较好地控制笔画字形,以便你可以独立地调整正文的高度和宽度。例如,你可以在`setusercharsize()`中使用比例因子,以精确

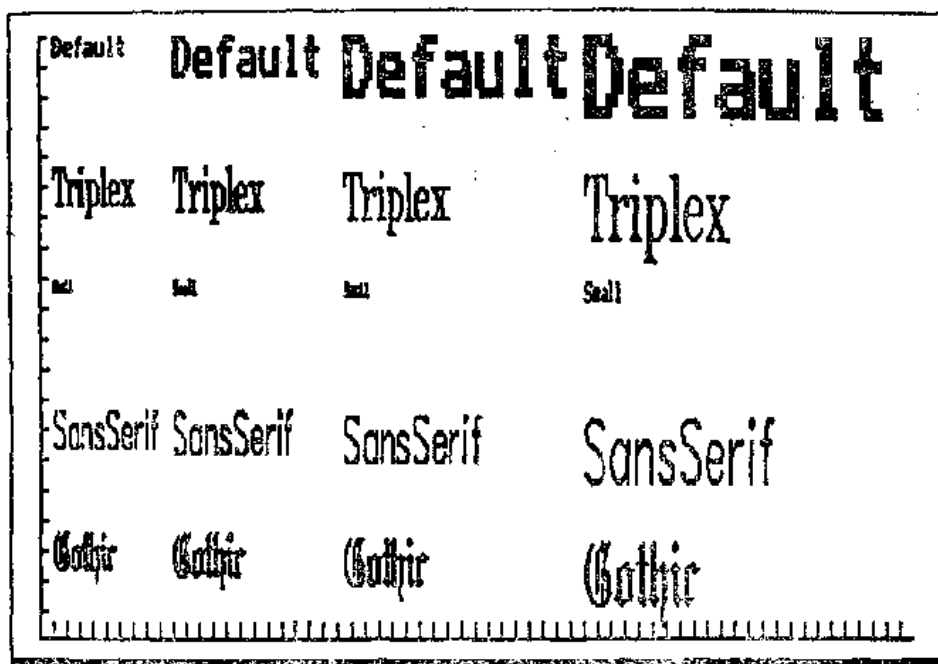


图 4.10 使用charsize 1, 2, 3和4的BGI字形

地把正文字符串送入矩形窗口中。

函数setusercharsize() 原型化为:

```
void far setusercharsize(int multx, int divx, int multy, int divy);
```

其中, 4个变元代表应用于所显示的正文的缺省大小的比例因子 (笔画字形是charsize4)。前两个变元指示延伸或压缩水平方向的正文的总数, 这些变元成对工作并且是定义应用比例因子值的比率。例如, 为加倍所有正文的宽度, multx可以置为2而divx置为1。类似地, 为缩减所有正文的高度为缺省高度的1/3, 我们可以置multy为1和置divy为3。这些数可取任意整数的比率, 直到所使用的图形模式的屏幕宽度为止。因此, 在CGA高分辨率模式 (640×200) 中, 比率multx/divx不能超过639, 而类似地比率multy/divy不能大于199。图4.11示出比例因子如何影响笔画字形字符的比例。

应当小心, 当使用垂直正文时, setusercharsize() 的4个变元是相对于正文自身的而

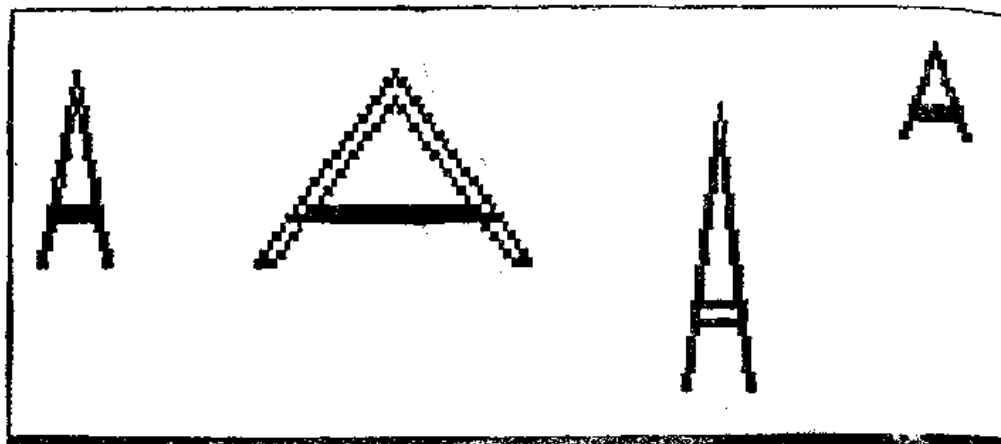


图 4.11 比例因子对笔画字形的影响

不是相对于屏幕的。因此,当正文为垂直时修改multx和divx将改变正文延伸跨越的行数。

4.4.1 把正文放入方框

现在让我们看看怎样能变比笔画字形正文,以便把它合适地送入矩形的区域中。程序autoscal.cpp允许我们指定矩形区域的尺寸和说明放在它中央的字符串。该程序自动变比正文字符串,以便它像图4.12所示那样,完全填入这个区域中。这个程序是值得仔细看看的,因为它使用了setusercharsize()函数。

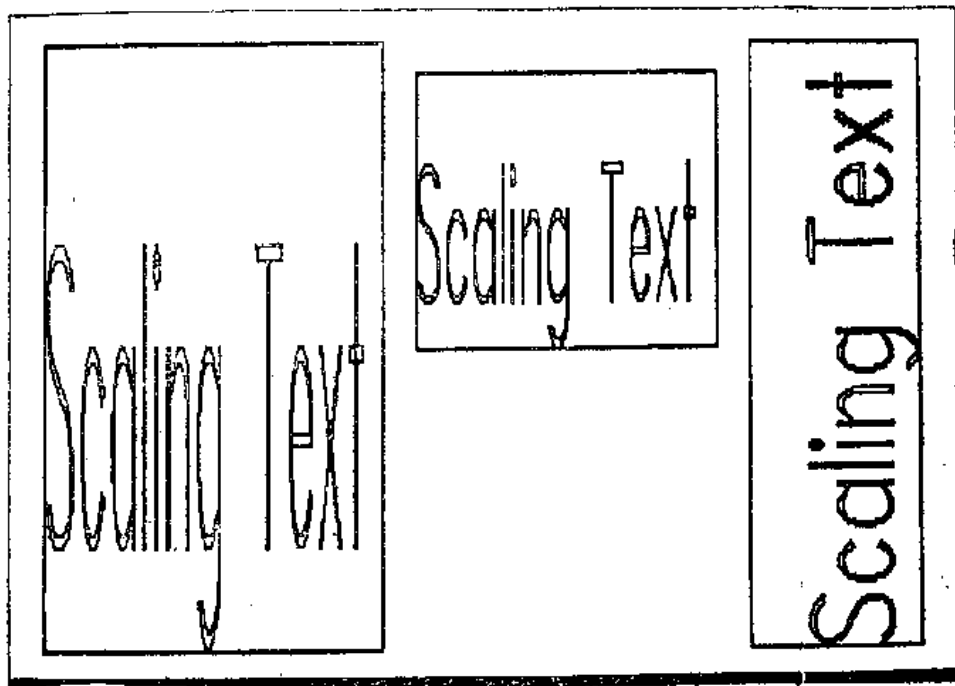


图 4.12 autoscal.cpp的输出

该程序包含两个函数,scaletext()和scaleverttext()。第一个函数缩小或放大正文,以便把它水平地送入它的变元说明的窗口中,后一个函数以同样的方式工作,不同之处在于它把正文垂直地放入窗口。

在每一个函数中,变元left, top, right和bottom指定正文在其中显示的区域边界。要显示的正文被包含在称为string的数组中。两个函数分别按水平和垂直方向把正文放入中央。所以每一个都包含以下语句:

```
settextjustify(CENTER_TEXT, CENTER_TEXT);
```

让我们仔细看看怎样用setusercharsize()变比正文串,以便恰好放入指定的窗口中。这可用包含在每一函数中的下列语句来完成:

```
setusercharsize(1, 1, 1, 1);  
settextstyle(font, HORIZ_DIR, USER_CHAR_SIZE);  
// Note the order of the box coordinates. Compare them with the  
// ordering for vertical text.  
setusercharsize(right-left, textwidth(string),  
                bottom-top, textheight(string)*3/2);
```

你大概首先注意到有两次对setusercharsize()的调用。让我们从第二个开始,如前

所述, `setusercharsize()` 中的变元的比率定义应用于正文字符串的比例因子, 由于我们需要正文延伸跨越该窗口, 我们可以置 `multx` 为 `right-left`, 而置 `divx` 为字符串的长度。实际上, 我们需要确保用缺省的正文放大率, 而不是用当前的设置计算字符串的长度。所以, 这解释了早些时候的如下用法:

```
setusercharsize(1,1,1,1);
```

类似地变比高度以适应该窗口。但是, 请注意, 从 `textheight()` 返回的高度按 $3/2$ 变比。必须使用这个因子以便把字符塞进方框, 使得像 `p`, `g`, `y` 那样的字母 (它们扩展在正文行之下) 不会逸出方框的底部。

```
// autoscal.cpp -- Given a box of a particular size, this program
// scales the text so that it fits within it. This routine
// guarantees that the text string will fit in the box, but it
// doesn't guarantee that the text will look good or that it will be
// readable. This can happen if the box is too small for the text.
#include <graphics.h>
#include <conio.h>

void scaletext(int left, int top, int right, int bottom,
               int font, char *string);
void scaleverttext(int left, int top, int right, int bottom,
                  int font, char *string);

main()
{
    int gmode, gdriver = DETECT;
    char string[] = "Scaling Text";

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    scaletext(0, 0, 200, getmaxy(), SANS_SERIF_FONT, string);
    scaletext(220, 10, 400, 100, SANS_SERIF_FONT, string);
    scaleverttext(420, 0, 520, getmaxy(), SANS_SERIF_FONT, string);
    getch();
    closegraph();
    return(0);
}

// Scale horizontal text to fit in a rectangular region
void scaletext(int left, int top, int right, int bottom,
               int font, char *string)
{
    // Reset usercharsize to all ones, so that the textwidth, textheight,
    // functions will return the number of pixels on the screen that the
    // default font size uses, not the current size, whatever it may be.
    setusercharsize(1,1,1,1);
    settextstyle(font, HORIZ_DIR, USER_CHAR_SIZE);

    // Note the order of the box coordinates. Compare with the ordering
    // for vertical text.
    setusercharsize(right-left, textwidth(string),
                    bottom-top, textheight(string)*3/2);
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    // Clear screen and draw a box where text is to be displayed
    setfillstyle(SOLID_FILL, getbkcolor());
    bar3d(left, top, right, bottom, 0, 0);

    // Write text string centered in the box
```

```

    outtextxy((right + left)/2, (bottom + top)/2, string);
}
// Scale vertical text to fit in a box
void scaleverttext(int left, int top, int right, int bottom,
                  int font, char *string)
{
    int x;
    char buffer [30];
    // ...other code...
    x=10;
    sprintf(buffer, "The value of x is %d", x);
    printf("%s", buffer);
    // ... other code ...
}

```

4.4.2 有关裁剪正文的笔记

虽然BGI支持位映像和笔画字形的裁剪，但这两者处理稍有不同。如果为当前观察端口设置裁剪，那么，当位映像字符扩充的某些部分超出观察端口的界时，位映像字符将完全被裁剪。但是，笔画字形字符则只裁剪超过观察端口的字符部分。

4.5 显示字符和数码

迄今为止，我们只谈到用图形模式把正文字符串写到屏幕中。你可能还想知道数字值和类似的东西怎样在图形模式中显示。是的，`outtext()`和`outtextxy()`只能显示正文字符串，所以，采用的技巧是把你需要显示的所有值转换为ASCII字符串，然后通过调用`outtext()`和`outtextxy()`显示这些字符串。

有许多方法把数字值转换为字符串，但最好的方法可能是通过C++库函数`sprintf()`。简言之，这个函数十分像`printf()`那样工作，除了不把正文写到屏幕，而是把它拷贝到字符串的第一个变元中之外。例如，下面一段代码把存放在整型变量`x`中的值转换为字符串，并印出其结果。

```
int sprintf(char *fmt,...);
```

如果执行这部分代码，字符串“The value of x is 10”将显示在屏幕中。当然，每当需要显示一数值时，都要用`sprintf()`语句点缀你的代码可能比较麻烦。更好的方法是开发一个将按图形模式工作的`printf()`函数。这正是我们在下一节要做的。

4.6 扩展的正文处理例程

虽然BGI正文处理例程具有很强的功能，但它们缺少若干需要添加的以简化图形编程任务的功能。例如，我们需要图形模式的正文输出例程，它具有`printf()`函数的通用性，以及需要它们的配对正文模式那样工作的输入例程。

在下面几小节中，我们将开发一组增强的正文例程，在本书的其余部分当要在屏幕显示正文时将使用它。这些例程由列表4.1和列表4.2分别示出的文件`gtext.h`和`gtext.cpp`支持。

表4.4示出在gtext.cpp包含的函数的表。

表 4.4 gtext.cpp中的函数

函数	描 述
<code>gprintf()</code>	模拟 <code>printf()</code> 的图形打印实用程序
<code>gprintfxy()</code>	显示位置(x, y)上正文的图形 <code>printf()</code> 函数
<code>ggetche()</code>	取得字符并把它回应到屏幕上
<code>gputch()</code>	把字符送到图形屏幕上
<code>ggets()</code>	允许用户在图形模式下键入字符串
<code>gscanf()</code>	在图形模式下模拟 <code>scanf()</code> 函数
<code>gscanfxy()</code>	接受用户在位置(x, y)的输入

4.6.1 printf() 的图形版本

乍看起来, `outtext()` 和 `outtextxy()` 都适合于用图形模式处理正文的应用程序, 但当我们把它们和通用的 `printf()` 比较时, 它们的限制就变得明显了。在本节, 我们将开发一个定制的 `printf()` 函数。它以图形模式工作, 并能支持各种BGI功能。这个函数称为 `gprintf()`, 包含在 `gtext.cpp` 中。

创建模拟 `printf()` 的函数的最大困难是, 该例程应能处理可变数的变元。幸好, Turbo C++ 提供针对这种情况的有用工具, 这些工具包括若干库函数, 它们能自动处理像 `printf()` 中的格式字符串并带有可变数目的变元。由于你可能不善长使用 Turbo C++ 提供的变长变元处理函数, 我们将稍微详细地谈谈我们的新图形 `printf()`。

首先, 当你不知道有多少变元要传送给函数时, 为变长变元编程是必不可少的。函数 `printf()` 就出现这种问题。由于函数 `gprintf()` 是模拟的 `printf()`, 我们必须并入这个特色。在编写定制的 `printf()` 例程时, 我们必须做的第一件事情是告诉编译程序, 该例程期待可变数目的变元, Turbo C++ 有一个特殊的变元类型, 它通知编译程序等待可变数目的变元。这通过在我们的 `gprintf()` 形式参数表中的三圆点序列来表达, 如下所述:

```
int gprintf(char *fmt, ...);
```

比较特殊的是, 这一行告诉编译程序等待一个格式化的字符串, 跟有0个或多个变元。当然, 每一这些附加的变元将对应于格式字符串中说明的变量。Turbo C++ 编译程序有一些内部函数把可变数目的变元和格式字符串相匹配。我们不久将会遇到。

为了在运行时从函数调用中抽出这些变元, Turbo C++ 提供函数 `va_start()` 和 `va_end()`。两个函数都在标题文件 `stdarg.h` 中原型化, 并用于初启和终止从堆栈中抽出变元。函数 `va_start()` 返回指向在堆栈中出现的第一个可变变元的指针, 它把指向这一位置的指针返回到 `argptr` 中。用这些例程, 我们可以抽出格式字符串并找出变元表的头。

如果在编译时知道变元的数目, 我们就可以用 `sprintf()` 函数编写代码(如先前所作的那样), 以使用格式字符串匹配任一变元。但是, 由于我们不知道有多少参数以及它们的类型是什么, 故不能使用 `sprintf()`。幸好, Turbo C++ 提供能接受指向格式字符串的指针(如 `sprintf()`)和指向可变数目变元的表的指针的变异 `sprintf()`。我们可以通

过使用上面提到的可变变元例程抽出这两片信息。因此，可用`vsprintf()`把可变数目的变元和格式字符串(`fmt`)组合为ASCII字符串。它返回成功地格式化的变元数目的计数。最后，使用`va_end()`终止对可变变元的存取。

在我们的`gprintf()`函数中，使用`outtext()`显示格式化的ASCII字符串。因此，将显示正文并相对于当前位置调准。如果强制为左对齐和水平的正文型，则当前位置将在调用`gprintf()`之后自动更新。该函数是，

```
#include <stdarg.h>

int gprintf(char *fmt, ...)
{
    va_list argptr;           // Pointer to argument list
    char str[80];             // Will hold converted list
    int count;                 // Number args printed

    va_start(argptr,fmt);     // Initialize va_ functions
    count = vsprintf(str,fmt,argptr); // Convert to string
    outtext(str);              // Display the string
    va_end(argptr);           // Close va_ functions
    return(count);             // Return conversion count
}
```

4.6.2 为笔画字形清道

有关BGI笔画字形的令人烦恼的事情是在重写已经显示在屏幕上的图形时，它们引起混乱。其原因是笔画字形只管画出一系列线段。它们在显示时不清除字符下面的空间。解决这个问题的一种方法是在显示之前在每一字符出现的地方擦去屏幕。这可以通过先在写字符之前画一填充的矩形区域来完成，这个区域的大小等同于每一字符、颜色为背景颜色。

我们已扩充`gprintf()`函数，以便它能清除画字符串处的屏幕部分。这是包括在`gtext.cpp`源文件中的`gprintf()`版本。如果你考查此代码，将会看到它使用`bar()`函数并配合`SOLID_FILE`设置背景颜色，以清除显示的字符串后面的屏幕。

4.6.3 `gprintfxy()` 函数

`gtext.cpp`源文件还包含一个称为`gprintfxy()`的函数，它类似于`gprintf()`函数。主要的差异是`gprintfxy()`是在(`x`, `y`)坐标位置上写字符串。像BGI的`outtextxy()`，`gprintfxy()`不更新当前的位置。由于该例程类似于`gprintf()`，我们将不再详细讨论它。

4.7 使用正文输入

BGI不直接支持的另一个部分是正文输入。幸好，我们可以很容易建立某些工具以提供正文输入功能。已有一个重要的输入例程是字符输入函数，它把输入回应到图形屏幕。另一个提供的函数是`ggetche()`，它模拟C++库函数`getche()`，但不同的是它把正文显示到图形屏幕。它被设计成只要正文是左对齐的和水平绘制，则更新当前位置。完整的函数是：

```
int ggetche(void)
{
```

```

char ch;

ch = getch();          // Get character with no echo
gprintf("%c",ch);      // Output character
return(ch);
}

```

这段代码是比较简单的。调用`getch()` 将从用户中获得单个的字符输入，但不回应此输入。一旦检索到该字符，就通过调用我们定制的库函数`gprintf()` 显示它。请记住，这个例程将更新当前位置。

你可能需要写一个可供比较的例程，它在一指定位置获得单个字符输入。这个例程可能称为`ggetchxy()`，并可能引用`gprintfxy()` 函数以代替`gprintf()`。

4.7.1 键入字符串

虽然上面的函数解决了图形模式的单个字符输入的问题，但键入一系列字符时仍然有些麻烦。我们将建立的下一个例程称为`ggets()`，它将允许你键入单个字符串并以图形模式回应。

实现中极为重要之点是，在图形模式中，退格字符和若干其它控制字符不再维持它们的原有功能。你可以用`getch()` 读这些字符，但它们不执行某些动作。例如，为删除一个字符，你实际上应该自己处理退格动作。此外，在图形模式中没有光标。

包括在`gtext.cpp`中的例程`ggets()` 解决了这些问题。该函数接受一个缓冲区，在其中将放置键入的全部正文，直到按下回车键为止。当键入正文时，将提供一下划线作为光标，并支持退格字符。在退出时，该函数返回一个指向缓冲区的字符指针。当使用这个函数时，你必须确保传送给`ggets()` 的字符缓冲区有足够的地方以接受整个键入的字符串。

该函数有两个主要的元素。第一个是连续接受由用户键入的某些正文，直到按下回车键为止的while循环。键入的每一个字符都进入传送给该函数的缓冲区。实际上，退格字符是不键入的，代替的是，它使得上一个字符以背景颜色重写，这将有效地擦去上一个字符。在这之后，当前的位置左移到该字符曾驻留的地方。

4.7.2 键入数字值

正像按图形模式显示数字值是个问题一样，键入数字值也如此。`gtext.c`源文件还包含两个称为`gscanf()` 和`gscanfxy()` 的函数。它们提供C++ `scanf()` 函数的基于图形的仿真。两个例程使用早些时候描述的`ggets()`，组合Turbo C++的可变变元，字符串`scanf()` 函数，`vsscanf()`，以支持在图形模式下的用户输入。虽然这些函数处理用户输入，但在许多方面它们类似于`gprintf()` 和`gprintfxy()` 函数，所以我们将不再详细讨论它们。

• 列表4.1 `gtext.h`

```

// gtext.h -- Header file for gtext.cpp
const int BUFSIZE = 140;
const char CR = 13;          // Carriage return
const char BS = 8;           // Backspace
const char ESC = 27;         // Esc key

```

```

int gprintf(char *fmt, ... );
int gprintfxy(int xloc, int yloc, char *fmt, ... );
int ggetche(void);
int gputch(int c);
char *ggets(char *buffer);
int gscanfxy(int locx, int locy, char *fmt, ... );
int gscanf(char *fmt, ... );

```

• 列表4.2 gtext.cpp

```

// gtext.cpp -- A set of text input and output routines for graphics mode .
#include <graphics.h>
#include <stdarg.h>
#include <stdio.h>

#include <string.h>
#include <math.h>
#include <conio.h>
#include "gtext.h"

// A graphics-based printf() function. It will print the output
// at location xloc, yloc. It does not affect the current position.
// Assumes LEFT_TEXT and HORIZ_DIR justification.
int gprintfxy(int xloc, int yloc, char *fmt, ... )
{
    va_list argptr;           // Pointer to argument list
    char str[BUFSIZE];        // Buffer to build string into
    int cnt;                  // Result of sprintf conversion
    struct fillsettingstype oldfill; // Current fill setting
    char userfillpattern[8];  // Current user-fill pattern

    va_start(argptr,fmt);     // Initialize va_ functions
    cnt = vsprintf(str,fmt,argptr); // Prints string to buffer
    if (str[0] == NULL) return(0);

    // Clear the space where the text is to be printed
    getfillsettings(&oldfill);
    if (oldfill.pattern == USER_FILL)
        getfillpattern(userfillpattern);
    setfillstyle(SOLID_FILL, getbkcolor());
    bar(xloc,yloc,xloc+textwidth(str),yloc+textheight("H")*5/4);
    if (oldfill.pattern == USER_FILL)
        setfillpattern(userfillpattern,oldfill.color);
    else
        setfillstyle(oldfill.pattern,oldfill.color);
    outtextxy(xloc,yloc,str);  // Write string to screen
    va_end(argptr);           // Terminate va_ functions
    return(cnt);              // Return the conversion count
}

// A graphics-based printf() function. It will update the current
// position. Assumes LEFT_TEXT and HORIZ_DIR justification.
int gprintf(char *fmt, ... )
{
    va_list argptr;           // Pointer to argument list
    char str[BUFSIZE];        // Buffer to build string into
    int cnt;                  // Result of sprintf conversion
    struct fillsettingstype oldfill; // Current fill setting
    char userfillpattern[8];  // Current user-fill pattern
    int xloc, yloc;           // Current current position

```

```

va_start(argptr,fmt); // Initialize va_ functions
cnt = vsprintf(str,fmt,argptr); // Prints string to buffer
if (str[0] == NULL) return(0);
// Clear the space where the text is to be printed
xloc = getx(); yloc = gety();
getfillsettings(&oldfill);
if (oldfill.pattern == USER_FILL)

    getfillpattern(userfillpattern);
    setfillstyle(SOLID_FILL,getbkcolor());
    bar(xloc,yloc,xloc+textwidth(str),yloc+textheight("H")+5/4);
    if (oldfill.pattern == USER_FILL)
        setfillpattern(userfillpattern,oldfill.color);
    else
        setfillstyle(oldfill.pattern,oldfill.color);
    outtext(str); // Write string to screen
    va_end(argptr); // Terminate va_ functions
    return(cnt); // Return the conversion count
}

```

```

// Graphics-based getch() function
int ggetche(void)
{

```

```

    char ch;

    ch = getch(); // Get character with no echo
    gprintf("%c",ch); // Output character
    return(ch);
}

```

```

// Graphics-based putch() function
int gputch(int c)
{

```

```

    char buffer[2];

    sprintf(buffer,"%c",c);
    gprintf(buffer);
    return(c);
}

```

```

// A graphics based text input routine. Returns the string entered.
// Echos text as it is entered. It supports the backspace character
char *ggets(char *buffer)
{

```

```

    int currloc, maxchars, oldcolor;
    struct viewporttype view;
    char ch, charbuff[3];

    buffer[0] = '\0';
    currloc = 0;
    getviewsettings(&view);
    maxchars = (view.right - getx()) / textwidth("M") - 1;
    if (maxchars <= 0) return(NULL);
    gprintfxy(getx(),gety(),"_");
    while ((ch=getch()) != CR)
        if (ch == BS) {
            if (currloc > 0) {
                currloc--;
                if (currloc <= maxchars) {
                    oldcolor = getcolor();
                    setcolor(getbkcolor());

```

```

        sprintf(charbuff,"%c",buffer[currloc]);
        gprintfxy(getx()-textwidth(charbuff),gety(),
            "%c_",buffer[currloc]);
        setcolor(oldcolor);
        moveto(getx()-textwidth(charbuff),gety());
    }
}
}
else {
    if (currloc < maxchars) {
        oldcolor = getcolor();
        setcolor(getbkcolor());
        gprintfxy(getx(),gety(),"_");
        setcolor(oldcolor);
        buffer[currloc] = ch;
        gputch(ch);
        currloc++;
    }
    else
        putch(0x07);
}
if (currloc < maxchars)
    gprintfxy(getx(),gety(),"_");
}
if (currloc <= maxchars) {
    oldcolor = getcolor();
    setcolor(getbkcolor());
    gprintfxy(getx(),gety(),"_");
    setcolor(oldcolor);
}
buffer[currloc] = '\0';
return(buffer);
}

// A graphics-based scanf() function. It updates the current position.
// Assumes LEFT_TEXT and HORIZ_DIR justification.
int gscanf(char *fmt, ... )
{
    va_list argptr;           // Argument list pointer
    char str[BUFSIZE];        // Buffer to accept user input
    int cnt;                  // Number of fields converted

    va_start(argptr,fmt);     // Initialize va_ functions
    ggets(str);               // Get the user input
    cnt = vsscanf(str,fmt,argptr); // Convert the input according
                                // to the format string
    va_end(argptr);           // Terminate the va_ functions
    return(cnt);              // Return the # of successful
                                // input conversions
}

// A graphics-based scanf(). It does not affect the current
// position and will echo input as it is entered at (xloc, yloc).
// Assumes LEFT_TEXT and HORIZ_DIR justification.
int gscanfxy(int xloc, int yloc, char *fmt, ... )
{
    va_list argptr;           // Pointer to argument list
    char str[BUFSIZE];        // Buffer to accept user input
    int cnt;                  // Number of conversions made
    int oldx, oldy;           // Original current position

```

```

    oldx = xloc;           // Save current position so that
    oldy = yloc;           // it can later be restored
    moveto(xloc,yloc);     // Move to the new current position
    va_start(argptr,fmt);  // Initialize the va_ functions
    ggets(str);            // Get the user's input
    cnt = vsscanf(str,fmt,argptr); // Convert input string
    va_end(argptr);        // Terminate va_ function
    moveto(oldx,oldy);     // Restore current position
    return(cnt);           // Return conversion count
}

```

第五章 表 示 图

在微机发展的早期,大多数微机用户都发现建立有用和吸引人的表示图是麻烦的。现在,随着带有高分辨率的视频卡和显示器的快速PC设备的出现,“表示图”的领域已形成。表示图是指用计算机的框图、图形和图画表示复杂信息的技术。

在本章,我们将向你说明如何使用Turbo C++的图形功能产生高质量的商业和科学框图与图形。特别是,我们将探索绘制饼图、条形图、楔形图和动画的技术。我们不可能涉及所有类型的用BGI工具产生的图形和框图,但在本章末尾,你将充分了解怎样用表示图定制你自己的C++应用。

5.1 基本的图形类型

BGI提供三种用以产生图形和框图的专用函数:饼图例程,条形图例程和三维条形图函数。用这些高级函数,可以不必费多少力气就能创建商务的可视图。在本章,我们将详细探索每一个这些函数。并用BGI中的其它一些工具生成其它类型的框图和图形,包括楔形图和动画框图。

5.1.1 饼图

如果你注意某些杂志或报纸,就可能会发现饼图。事实上,你还将在流行的软件应用程序(如Borland的Quattro)中发现它们。在本节,我们将开发一个实用程序,它能自动建立完整的饼图。我们所要做的一切是向程序提供饼片的数据和标题。该程序自动确定饼图的大小以装入屏幕、建立插图并书写标题。这个程序(在列表5.1中示出)是用Turbo C++的pieslice()函数和一组我们自己定制的例程设计的。

5.1.1.1 画饼片

在第三章,介绍了pieslice()函数,我们将用它来画饼图中的每一片。我们的任务是确定如何把一组数据分配给饼图中的各片。典型地,每一片表示一特定数据组相对于整个图的比例。换句话说,每一片代表整个饼的百分率。

对于我们的程序,将假定有一系列百分比值可供使用。每一百分比值对应于一个饼片,而所有的片总和为100%。在某些应用中使用一组原始的数据,程序必须根据它们计算出适当的百分比值。为使程序简单,我们将假定这些百分率已是可用的。

我们的pieslice()函数的增强版本称为pieceofpie()。我们将对需要画的图的每一饼片引用这定制的函数。这个函数的简化版本的代码是:

```
void pieceofpie(int x, int y, int radius, double slicepercentage,
               int color, int fill)
```

```

int startangle, endangle;
static double startpercentage = 0.0;

setfillstyle(fill,color);
startangle = startpercentage / 100.0 * 360.0;
endangle = (startpercentage+slicepercentage) / 100.0 * 360.0;
pieslice(x,y,startangle,endangle,radius);
startpercentage += slicepercentage;
}

```

piecofpie()的前三个变元定义要画的饼片的中心点和半径。从这个代码,你可以看到这些值被简单地传送给Turbo C++ pieslice()函数。但是,我们仍然需要确定饼片的总的大小,这依赖于slicepercentage中的值。让我们首先看看如何确定起始角以开始画当前的饼片,这就是static double startpercentage的目的。它保留已经显示的圆的百分比的运行总数。该说明把它的初始化为0,如下所示:

```
static double startpercentage = 0.0;
```

每次调用piecofpie()时,就把slicepercentage加到这行总数中。然后从这个位置开始画下一个饼片。因此, startpercentage指定画饼片的开始角,而slicepercentage指示例程把片做成多大。实际上,这些百分比值必须在使用之前转换。Turbo C++ pieslice()函数等待向它传送角度,而不是百分率。因此,我们必须把这些值转换为角度。这通过两个语句来完成:

```

startangle = startpercentage / 100.0 * 360.0;
endangle = (startpercentage+slicepercentage) / 100.0 * 360.0;

```

两个语句是类似的,每一都基于以下比率:

$$\frac{\text{要求的角}}{360\text{度}} = \frac{\text{百分率}}{100\%}$$

在函数中要注意的唯一别的事情是,它用传送给它的最后两个变元指定的值设置当前填充类型和颜色。设置这些绘图参数是为了使得各自不同地画饼片。

这样,我们的函数piecofpie()已准备好被调用以建立饼图。但是,让我们稍微修改一下它,以便能在每一饼片的旁边为各百分比值加标签。

5.1.1.2 为饼片写标签

有许多种不同的方法为饼图加标签。我们将为每一饼片把百分比值写在片的界外。这种技术使得观察者容易确定饼片的大小。某些其它可能是把题目写在饼片之下,或甚至把题目和百分比值写在饼片自身中。列表5.1包括这个函数的修改列表和我们的饼图程序的其余源代码。

当考察piecofpie()函数时,第一件要注意的事情是它包含一个称为lradius的新变量。这个变量用于确定饼图中心到写标签处的距离。对于我们的应用而言,用以下语句把lradius设置为1.2倍的饼图半径。

```
lradius = (double)radius * 1.2; // Varies with character size
```


这个计算依赖于你在屏幕上有多少地方留给标签和正使用的正文大小，你可以为此推导一个一般公式，但为简单起见，我们仅乘以常数1.2。

根据约定，每一标签都放在饼片的中心。它的角度存放在labelangle，并且是在饼片的起始角和它的终止角的半途中。该值通过直接插入函数torad()转化为弧度。

```
labelangle = (startangle + endangle) / 2;  
labelloc = torad(labelangle);
```

再者，写标签处的x和y坐标是由以下几行代码计算的。（请注意：y值必须针对屏幕的长宽比作调整。）

```
x += cos(labelloc) * lradius;  
y -= sin(labelloc) * lradius * aspectratio;
```

下面的语句设置正文参数。首先，字形被置为三笔画字形。往下你将发现一系列嵌套的if-else语句，它们确定使用哪种类型的正文版面调整。例如，如果标签是印在饼片的右边，则它的调准被置为LEFT_TEXT和CENTER_TEXT。但是，如果正文写在饼圆的左边，则它的调准被置为RIGHT_TEXT和CENTER_TEXT。最后，用sprintf()格式化正文，并用outtextxy()把它写在屏幕上。

```
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);  
if (labelangle >= 300 || labelangle < 60)  
    settextjustify(LEFT_TEXT,CENTER_TEXT);  
else if (labelangle >= 60 && labelangle < 120)  
    settextjustify(CENTER_TEXT,BOTTOM_TEXT);  
else if (labelangle >= 120 && labelangle < 240)  
    settextjustify(RIGHT_TEXT,CENTER_TEXT);  
else  
    settextjustify(CENTER_TEXT,TOP_TEXT);  
sprintf(buffer,"%3.1lf%%",showpercentage);  
outtextxy(x,y,buffer);
```

现在，我们已经看到如何在饼图中画每一饼片。为画完整的饼图，我们仅需要用适当的值重复调用pieceofpie()函数。

5.1.1.3 使每一饼片不同

可能需要我们去做的一件事是以不同颜色画每一饼片，或至少是用不同的填充图案填写它。为完成这点，我们将顺序通过当前图形模式提供的颜色和11个填充图案。（我们不使用空的填充图案，因为它在CGA高分辨率模式中不生成唯一的填充图案。）

列表5.1中的程序使用函数nextcolorandfill()以顺序通过每一填充图案和颜色组合。它使用初始设置为填充图案1和填写颜色0的静态变量。如果填充图案和颜色的所有组合用尽，则书写函数以便顺序退出。你可能需要改变这点，以便它再次循环通过这些值。

5.1.1.4 建立插图

在pieceofpie()的末尾调用showkey()函数在屏幕的左边用颜色键写出一项。这个插图示出用在每一饼图和它相应标签的每一颜色和填充图案。函数showkey()为每一饼片绘画一个矩形，从屏幕的左上角开始往下画。以深度为0值使用bar3d()函数画填充的区域。方框为16个像素宽16像素高并乘以长宽比。这种修正是为了使区域成正方形。正方形的位置

由静态变量x和y保持。x值总是等于3。但是，每次调用函数时，y位置增加18个像素。

5.1.1.5 饼图程序

图5.1表示列表5.1所提供的程序显示。请注意，该程序显示出在饼图上典型地出现的所有对象，包括饼图的标题、插图和标签。主程序执行初始化BGI的工作，并绘制饼图的基本成分。实际工作函数是pieceofpie()，它负责画每一个饼片。

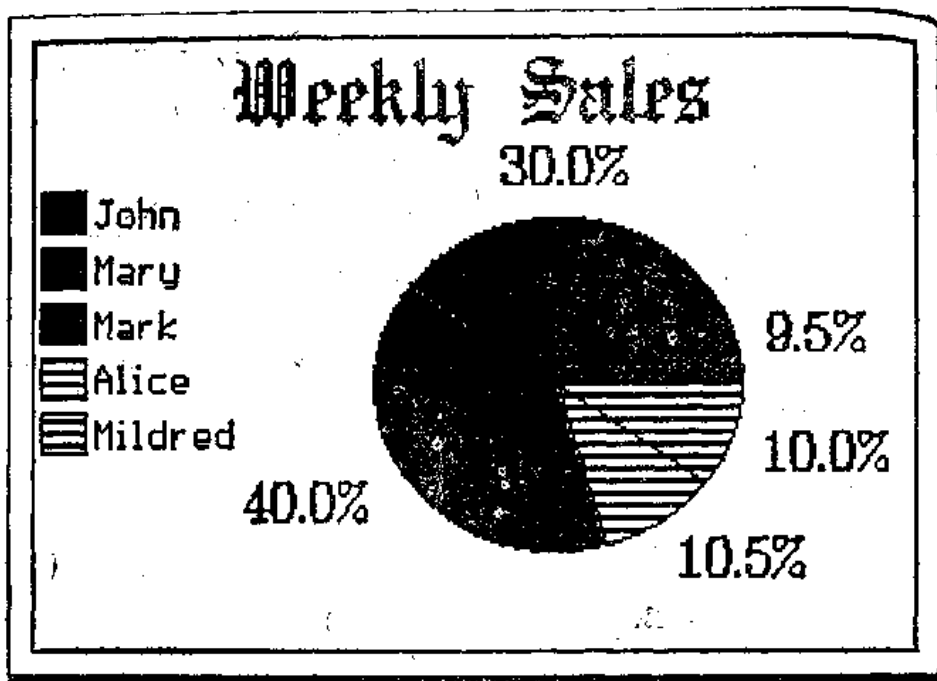


图 5.1 饼图程序的输出

虽然pie.cpp是我们迄今为止所编写过的最复杂的程序之一，但你将会发现，该程序并不太难理解。我们将介绍两个新的C++特色：“直接插入函数”和“按参引传送变量”。

如果仔细观察nextcolorandfill()的定义，你将注意到，变元color和fill都是用&符号说明的，

```
void nextcolorandfill(int& color, int& fill);
```

这种技术告诉编译程序变量是按参引传送而不按值传送的。如果这个代码用Turbo C而不是用C++编写，则我们要按以下语法定义函数：

```
void nextcolorandfill(int *color, int *fill);
```

而且参数应如下传送：

```
nextcolorandfill(&color, &fill);
```

用这种方法的问题是，它易于忘记地址操作符(&)，或在存取该变量的值时忘记用*符号参引该指针变量。两种忽略都会实际使你的代码运行混乱。例如，再看一下列表5.1中的代码，你将看到是用

```
nextcolorandfill(color, fill);
```

调用nextcolorandfill(), 而变元color和fill则在函数体内用以下简单语法赋值,

```
fill = ffill;
color = fcolor;
```

• 列表5.1 pie.cpp

```
// pie.cpp -- Demonstrates the pie chart capabilities of the BGI
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double percentvalues[] = {9.5, 30, 40, 10.5, 10};
char *labels[] = {"John", "Mary", "Mark", "Alice", "Mildred"};
char title[] = "Weekly Sales";
int numvalues = 5;
double aspectratio;

// The function prototypes for this program
void pieceofpie(int x, int y, int radius, double showpercentage,
               int color, int fill, char *label);
void showkey(int color, int fill, char *label);
void nextcolorandfill(int& color, int& fill);

main()
{
    int gdriver = DETECT, gmode;
    int i, x, y, radius, color, fill, xasp, yasp;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    rectangle(0, 0, getmaxx(), getmaxy());
    // Write a header to the graph
    settextjustify(LEFT_TEXT, TOP_TEXT);
    settextstyle(GOTHIC_FONT, HORIZ_DIR, 4);
    outtextxy((getmaxx() - textwidth(title)) / 2, 0, title);

    x = getmaxx() / 2;    y = getmaxy() / 2;
    getaspectratio(&xasp, &yasp);
    aspectratio = (double)xasp / (double)yasp;
    radius = (double)y * 5.0 / 9.0 / aspectratio;
    y += textheight(title) / 2;
    x *= 6.0 / 5.0;
    nextcolorandfill(color, fill);
    for (i = 0; i < numvalues; i++) {
        pieceofpie(x, y, radius, percentvalues[i], color, fill, labels[i]);
        nextcolorandfill(color, fill);
    }
    getch();
    closegraph();
    return(0);
}

// Inline function to convert degrees to radians
inline double torad(int d)
{
    return((double)(d) * 3.14159) / 180.0;
}
```

```
void pieceofpie(int x, int y, int radius, double showpercentage,
               int color, int fill, char *label)
{
```

```
    int startangle, endangle, labelangle;
    double labelloc, lradius;
    static double startpercentage = 0.0;
    char buffer[40];

    // Offset label radius by a factor of 1.2. This value
    // is dependent on the size of characters you are using.
    lradius = (double)radius * 1.2;
    setfillstyle(fill,color);
    startangle = startpercentage / 100.0 * 360.0;
    endangle = (startpercentage + showpercentage) / 100.0 * 360.0;
    labelangle = (startangle + endangle) / 2;
    labelloc = torad(labelangle);
    pieslice(x,y,startangle,endangle,radius);
    x += cos(labelloc) * lradius;
    y -= sin(labelloc) * lradius * aspectratio;
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
```

```
    // Set text justification depending upon location of pieslice
    if (labelangle >= 300 || labelangle < 60)
        settextjustify(LEFT_TEXT,CENTER_TEXT);
    else if (labelangle >= 60 && labelangle < 120)
        settextjustify(CENTER_TEXT,BOTTOM_TEXT);
    else if (labelangle >= 120 && labelangle < 240)
        settextjustify(RIGHT_TEXT,CENTER_TEXT);
    else
        settextjustify(CENTER_TEXT,TOP_TEXT);
    sprintf(buffer,"%3.1f%%",showpercentage);
    outtextxy(x,y,buffer);
    startpercentage += showpercentage;
    showkey(color,fill,label);
}
```

```
void showkey(int color, int fill, char *label)
{
```

```
    static int x = 3;
    static int y = 50;

    setfillstyle(fill,color);
    bar3d(x,y,x+16,y+16*aspectratio,0,0);
    settextjustify(LEFT_TEXT,CENTER_TEXT);
    settextstyle(SMALL_FONT,HORIZ_DIR,5);
    outtextxy(x+20,y+8*aspectratio,label);
    y += 18;
}
```

```
void nextcolorandfill(int& color, int& fill)
{
```

```
    static int ffill = 1;
    static int fcolor = 0;

    fcolor++;
    if (fcolor > getmaxcolor()) {
        fcolor = 1;
        ffill++;
        if (ffill > 11) {
            closegraph();
            printf("Too many calls to nextcolor ...");
            exit(1);
        }
    }
}
```

```

    }
    fill = ffill;
    color = fcolor;
}

```

5.1.1.6 强调一个饼片

在某些饼图表示中，我可能需要强调一个或多个饼片。有几种方法完成这点。一个方法是如图5.2所示，把该饼片从饼图的其余部分偏移一个小量程。虽然我们不想把这个特色添加到前面的例子中，但将提供一个实现它的函数。

为偏移饼片，我们将需要把它从饼图的其余部分移出，即沿着圆形的半径扩展它，例

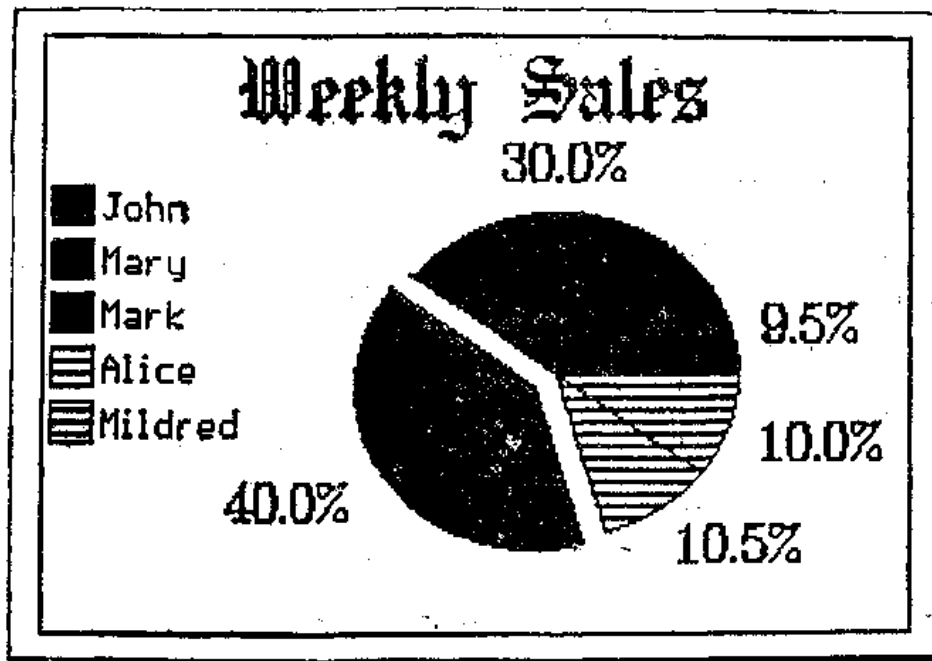


图 5.2 为了强调可偏移一饼片

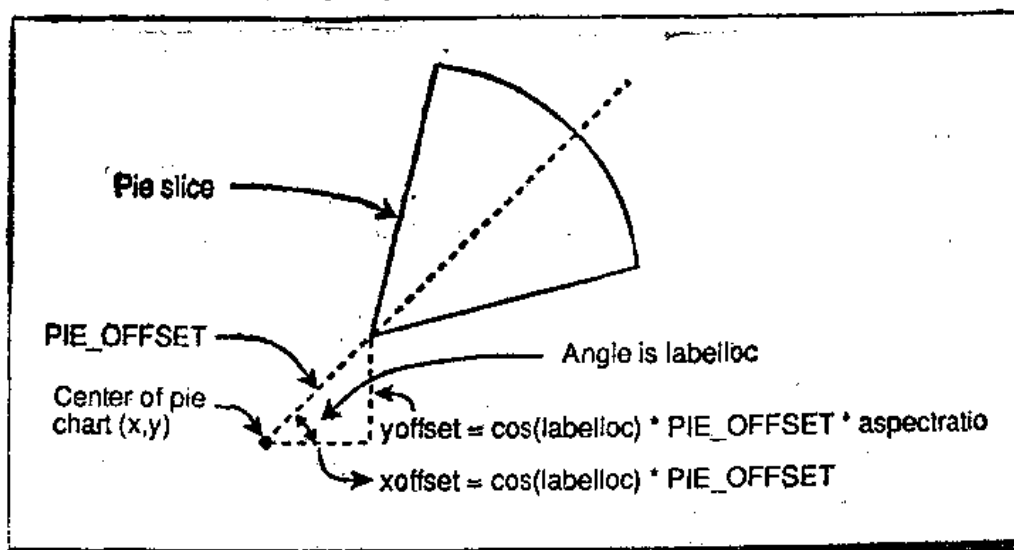


图 5.3 偏移饼片的计算

如,让我们通过乘它的半径以偏移饼片。这个偏移必须加到标志饼片和它的标签的“中心”的x和y坐标上。必须加到x和y位置的数量与饼片画在什么地方有关。还有,我们需要考虑屏幕的长宽比。图5.3示出移动饼片的比例和偏移计算。

下面提供生成偏移饼片的函数,它称为pieceofpie2(),类似于前面的pieceofpie()。主要的差异是pieceofpie2()包含附加的变元offsetflag。当这个变元置为非0值时,该函数绘制一个离开饼图其余部分的饼片偏移,如下面语句所示。首先,偏移的数量如下计算:

```
xoffset = cos(labelloc) * PIE_OFFSET;  
yoffset = sin(labelloc) * PIE_OFFSET * aspectratio;
```

其中,PIE_OFFSET是置为值10的常数。它定义将使用多大的偏移。请注意,这些语句类似于用以计算标签位置的语句。调用pieslice()必须修改以及反映出这个偏移:

```
pieslice(x+xoffset,y-yoffset,startangle,endangle,radius);
```

此外,调用outtextxy()以书写正文标签也必须修改为使用偏移值,如:

```
outtextxy(x+xoffset,y-yoffset,buffer);
```

下面列出完整的函数,你可以像上面的例程那样使用这个函数。除了当需要偏移特定饼片时,你要做的一切是置偏移标志为非0值,否则,饼图按正规绘制。

```
const double PIE_OFFSET = 10.0;    // Amount of offset  
  
void pieceofpie2(int x, int y, int radius, double showpercentage,  
                int color, int fill, char *label, int offsetflag)  
{  
    int startangle, endangle, labelangle;  
    double labelloc, lradius;  
    char buffer[40];  
    int xoffset, yoffset;  
    static double startpercentage = 0.0;  
  
    lradius = (double)radius * 1.2;    // Dependent on size of chars  
    setfillstyle(fill,color);  
    startangle = startpercentage / 100.0 * 360.0;  
    endangle = (startpercentage + showpercentage) / 100.0 * 360.0;  
    labelangle = (startangle + endangle) / 2;  
    labelloc = torad(labelangle);  
    if (offsetflag) {  
        xoffset = cos(labelloc) * PIE_OFFSET;  
        yoffset = sin(labelloc) * PIE_OFFSET * aspectratio;  
        pieslice(x+xoffset,y-yoffset,startangle,endangle,radius);  
    }  
    else  
        pieslice(x,y,startangle,endangle,radius);  
  
    x += cos(labelloc) * lradius;  
    y -= sin(labelloc) * lradius * aspectratio;  
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);  
    // Set text justification depending upon location of pieslice  
    if (labelangle >= 300 || labelangle < 60)  
        settextjustify(LEFT_TEXT,CENTER_TEXT);  
    else if (labelangle >= 60 && labelangle < 120)  
        settextjustify(CENTER_TEXT,BOTTOM_TEXT);  
    else if (labelangle >= 120 && labelangle < 240)  
        settextjustify(RIGHT_TEXT,CENTER_TEXT);  
}
```

```

else
    settextrjust(CENTER_TEXT.TOP_TEXT);
    sprintf(buffer,"%3.1ff%%",showpercentage);
    if (offsetflag)
        outtextxy(x+xoffset,y-yoffset,buffer);
    else
        outtextxy(x,y,buffer);
    startpercentage += showpercentage;
    showkey(color,fill,label);
}

```

5.1.2 建立条形图

BGI提供两个专用函数画条形图：`bar()`和`bar3d()`。我们在第三章简要地考查过这两个函数。现在将仔细观察怎样才能实际使用它们以建立表示的条形图。

创建一个条形图本身是很简单的。但是，复杂性出现在当我们试图添加各种装饰以建立专业化外观的条形图或自动化图形生成过程。本节提供的一个条形图程序称为`barutil.cpp`，它自动生成一个针对一组数据的条形图，但为简化而省去了某些通用性。更为特殊的是，该程序使用BGI `bar()`函数建立像图5.4那样的条形图。列表5.2示出完整的`barutil.cpp`程序。显示的数据包含在以下的三个变量中，

```

int yvalues[] = {5, 4, 7, 6};
char *xstrings[] = {"Jan", "Feb", "March", "April"};
char title[] = "Sales (in thousands) ";

```

在本例中，条形图图解了四个月的商业销售情况。图的题目存放在称为`title`的字符数组中。条形的数据放在`yvalues`数组。为简单起见，这些值被限制为整数存放在`xstrings`。数组中的字符数组列出了这些条形的标签。这些标签将出现在每一条形的水平轴之下。最后，由于绘制有4个条形，所以`NUM_BARS`常数被置为4。

绘制该图的第一步是调用列表5.2中提供的`display_chart()`函数，如你所见，这个函

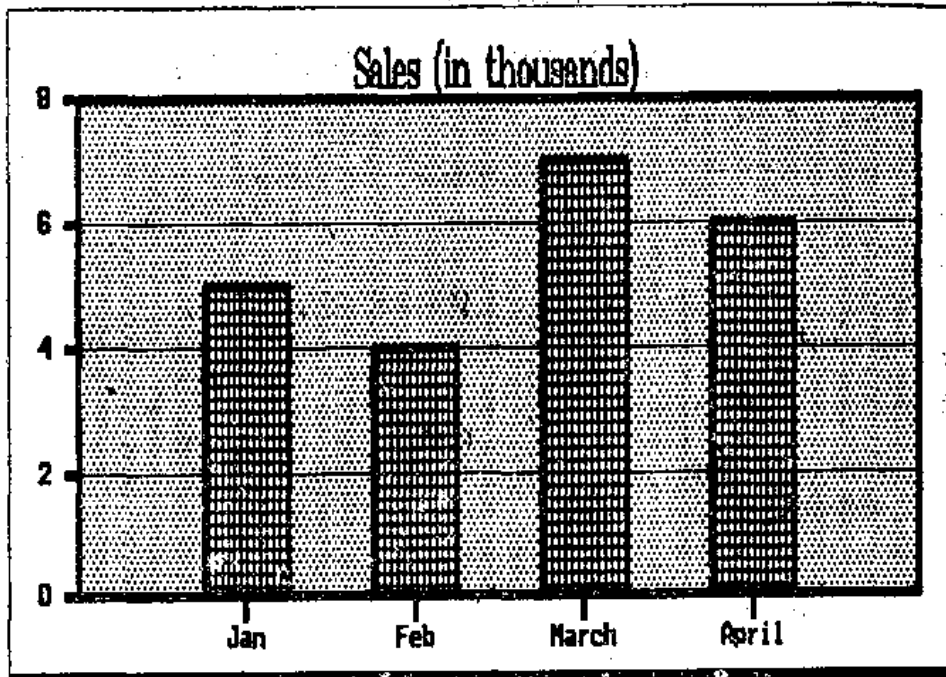


图 5.4 `barutil.cpp`的输出

数传送各种要显示的值和标题,以及一个称为numrules的值。这个整型值指定在图中使用的水平区的数目。在我们的例子中,将使用4个水平尺,你可能需要试验用这个值产生不同的图形。

基本上, `display_chart()` 负责执行4个重要任务。首先,它基于屏幕的大小以及 `SCREEN_BORDER_X` 和 `SCREEN_BORDER_Y` 定义的像素边界,计算出图形的边界。其次,把标题写在屏幕上,为适应各种长度的标题,使用在第四章开发的 `scaletext()` 函数修改版本自动调整正文以适应图形的项。设计这个 `scaletext()` 版本以便它适合在x和y方向把正文延伸同样的数量。这是试图避免使正文在一个方向延伸太长以至变成不可读。

`display_chart()` 的第三部分由对 `getmax()` 函数的调用和一个while循环组成。它确定y(垂直)轴的最大值。`getmax()` 函数返回包含在 `yvalues` 的值数组中的最大值。这个值保留在 `maxyvalue` 中。while循环增加 `maxyvalue`, 直到它达到一个数目。这个数目是调用 `display_char()` 时定义的水平尺数的倍数。while循环确保程序选出图形的最大值,它将产生能标在图中每个水平尺的y轴上的总数。

最后,调用 `drawchart()`, 这个函数也在列表5.2中提供 (`barutil.cpp`)。它实际画出条形图的背景和水平与垂直轴的标签以及条形自身。背景是通过调用 `bar3d()` 来建立的。请注意,我们正使用这个函数以替代 `fillpoly()`, 因为它的变元易于说明。或者,也可以使用函数 `bar()`, 但它不能像 `bar3d()` 那样画出背景的边界。

在 `drawchart()` 中,在调用 `bar3d()` 和for循环之后的4行用于绘制横跨背景的水平尺,并创建厚厚的散列标记和y轴上的标签。计算散列标记之间的距离并把它存放在变量 `offset` 中。在for循环中使用它以便置放横跨图形的尺子。直接跟随for循环的两个语句用于编写对应于水平轴0值的标签。

往下,计算比例因子,后面将用它来精确确定每一条形画多高。执行这个计算的语句是:

```
scale = (double)(bottom - top) / (double)maxyvalue;
```

如上所示,比例因子依赖于图形的总高度和在 `display_chart()` 中确定的最大值,图5.5示出条形的比例因子计算。

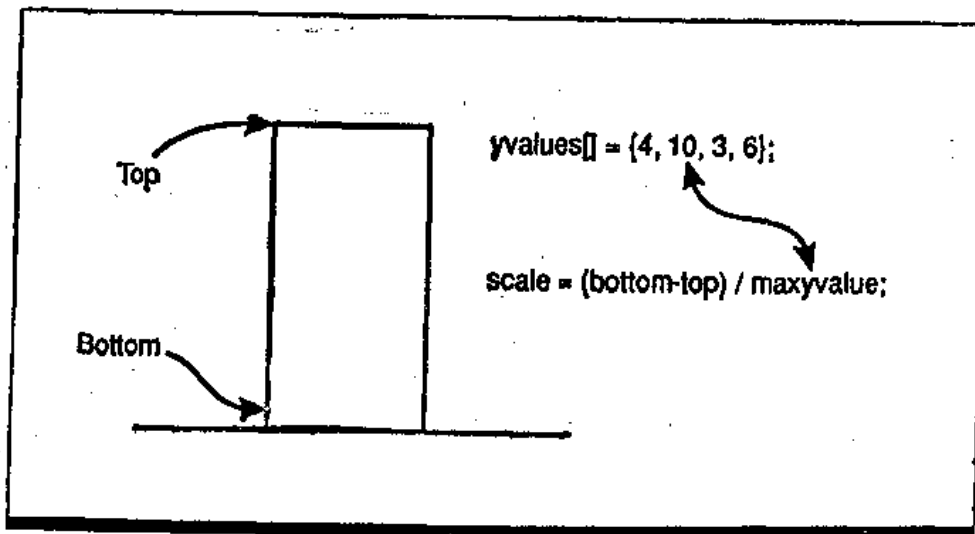


图 5.5 计算条形的高

跟随这个语句的几行代码类似于前面讨论的，但是，这些代码现在在水平轴为分区和标签作标记。for循环使用xstrings数组中的字符串作为标签，同时在for循环中还有对bar3d()的调用。这个语句实际上建立每一个条形。请注意，这些对bar3d()的调用使用深度0，以便不带深度画条形和对背景一样使用bar3d()而不是较简单的bar()函数，因为它提供每一个条形的边界。

为了使这个图形程序适应你的数据，你将需要在xvalues、yvalues和title中提供你自己的值，并且改变NUM_BARS所表示的数，以对应你具有的独特数据点的数目。最后，你可以选择NUM_RULES的值，它将影响程序所绘制的水平尺数目和其后在垂直轴上的标签数目。

某些你可能想添加的内容是垂直和水平轴的标题。还有，你可能想修改程序，以便它沿着垂直轴接受浮点值而不仅仅是整数。当然，还有许多其它修改你可能想去试验。

• 列表5.2 barutil.cpp

```
// barutil.cpp -- This is a utility program that generates a professional
// looking bar graph. It supports several options.
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
const int SCREEN_BORDER_Y = 20; // Put a 10 pixel border at top and bottom
const int SCREEN_BORDER_X = 20; // Make a 10 pixel border on each side
const int NUM_RULES = 4;        // Use four rules on the backdrop
const int HASH_WIDTH = 8;       // Make hash marks on rules this long
const int NUM_BARS = 4;         // Four strings on x axis

void display_chart(char *title, char *xstrings[], int yvalues[],
                  int numrules);
int getmax(int values[]);
void drawchart(int left, int top, int right, int bottom, int numrules,
              int maxyvalue, char *xstrings[], int yvalues[]);
void scaletext(int left, int top, int right, int bottom, char *string);

int maxx, maxy;                // Dimensions of graphics screen

main()
{
    int gmode, gdriver = DETECT;
    int yvalues[] = {5, 4, 7, 6};
    char *xstrings[] = {"Jan", "Feb", "March", "April"};
    char title[] = "Sales (in thousands) ";

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    maxx = getmaxx();    maxy = getmaxy();
    display_chart(title, xstrings, yvalues, NUM_RULES);
    getch();
    closegraph();
    return(0);
}

void display_chart(char *title, char *xstrings[], int yvalues[], int numrules)
{
    int maxyvalue, left, top, right, bottom;

    if (numrules < 0) numrules = 1; // Make sure numrules is > zero
    // Determine border points of backdrop
    left = SCREEN_BORDER_X + maxx / 8;
```

```

top = SCREEN_BORDER_Y;
right = maxx - SCREEN_BORDER_X;
bottom = maxy - SCREEN_BORDER_Y;
// Display the title at the top. Scale it to fit.
scaletext(left,0,right,top,title);
// Determine a maximum value for the chart scale. It should be at least
// as large as the largest list of values and a multiple of the number
// of rules that the user desires; numrules should not be greater than
// maxyvalue.
maxyvalue = getmax(yvalues); // Get the largest y value
while ((maxyvalue % numrules) != 0)
    maxyvalue++;
drawchart(left,top,right,bottom,numrules,maxyvalue,xstrings,yvalues);

```

// Find and return the largest value in the array of values

```

int getmax(int values[])
{

```

```

    int i=0, largest=1;

```

```

    for (i=0; i < 4; i++)
        if (values[i] > largest)
            largest = values[i];
    return(largest);
}

```

```

void drawchart(int left, int top, int right, int bottom, int numrules,
               int maxyvalue, char *xstrings[], int yvalues[])
{

```

```

    int i, height, inc;
    double scale;
    char buffer[10];
    int offset; // Draw horizontal rules this many pixels apart

```

```

    // Make a thick border
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setfillstyle(CLOSE_DOT_FILL,BLUE);
    // Use bar3d with a depth of zero to draw a bar with a border
    bar3d(left,top,right,bottom,0,0);

    offset = (bottom - top) / numrules + 1;
    inc = maxyvalue / numrules;
    settextjustify(RIGHT_TEXT,CENTER_TEXT);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    for (i=0; i<numrules; i++) {
        // Draw rule as thin line
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        line(left,top+i*offset,right,top+i*offset);
        // Show thick hash mark
        setlinestyle(SOLID_LINE,0,THICK_WIDTH);
        line(left-HASH_WIDTH,top+i*offset,left,top+i*offset);
        sprintf(buffer,"%d",inc * (numrules - i));
        outtextxy(left-HASH_WIDTH-textwidth(buffer),top+i*offset,buffer);
    }

```

```

    line(left-HASH_WIDTH,bottom,left,bottom); // Draw bottom hash mark
    outtextxy(left-HASH_WIDTH-textwidth(buffer),bottom,"0");
    // Now write out the values for the horizontal axis
    // Figure the amount to scale all bars
    scale = (double)(bottom - top) / (double)maxyvalue;
    setfillstyle(HATCH_FILL,BLUE); // Make all bars HATCH_FILL
    offset = (right - left) / (NUM_BARS + 1);
    settextjustify(CENTER_TEXT,TOP_TEXT);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);

```

```

for (i=1; i<=NUM_BARS; i++) {
    // Show thick hash mark
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    line(left+i*offset,bottom,left+i*offset,bottom+HASH_WIDTH);
    outtextxy(left+i*offset,bottom+HASH_WIDTH+2,xstrings[i-1]);
    // Draw the bars for the values. Make the total width of one of the
    // bars equal to half the distance between two of the hash marks on
    // the horizontal bar.
    height = yvalues[i-1] * scale;
    bar3d(left+i*offset-offset/4,bottom-height,
          left+i*offset+offset/4,bottom,0,0);
}

// Scale the text so it fits into the specified rectangle. This
// routine is a modification of the scaletext function in Chapter 3.
// It uses setusercharsize() to try to scale the text so that it
// looks good. Uses Triplex font.
void scaletext(int left, int top, int right, int bottom, char *string)
{
    int height;

    settextjustify(CENTER_TEXT,TOP_TEXT);
    setusercharsize(1,1,1,1);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
    // In height calculation make room for letters extending below line
    // by multiplying textheight by 5/4.
    height = textheight(string) * 5 / 4;
    if (height > bottom-top) { // Text is too tall so find how
        // Try scaling down x and y by same amount in order to keep
        // text well proportioned
        setusercharsize(bottom-top,height,bottom-top,height);
        settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
        // Enough room so write it
        if (textwidth(string) <= right - left)
            outtextxy((right+left)/2,top,string);
        else {
            // Doesn't fit with equal scaling - so squash it in!
            setusercharsize(1,1,1,1);
            settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
            setusercharsize(right-left,textwidth(string),bottom-top,height);
            settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
            outtextxy((right+left)/2,top,string);
        }
    }
    // String is too long - try scaling equally
    else if (textwidth(string) > right-left) {
        setusercharsize(right-left,textwidth(string),
                        right-left,textwidth(string));
        settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
        // Enough room so write it
        if (textheight(string)*5/4 <= bottom-top)
            outtextxy((right+left)/2,top,string);
        else { // Doesn't fit with equal scaling - so-squash it in!
            setusercharsize(1,1,1,1);
            settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
            setusercharsize(right-left,textwidth(string),bottom-top,height);
            settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
            outtextxy((right+left)/2,top,string);
        }
    }
}

```

```

}
else // Just write out text -- enough room
    outtextxy((right+left)/2,top,string);
}

```

5.1.3 三维条形图

在上一节中，我们学习了如何开发能建立二维条形图的实用程序。往下，我们将向你指出怎样修改此程序，以便它能画三维条形图。事实上，由于bar3d()已经用于画条形，我们所做的只不过是改变传送给drawchart()中bar3d()的深度变元，以获得三维条形图。

通常，如果设置条形图的深度为最高条形高度的1/4，则将得到非常满意的结果。因此，bar3d()的一个修改语句是：

```

bar3d(left+i*offset-offset/4,bottom-height,left+i*offset+offset/4,
      bottom,height_of_highest_bar/4,0);

```

你可以很容易通过修改bar3d()中的倒数第二个变元改变条形的深度。但是，如果深度太大，条形可能开始合并，如果发生这种情况，你将应增加条形之间的空间或对条形的深度使用更小的值。

5.1.4 楔形图

你可能要对barutil.cpp程序作的另一种修改(列表5.2)是改动它以便能建立楔形图。如图5.6所示的那样，这种楔形图代表4个计算机销售商的计算机监视器销售情况。为了使图形有趣，把小的监视器图形堆放在每另一个的顶上，就像楔形物一样，为的是表示出销售的总数。

或许建造一个好楔形图的最艰难任务是如何使用正确的图画，在作出设计以满足你的要求之前，用各种顺序的BGI函数来试验。

为了建立楔形图，基本思想是画出小的图，用getimage()取得它的映像，然后根据需要

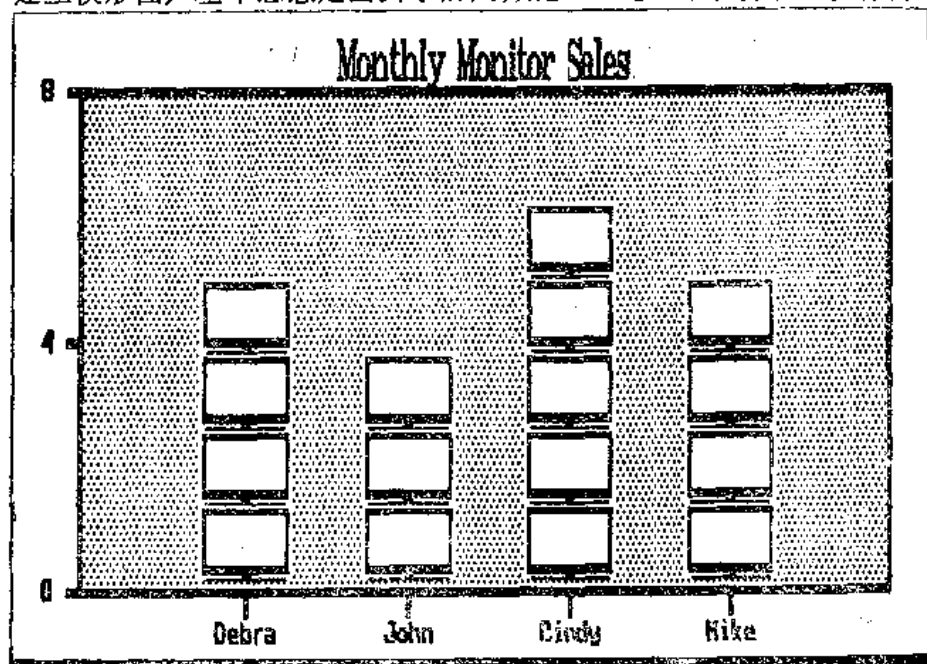


图 5.6 一个样板楔形图

用putimage()重画它。图5.6的楔形图是由barutil.cpp程序建立的。仅用了极少量的修改。不打算再示出整个代码和冒可能使这些修改含混的危险,我们将简单地每次描绘其中之一。

首先,将需要说明少量新变量,我们将用两个常数来定义监视器(楔形图的映像)之一的像素尺寸。此外,我们将需要一个数组来保存监视器的映像。因此,必须把下列的行添加到barutil.cpp的顶部:

```
const int PICT_HEIGHT = 24;      // Pixel height of each picture element
const int PICT_WIDTH = 32;       // Pixel width of each picture element
void *picture;                   // Pointer to picture element image
```

此外,在生成图形时,程序被修改为使用下列的值和标题:

```
int yvalues[] = {5,4,7,6};
char *xstrings[] = {"Debra", "John", "Cindy", "Mike"};
char title[] = "Monthly Monitor Sales";
```

要作的另一种修改涉及建立监视器的映像,在图5.6中使用的监视器映像由下列语句建立,这些语句被添加到main()中,紧接在图形初始化之后。

```
// The following lines create the picture of the monitor
bar(2,2,PICT_WIDTH-2,PICT_HEIGHT-2);
setfillstyle(SOLID_FILL,BLACK);
bar(4,4,PICT_WIDTH-4,PICT_HEIGHT-5);
setcolor(BLACK);
line(PICT_WIDTH-4,PICT_HEIGHT-3,PICT_WIDTH-5,PICT_HEIGHT-5);
line(PICT_WIDTH-8,PICT_HEIGHT-3,PICT_WIDTH-9,PICT_HEIGHT-5);
setcolor(getmaxcolor());
setfillstyle(SOLID_FILL,getmaxcolor());
bar(PICT_WIDTH/2-2,PICT_HEIGHT-2,PICT_WIDTH/2+2,PICT_HEIGHT);
line(4,PICT_HEIGHT,PICT_WIDTH-4,PICT_HEIGHT);
```

一旦画出了对象的图,它的映像就被拷贝并保留到我们早些时候说明过的数组picture中。这里是一些实现这个任务的语句:

```
picture = malloc(imagesize(0,0,PICT_WIDTH,PICT_HEIGHT));
if (picture == NULL) {      // Couldn't allocate memory
    closegraph();
    printf("Failed to allocate picture image space.\n");
    exit(1);
}
getimage(0,0,PICT_WIDTH,PICT_HEIGHT,picture);
putimage(0,0,picture,XOR_PUT);
```

请注意,我们必须在继续下去之前擦去屏幕上原来的图画。这是通过使用早些时候讨论过的XOR_PUT选项完成的。现在,有一个可用的映像,我们准备调用图形程序来建立自己的楔形图。请注意,由于使用malloc(),你还必须包含标题文件alloc.h。

这导致我们进行另一种改变。在drawchart()函数中。我们需要用画一堆楔形物映像的代码替换对bar3d()的调用(它画出条形)。这可以通过用一个for循环替换对bar3d()的调用来完成。该循环根据代表图形值的高度来堆放picture的映像。下面是新的for循环:

```
for (j=PICT_HEIGHT; j<=height; j += PICT_HEIGHT)
    putimage(left+i*offset-PICT_WIDTH/2,bottom-j-4,picture,COPY_PUT);
```

你还将需要对这个for循环说明一个称为j的整变量。请注意,这个语句只画出楔形物

映像的整个副本，它不画部分的映像。如果你需要这个附加的功能，可以重写映像的百分率试试。

5.2 动 画 图

图和框图不一定是静态和无生命的，使图形更为有趣的一种方法是添加动画。本节讨论通过使用称为间隔化的技术来获得动画化条形图。

简而言之，“间隔化”是一个进程，其中，对象的初始和最后形状是预定义的，程序自动计算它们之间的中间状态。通过顺序遍历这些中间状态，对象被有效地动画化。在第七章，我们将更为详细地考察间隔化和其它的动画技术。

对于条形图中的条形，初始状态仅是0高度的条形，而最后的映像是以它的全高度显示的条形。条形的中间映像是在不断增加的高度上描绘的条形。动画的平滑性通常依赖于对象中间状态之间的变化数目。对此的另一种说法是：平滑性依赖于在动画中使用的中间状态的数目。

列表5.3中示出的是表示间隔化概念的程序。它可应用于条形图。它的最后输出在图5.7中示出。这些代码仅展示出概念，如果你需要用它进一步试验，可以把它综合到早些时候建立的barutil.cpp程序中试试。

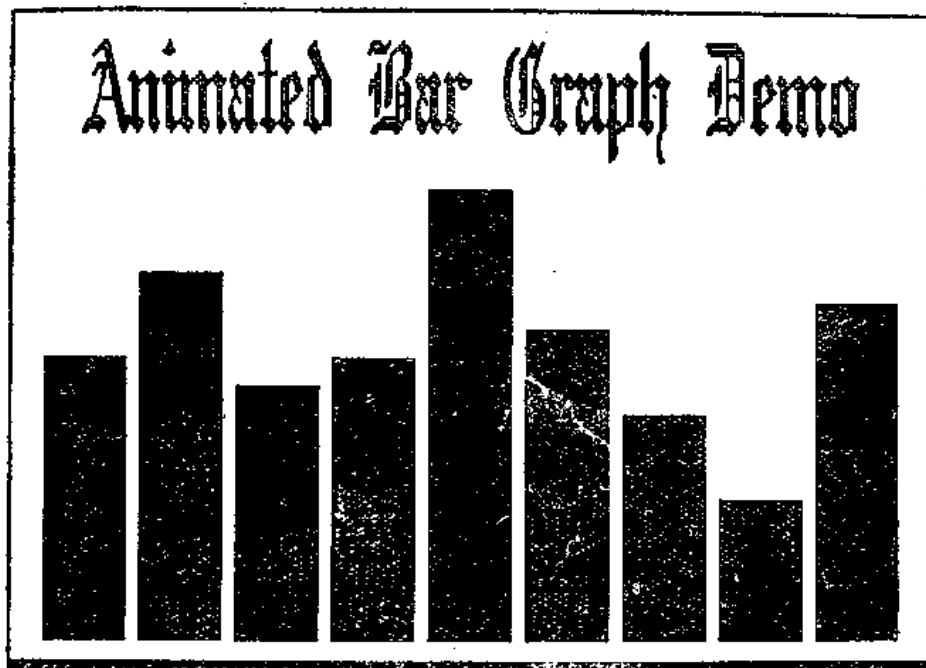


图 5.7 animate.cpp的最后输出

• 列表5.3 animate.cpp

```
/ animate.cpp -- Demonstrates how an animated bar graph can be created
#include <graphics.h>
#include <conio.h>
#include <dos.h>
```

```

const int NUM_STEPS = 10;           // Controls the "growth" rate of the
                                     // bars as they are animated
void animatedbar(int left, int top, int right, int bottom);

main()
{
    int gmode, gdriver = DETECT;
    char title[] = "Animated Bar Graph Demo";

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    // Write a header to the graph
    settextjustify(LEFT_TEXT, TOP_TEXT);
    settextstyle(GOTHIC_FONT, HORIZ_DIR, 5);
    // Center the title on the first row of the screen
    outtextxy((getmaxx() - textwidth(title)) / 2.0, title);
    setfillstyle(SOLID_FILL, WHITE);
    // Draw and animate the bars one at a time
    animatedbar(50, 100, 100, 200);
    animatedbar(110, 70, 160, 200);
    animatedbar(170, 110, 220, 200);
    animatedbar(230, 100, 280, 200);
    animatedbar(290, 40, 340, 200);
    animatedbar(350, 90, 400, 200);
    animatedbar(410, 120, 460, 200);

    animatedbar(470, 150, 520, 200);
    animatedbar(530, 80, 580, 200);
    getch();
    closegraph();
    return(0);
}

void animatedbar(int left, int top, int right, int bottom)
{
    int i, inc;

    inc = (bottom - top) / NUM_STEPS;
    for (i = 0; i < NUM_STEPS; i++) {
        bar(left, bottom - inc * i, right, bottom);
        delay(20); // 20 ms delay--try experimenting with this
    }
}

```

第六章 二维图形技术

本章建立开发二维图形程序的基础。我们从探索屏幕坐标和世界坐标之间的关系开始,提供在这些坐标之间变换的技术。此外,还将引进变换的概念,它可用于描述图形对象如何能在二维中处理。作为探讨的一部分,我们将开发一个称为matrix.cpp的程序包。它适用于许多所提供的概念。

6.1 屏幕坐标

BGI支持若干不同的视频适配器和模式。每个都带有它自己的彩色范围、分辨率和存储页面的数目。遗憾的是,这种灵活性带来了所开发的图形程序的兼容性问题。怎样才能编写一个程序,使得它在各种图形模式下同等地工作呢?我们已经遇到的一个问题是屏幕分辨率。不仅要考虑各种模式的分辨率,而且还必须针对各种模式的屏幕长宽比进行调整。

每一种图形模式都有和它相联的长宽比。长宽比基于每一像素的宽与高之间的比率。当我们试图在屏幕上绘制一个特定大小和形状的图时,这个比率是很重要的。例如,如果用CGA 320×200模式,每一像素的高差不多是宽的两倍。因此,如果我们画4×4的像素块,将得不到屏幕上的正方形。为精确地绘制正方形,我们必须如图6.1那样调整模式的长宽比。

幸好,BGI提供了getaspectratio()函数,我们可以用它来确定当前模式的长宽比率。它的函数原型在graphics.h中说明为:

```
void far getaspectratio(int far *xasp, int far *yasp);
```

可以相除变元xasp和yasp来计算出屏幕长宽比。在大多数模式下,像素都是高稍微比宽长的。因此,BGI规格化yasp值为10000而返回某些小于10000的xasp值。因此,屏幕的长宽比可如下计算:

```
aspectratio = (double)xasp / (double)yasp;
```

然后,可用这个值对每一个y的尺寸进行处理,以便使用正确的比例和定位来绘制对象。例如图6.1所示的4×4像素的正方形的高变为:

```
screen_height = 4 * aspectratio.
```

由于xasp通常小于yasp,故aspectratio被置为某些分数值,而screen_height计算为小于4的值。这产生一正确比例的正方形如图6.1所示。

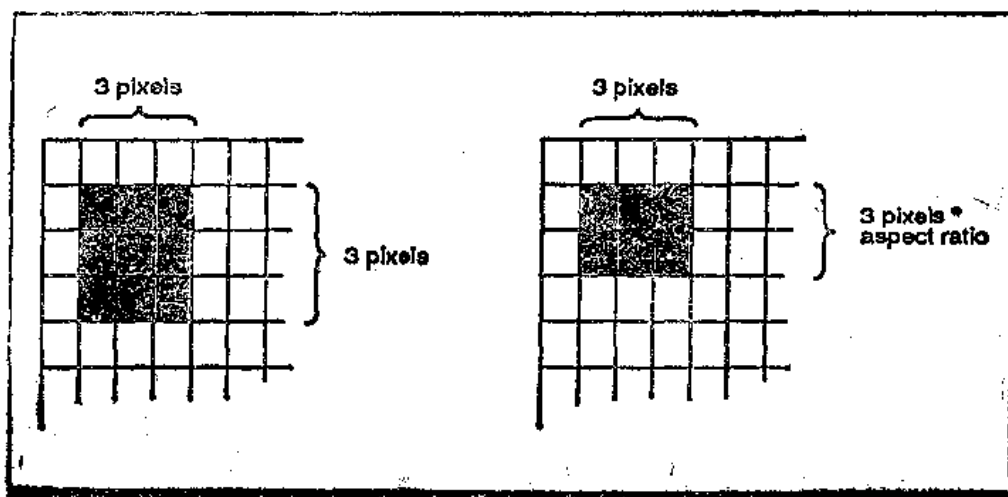


图 6.1 按屏幕的长宽比作调整

6.2 屏幕和世界坐标

在上一节，我们学习了如何调整存在于大多数图形模式中x和y尺寸之间的差异。类似地，我们必须调整图形对象，使它们按不同屏幕分辨率在图形模式下显示时保持其大小。为实现这点，我们将使用称为“世界坐标”的标准坐标。

迄今为止，我们仅用屏幕坐标工作。当对带有不同分辨率的图形模式工作时，这种方法可能会出问题。例如，一个10×10的正方形在320×200模式下和在640×200模式下看起来大有差别。此外，如果我们定义一个对象定位在(500, 150)，则它在320×200模式下可能甚至不出现。我们需要的是变换对象的能力，以便能按比例和正确定位显示它们。为此，我们将使用世界坐标（例如，英寸、英尺、公尺等）来表示对象，然后把它们变换为屏幕坐标，以便它们在被绘制之前正常地适应屏幕。

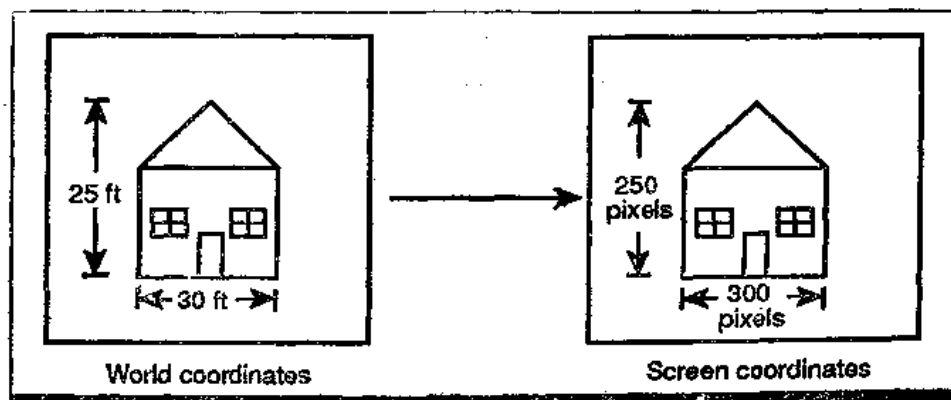


图 6.2 从世界坐标向屏幕坐标变换对象

通常，我们需要像图6.2所示在世界坐标和屏幕坐标之间进行变换。我们可以使用图6.3所示的关系实现这点。等式

$$\begin{aligned} x' &= a * x + b \\ y' &= c * y + d \end{aligned}$$

定义了世界坐标中的点(x, y)如何能变换为屏幕坐标(x', y')。值L1, T1, R1和B2

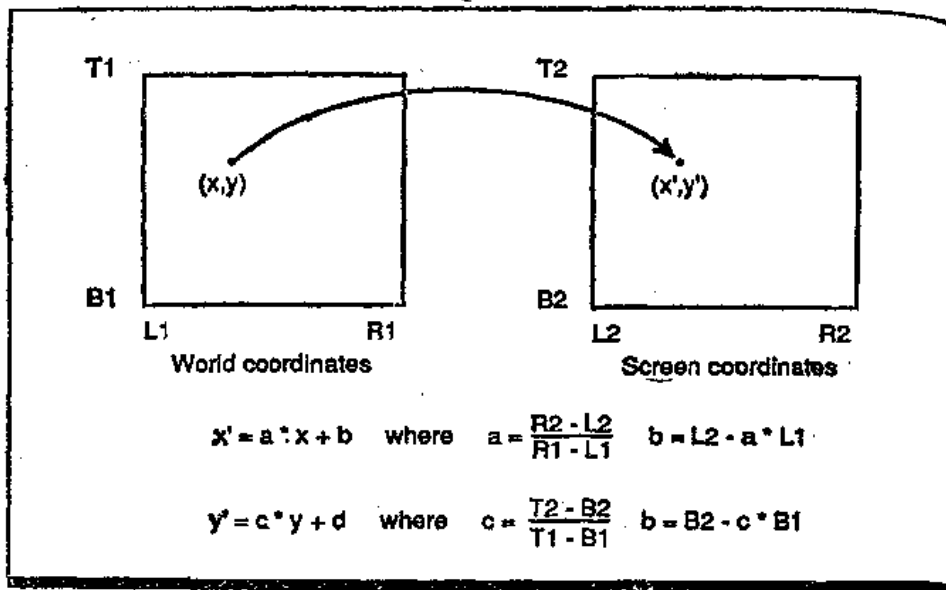


图 6.3 从世界坐标向屏幕坐标变换的方程

定义对象被变换后进入的屏幕区域的大小。现在，让我们把这些等式放到程序包 `matrix.cpp` 中。这个文件和它的标题的代码 `matrix.h` 在列表 6.1 和列表 6.2 中示出。

首先，我们需要两个函数，它们能设置世界坐标和屏幕坐标的范围，这两个函数称为 `set_window()` 和 `set_viewport()`。每一个函数被使用时都带有区域的左、顶、右和底界，而在 `matrix.cpp` 中，为这些范围内部设置了适当的一组全程变量。此外，`set_viewport()` 计算在图 6.3 中示出的比率。例如，两个函数调用

```
set_window(0.0,0.0,3.0,3.0);
set_viewport(0.0,getmaxx(),getmaxy());
```

设置了世界坐标和屏幕坐标中的值的范围，以便世界坐标中 $(0.0, 0.0)$ 和 $(3.0, 3.0)$ 之间的所有对象都被变换为全屏幕。超出这个范围的对象被截去和不显示。

作为另一个例子，你可以使用同样的世界坐标设置，但定义屏幕上的观察端口为：

```
set_viewport(0.0,getmaxx()/2,getmaxy()/2);
```

在这种情况下，如上的同样对象现在只出现在屏幕的左上半。类似地，你可以通过减小传递给 `set_window()` 的区域的大小有效地放大所显示的对象。请注意，当编写程序时，必须用其后对 `set_viewport()` 的调用来匹配每一对 `set_window()` 的调用，以便使变换生效。

`matrix.cpp` 中实际执行坐标系统之间变换的函数称为 `WORLDtoPC()` 和 `PCtoWORLD()`。每一函数把坐标系统中的一点变换为其它坐标系统中的一点。例如，`WORLDtoPC()` 函数用如下代码把世界坐标变换为屏幕坐标：

```
void WORLDtoPC(double xw, double yw, int &xpc, int &ypr)
{
    xpc = (int)(a * xw + b);
    ypr = (int)(c * yw + d);
}
```

变元xw和yw是世界坐标,它们要变换成屏幕坐标xpc和ypc。你可把这些语句和图6.3所示的等式作比较。类似地,PCtoWORLD() 通过如下语句把屏幕坐标变换为世界坐标:

```
void PCtoWORLD(int xpc, int ypc, double &xw, double &yw)
{
    xw = (xpc - b) / a;
    yw = (ypc - d) / c;
}
```

我们将在第十三章的CAD程序使用这些函数,以定义怎样和在哪里显示对象。

6.3 变 换

在上一节中,我们学习了如何把世界坐标中的对象变换到屏幕上。现在,让我们探讨处理这些对象的方法,以便能按不同位置、方向和比例来绘制它们。我们将使用变换来产生这些效果。

“变换”是在图形程序设计中的基本数学运算。从字面上解释,变换是一个函数(或等式),它定义如何把一组数据改变成变换为另一组。例如,假定在屏幕上有一幅车轮的图画。我们需要使它旋转起来,那末,可以使用旋转变换以确定当轮子移动时它怎样受影响。

大多数普通变换是平移、旋转和变比。在本节中我们将详细探讨这些变换的每一种,并添加函数到matrix.cpp,以执行这些操作。表6.1中示出将添加到matrix.cpp中的这些变换函数的表。

表 6.1 matrix.cpp中的二维变换

函 数	描 述
PCtranslatepoly ()	平移屏幕坐标的多边形
PCscalepoly ()	变比屏幕坐标的多边形
PCrotatepoly ()	旋转屏幕坐标的多边形
PCshearpoly ()	剪切屏幕坐标的多边形
WORLDtranslatepoly ()	平移世界坐标的多边形
WORLDscalepoly ()	变比世界坐标的多边形
WORLDrotatepoly ()	旋转世界坐标的多边形

在第十三章开发的CAD程序使用matrix.cpp工具箱以平移和旋转对象。当我们在第十四章探讨三维图形时,将靠变换生成三维对象的透视图。但现在,我们只涉及二维的对象。

6.3.1 平移

最简单的变换之一是产生平移。本质上,“平移”定义一个点如何在空间的一个位置移动到另一位置。例如,如果我们在屏幕左边画了一个方框,并且需要把它移动到右边,我们可以用平移变换来说明每一点必须如何移动。

例如,平移屏幕上一个像素可以这样完成,即,对每一个移动点的x和y坐标添加一适当的值。这可以用C++写成:

```
new_x = x + translate_x;
new_y = y + translate_y;
```

translate_x和translate_y的值可正可负,并分别定义点(x,y)在x和y方向应平移多少。

平移一个对象,如多边形,只不过是平移构成如图6.4解释的多边形的每一点罢了。matrix.cpp中完成这一工作的函数称为PCtranslatepoly(),如下所示:

```
void PCtranslatepoly(int numpoints, int *poly, int tx, int ty)
{
    for (int i=0; i<numpoints*2; i+=2) {
        poly[i] += tx;
        poly[i+1] += ty;
    }
}
```

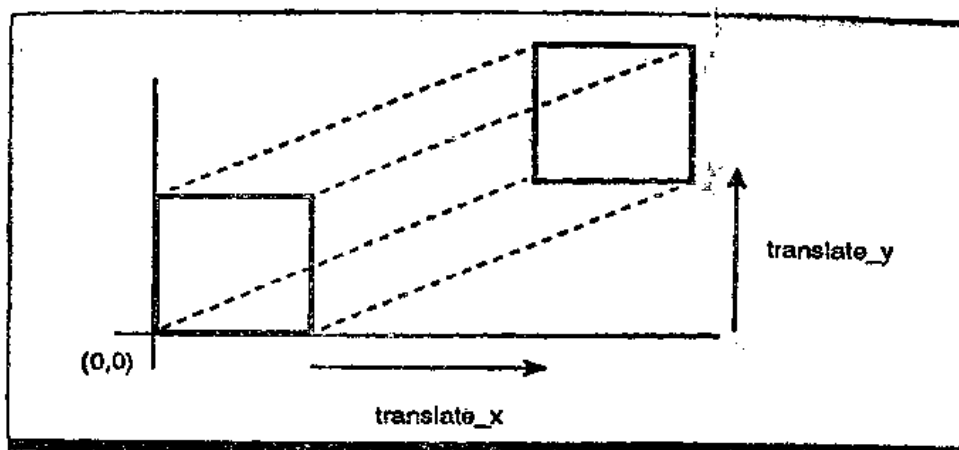


图 6.4 平移一个对象

由于平移量被说明为整数,故它被设计或在屏幕坐标下工作。matrix.cpp中包含类似的函数WORLDtranslatepoly(),它被设计以平移用世界坐标说明的多边形。因此,tx,ty和数组poly都定义成双倍的。

当你重新观察这个函数时,请注意我们使用了捷径方法说明变量i,即对此变量的说明放在for语句的第一行。虽然在C中不允许这种类型的说明,但Turbo C++是支持的。我们将在许多例子中使用它,因为它简化了说明局部变量的任务。

6.3.2 变比一个二维多边形

通过用一个比例因子乘原始多边形的每一坐标,可变比一个二维的多边形。例如,假定我们需要变比图6.5所示的方框。x方向加倍, y方向缩为一半。这只需把以下方程应用

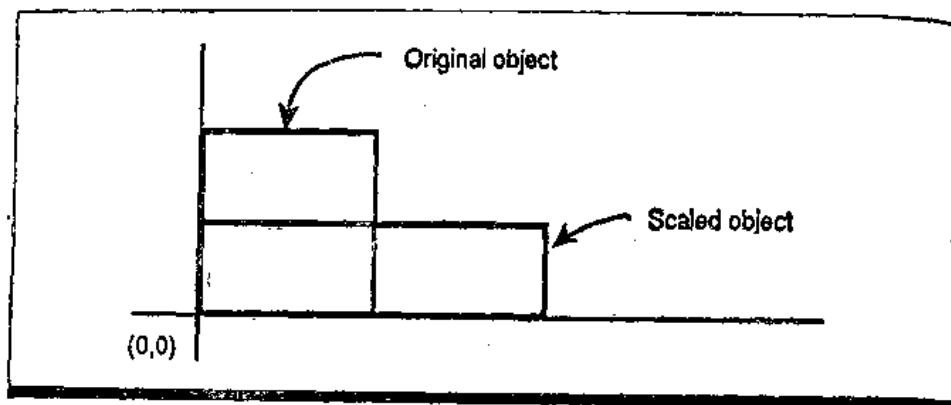


图 6.5 通过对x乘2、对y乘1/2变比一个对象

于方框的每一坐标即可：

```
scaledx = x * scalex;  
scaley = y * scaley;
```

对屏幕坐标的多边形执行这一操作的matrix.cpp中的函数称为PCscalepoly()，如下所示：

```
void PCscalepoly(int numpoints, int *poly, double sx, double sy)  
{  
    for (int i=0; i<numpoints*2; i+=2) {  
        poly[i] *= sx;  
        poly[i+1] *= sy;  
    }  
}
```

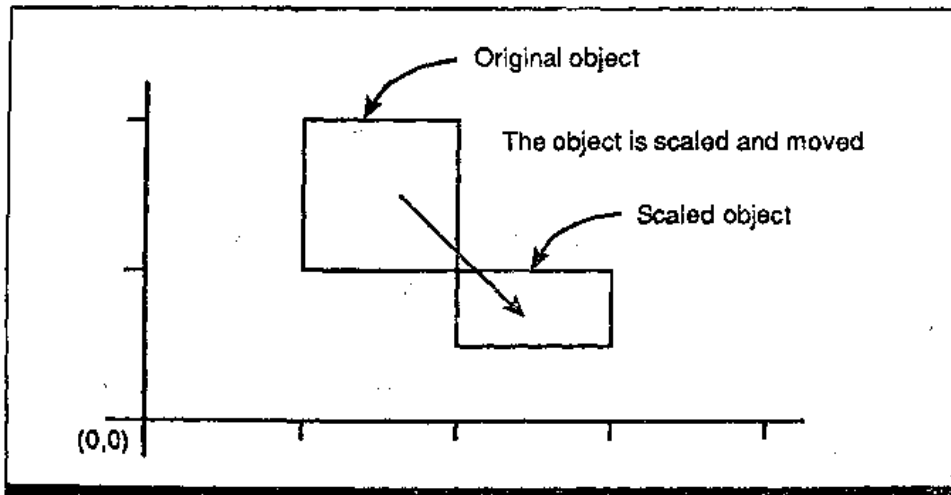


图 6.6 变比不在原点的对象会改变它的大小和位置

对在世界坐标中说明的多边形相应类似的函数，它称为WORLDscalepoly()。

但是，仅仅调用PCscalepoly()去变比对象并不总能产生期望的结果。由于对象中的每一坐标被比例因子相乘，因此该对象可能如图6.6所示随变比而移动，除非坐标之一是(0, 0)，这个坐标将停留在同一地方。因此，如果你需要变比一个对象，但仍保持它的点之一静止不动，以便该对象不移动，则必须采取两个附加的步骤。首先，必须平移该对象，以便你需要变比（但保持固定）的点移到原点。其次，在应用变比变换之后，该对象必须按先前的平移量值平移回去。纯效果是把对象变比，但平移到原点且送回的点保持不动。这种变比顺序（在图6.7解释）实际上比第一种更有用。

例如，下面的代码加倍了数组poly中的多边形的大小。先平移它以便它的第一点在原点，变比它，然后把多边形往回平移。

```
px = poly[0];  
py = poly[1];  
  
PCtranslatepoly(numpoints, poly, -px, -py);  
PCscalepoly(numpoints, poly, 2.0, 2.0);  
PCtranslatepoly(numpoints, poly, px, py);
```

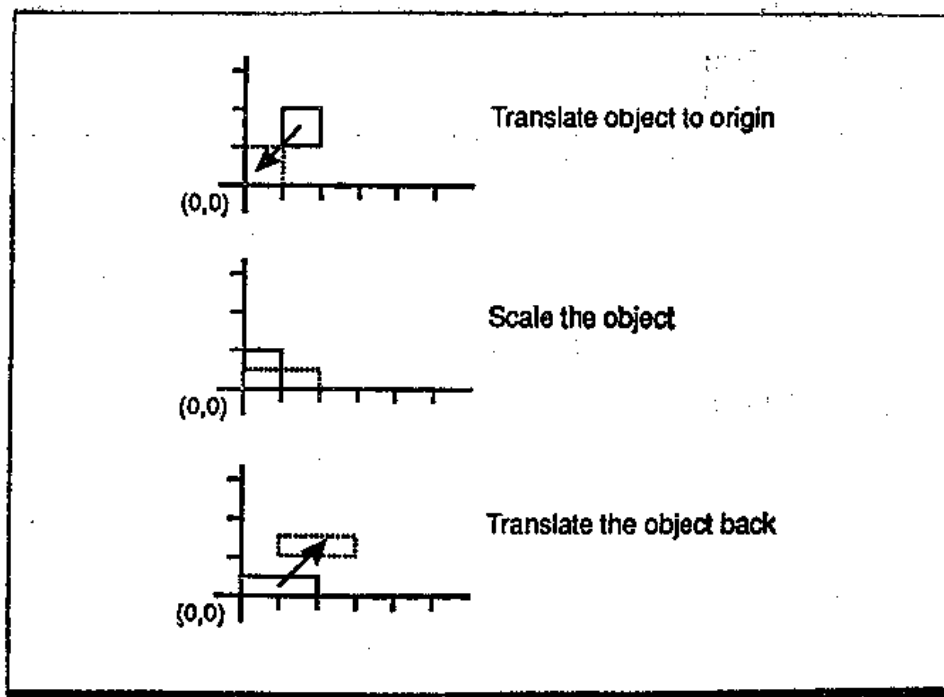


图 6.7 正常变比不在原点的对象的过程

6.3.3 旋转一个二维多边形

另一个有用的变换是围绕一点旋转一个对象。执行这种旋转的方程是：

```
rotatedx = x * cos(angle) - y * sin(angle);
rotatedy = x * sin(angle) + y * cos(angle);
```

不打算涉及推导这些方程的几何学，让我们从用户的观点参看它们是如何工作的。方程右边的x和y表示正被旋转的点，而angle变量说明该点旋转的角度。在Turbo C++中，这个值需要用弧度表示。由于说明角度用度更为普遍，我们将用直接插入函数 `torad()` 把按度表示的角转换为按弧度表示：

```
inline double torad(int d)
{
    return((double)(d) * 3.14159 / 180.0);
}
```

回想一下第五章，直接插入函数类似于宏；但是，它们支持类型检查并采用其它函数遵守的范围规则。

为成功地旋转一个对象，我们需要找出该对象围绕它旋转的点，平移它到原点，旋转多边形，然后把对象平移回去。执行旋转操作的 `matrix.cpp` 中的函数称为 `PCrotatepoly()`（见下面的程序），它能用于旋转一个多边形，如图6.8所示。

```
void PCrotatepoly(int numpoints, int *poly, double angle)
{
    int i, x, y;
    double rad, costheta, sintheta;

    rad = torad(angle);
    costheta = cos(rad);
    sintheta = sin(rad);
```

```

for (i=0; i<numpoints*2; i+=2) {
    x = poly[i];
    y = poly[i+1];
    poly[i] = x * costheta - y * sintheta / aspectratio;
    poly[i+1] = x * sintheta * aspectratio + y * costheta;
}

```

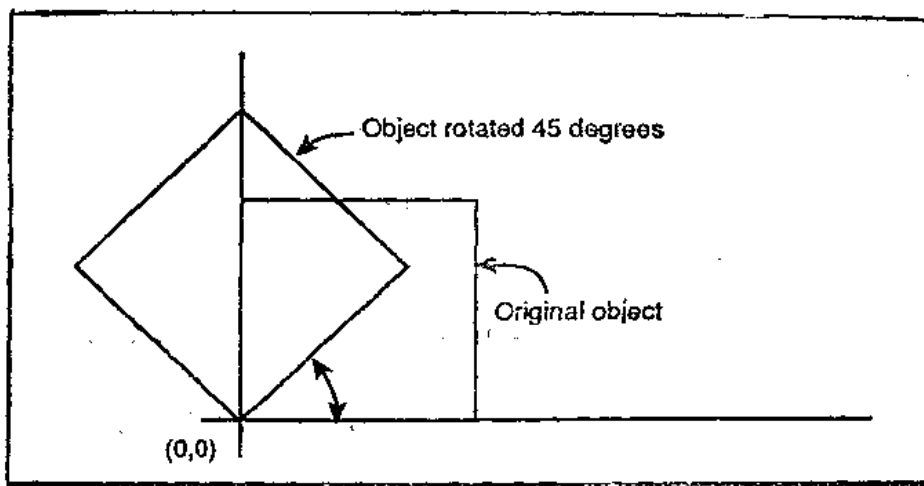


图 6.8 旋转一个对象

请注意，方程中的y值被aspectratio调整。要求这点是因为这个函数按屏幕坐标操作。函数WORLDrotatopoly()有类似的代码，不过，由于它被设计或用世界坐标工作，故不需要这些调整。

6.3.4 剪切变换

某些变换产生一些比较有趣的效果，属于这个范畴的更为普通的转换之一是剪切对象，把它送到应用的地方，如图6.9所示。它涉及用比例因子乘对象的每一坐标和添加偏移值。产生剪切效果的一对方程是：

```

shearedx = x + c * y;
shearedy = d * x + y;

```

如同对于旋转那样，我们需要使多边形围绕一点剪切，于是必须平移多边形到原点。应用剪切变换，然后把该对象平移回去。此外，如果我们变换屏幕上的像素，则将需要补偿监视器的长宽比。能用来剪切位于原点的多边形的例程是：

```

void PCshearpoly(int numpoints, int *poly, double c, double d)
{
    int i, x, y;

    for (i=0; i<numpoints*2; i+=2) {
        x = poly[i];
        y = poly[i+1];
        poly[i] = x + c * y / aspectratio;
        poly[i+1] = d * x * aspectratio + y;
    }
}

```

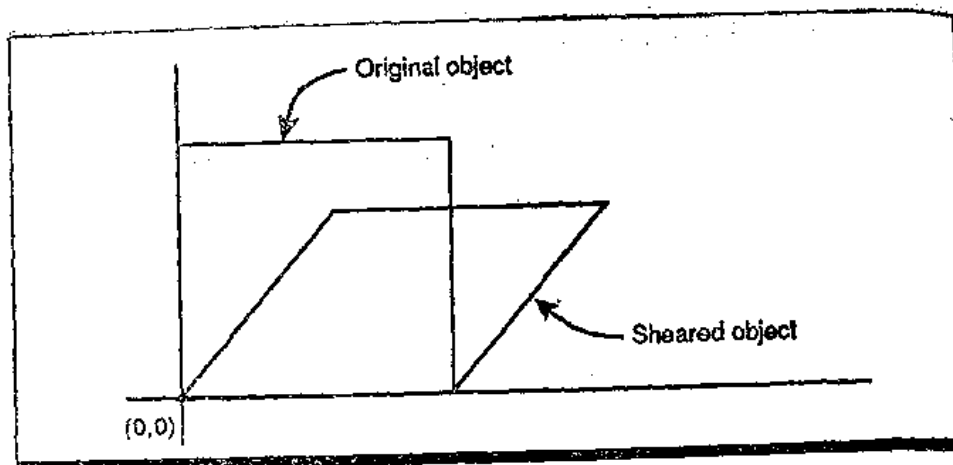


图 6.9 剪切一个对象

• 列表6.1 matrix.h

```
// matrix.h -- The following are a list of matrix operations that
// are supported by matrix.cpp.

void PCtranslatepoly(int numpoints, int *poly, int tx, int ty);
void PCscalepoly(int numpoints, int *poly, double sx, double sy);
void PCrotatepoly(int numpoints, int *poly, double angle);
void PCShearpoly(int numpoints, int *poly, double c, double d);
void copypoly(int numpoints, int *polyfrom, int *polyto);
void WORLDtranslatepoly(int numpoints, double *poly, double tx, double ty);
void WORLDSscalepoly(int numpoints, double *poly, double sx, double sy);
void WORLDRotatepoly(int numpoints, double *poly, double angle);
void copyWORLDpoly(int numpoints, double *polyfrom, double *polyto);
void PCpolytoWORLDpoly(int numpoints, int *PCpoly, double *WORLDpoly);
void WORLDpolytoPCpoly(int numpoints, double *WORLDpoly, int *PCpoly);
void set_window(double xmin, double xmax, double ymin, double ymax);
void set_viewport(int xmin, int xmax, int ymin, int ymax);
void WORLDtoPC(double xw, double yw, int &xpc, int &ypc);
void PCtoWORLD(int xpc, int ypc, double &xw, double &yw);
```

• 列表6.2 matrix.cpp

```
// matrix.cpp -- This file supports the matrix operations that will be
// used in the two- and three-dimensional graphics programs later in
// this book.
#include <graphics.h>
#include <math.h>
#include "matrix.h"

extern double aspectratio; // Define the aspect ratio.
static double a, b, c, d; // Internal values set by
static int xvl, xvr, yvl, yvb; // set window and viewport
static double xwl, xwr, ywl, ywb; // to map to screen coordinates

// Inline function to convert degrees to radians
inline double torad(int d)
{
    return((double)(d) * 3.14159 / 180.0);
}
```



```

// Translates a two-dimensional polygon by tx and ty. The polygon
// should be in screen coordinates. Note the declaration style of i.
void PCtranslatepoly(int numpoints, int *poly, int tx, int ty)
{
    for (int i=0; i<numpoints*2; i+=2) {
        poly[i] += tx;
        poly[i+1] += ty;
    }
}

// Scales a polygon by sx in the x dimension and sy in the y
// dimension. The polygon should be in screen coordinates.
void PCscalepoly(int numpoints, int *poly, double sx, double sy)
{
    for (int i=0; i<numpoints*2; i+=2) {
        poly[i] *= sx;
        poly[i+1] *= sy;
    }
}

// Rotates a polygon by the number of degrees specified in angle.
// The polygon should be in screen coordinates.
void PCrotatepoly(int numpoints, int *poly, double angle)
{
    int i, x, y;
    double rad, costheta, sintheta;

    rad = torad(angle);
    costheta = cos(rad);
    sintheta = sin(rad);
    for (i=0; i<numpoints*2; i+=2) {
        x = poly[i];
        y = poly[i+1];
        poly[i] = x * costheta - y * sintheta / aspectratio;
        poly[i+1] = x * sintheta * aspectratio + y * costheta;
    }
}

// Shears a polygon by applying c to the x dimension and d to
// the y dimension. The polygon should be in screen coordinates.
void PCshearpoly(int numpoints, int *poly, double c, double d)
{
    int i, x, y;

    for (i=0; i<numpoints*2; i+=2) {
        x = poly[i];
        y = poly[i+1];
        poly[i] = x + c * y / aspectratio;
        poly[i+1] = d * x * aspectratio + y;
    }
}

// Copies from one integer polygon to another
void copypoly(int numpoints, int *polyfrom, int *polyto)
{
    for (int i=0; i<numpoints*2; i++)
        polyto[i] = polyfrom[i];
}

```

```

// Translates a polygon in world coordinates
void WORLDtranslatepoly(int numpoints, double *poly, double
{
    for (int i=0; i<numpoints*2; i+=2) {
        poly[i] += tx;
        poly[i+1] += ty;
    }
}

// Scales a polygon in world coordinates by sx and sy
void WORLDscalepoly(int numpoints, double *poly, double sx, double sy)
{
    for (int i=0; i<numpoints*2; i+=2) {
        poly[i] *= sx;
        poly[i+1] *= sy;
    }
}

// Rotates a polygon in world coordinates
void WORLDrotatepoly(int numpoints, double *poly, double angle)
{
    int i;
    double x, y;
    double rad, costheta, sintheta;

    rad = torad(angle);
    costheta = cos(rad);
    sintheta = sin(rad);
    for (i=0; i<numpoints*2; i+=2) {
        x = poly[i];
        y = poly[i+1];
        poly[i] = x * costheta - y * sintheta;
        poly[i+1] = x * sintheta + y * costheta;
    }
}

// Copies one polygon in world coordinates to another
void copyWORLDpoly(int numpoints, double *polyfrom, double *polyto)
{
    for (int i=0; i<numpoints*2; i++)
        polyto[i] = polyfrom[i];
}

// Converts a world coordinate to a screen coordinate
void WORLDtoPC(double xw, double yw, int &xpc, int &ypc)
{
    xpc = (int)(a * xw + b);
    ypc = (int)(c * yw + d);
}

// Converts a screen coordinate to a world coordinate
void PCtoWORLD(int xpc, int ypc, double &xw, double &yw)
{
    xw = (xpc - b) / a;
    yw = (ypc - d) / c;
}

// Defines the window used in real world coordinates
void set_window(double xmin, double xmax, double ymin, double ymax)
{
    xw = xmin;    xwr = xmax;
    ywb = ymin;    ywt = ymax;
}

```

```

// Defines the region on the screen in which world objects
// are mapped to
void set_viewport(int xmin, int xmax, int ymin, int ymax)
{
    xvl = xmin;    xvr = xmax;
    yvb = ymin;    yvt = ymax;

    a = (xvr - xvl) / (xwr - xwl);    b = xvl - a * xwl;
    c = (yvt - yvb) / (ywt - ywb);    d = yvb - c * ywb;
}

// Convert a list of polygon points from screen coordinates
// to WORLD coordinates
void PCpolytoWORLDpoly(int numpoints, int *PCpoly, double *WORLDpoly)
{
    for (int i=0; i<numpoints; i++)
        PctoWORLD(PCpoly[i*2], PCpoly[i*2+1], WORLDpoly[i*2],
                  WORLDpoly[i*2+1]);
}

// Convert a list of polygon points from WORLD coordinates to
// PC screen coordinates
void WORLDpolytoPCpoly(int numpoints, double *WORLDpoly, int *PCpoly)
{
    for (int i=0; i<numpoints; i++)
        WORLDtoPC(WORLDpoly[i*2], WORLDpoly[i*2+1], PCpoly[i*2],
                  PCpoly[i*2+1]);
}

```

6.4 矩阵守护程序

列表6.3提供一个简短的程序，你能用它测试包含在matrix.cpp中的矩阵运算。它对数组points中的多边形应用几种在matrix.cpp中开发的变换。在显示每一变换的结果之后，你必须按任一键以转向下一个。

为了编译该程序，你需要把matrix.cpp和mtxtest.cpp连接。

• 列表6.3 mtxtest.cpp

```

// mtxtest.cpp -- This program tests the transforms defined in
// the matrix.cpp file
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include "matrix.h"

int xasp, yasp;
double aspcratio;

main()
{
    int gmode, gdriver = DETECT;
    int points[] = {150,80, 400,80, 400,150, 150,150, 150,80};

    int px, py, numpoints = 5;
    int polycopy[10];
}

```

```

initgraph(&gdriver,&gmode,"\\tc\\bgi");
getaspectratio(&xasp,&yasp);
aspectratio = (double)xasp / (double)yasp;
rectangle(0,0,getmaxx(),getmaxy());
// Test translate polygon
drawpoly(numpoints,points);
copypoly(numpoints,points,polycopy);
px = polycopy[0];
py = polycopy[1];
PCtranslatepoly(numpoints,polycopy,-px,-py);
drawpoly(numpoints,polycopy);
getch();

// Scale the polygon about the point (px, py)
copypoly(numpoints,points,polycopy);
PCtranslatepoly(numpoints,polycopy,-px,-py);
PCscalepoly(numpoints,polycopy,1.5,1.5);
PCtranslatepoly(numpoints,polycopy,px,py);
drawpoly(numpoints,polycopy);
getch();

// Rotate the object about the point (px, py)
copypoly(numpoints,points,polycopy);
PCtranslatepoly(numpoints,polycopy,-px,-py);
PCrotatepoly(numpoints,polycopy,45);
PCtranslatepoly(numpoints,polycopy,px,py);
drawpoly(numpoints,polycopy);
getch();

// Shear the object about the point (px, py)
clearviewport();
drawpoly(numpoints,points);
copypoly(numpoints,points,polycopy);
PCtranslatepoly(numpoints,polycopy,-px,-py);
PCshearpoly(numpoints,polycopy,0,1);
PCtranslatepoly(numpoints,polycopy,px,py);
drawpoly(numpoints,polycopy);
getch();
closegraph();           // Exit graphics mode
return(0);

```

1

第七章 动 画

动画是富有吸引力的图形特征，这有许多原因。它有助于使显示屏的一部分变得醒目，展示某些事情如何进行，形成交互式编程的基础，或使程序更为可视和有趣。本章探讨的各种动画技术，大都是你在图形编程中可能要用到的。包括间隔化，用`getimage()`和`putimage()`移动对象，通过改变调色板颜色使对象动画化，以及使用多屏幕页面形成动作等。

7.1 间 隔 化

第五章已简要地介绍了间隔化。在那里，我们通过它把条形图动画化。现在，我们将仔细探讨间隔化问题，看看它如何能被用来产生更通用的动画效果。

间隔化是一种简单的技术。其基本思想是先定义一对象的起始和结束坐标，然后，当对象从开始状态演变为最终状态的过程中，计算和显示它。现在看一个简单的例子。假定我们需要通过把单个的点从一个位置移动到另一处而使之动画化。我们所知道的一切是点从何处开始，何处结束，以及应取的中间步数。例如，如果点在(0, 0)开始，而需要用15步把它移到(150, 150)，则必须按10个像素的增量移动该点。

下一步是要编写一个程序，它能计算和指出动画化点的中间步。计算只不过是对点的开始和结束位置做线性插值。在我们的例子中，需要画点15次，在x和y方向每次移动10个像素，因此，能执行这个任务的代码是：

```
number_of_steps = 15;
step_size = (stopx - startx) / number_of_steps;
for (j=0; j<number_of_steps; j++) {
    inc_amount = step_size * j;
    putpixel(startx + inc_amount, starty + inc_amount, WHITE);
    putpixel(startx + inc_amount, starty + inc_amount, BLACK);
}
putpixel(x + inc_amount, y + inc_amount, WHITE);
```

变量`step_size`确定点在每一状态之间移动的距离，它依赖于所要求的中间状态的数目（在这个例子中是15）和总的间隔距离。`for`循环通过计算`inc_amount`步进通过动画的每一中间步。它确定像素该移到何处。通过调用`putpixel()`给出该像素，然后通过另一个调用`putpixel()`擦去它。在这个例子中，x和y方向等值地增加，但我们可以很容易修改此代码，使得x和y方向按不同的增量改变。

这个例子告诉我们如何横跨屏幕移动一个点，但怎样才能动画化一个对象呢？这种转移是简单的，如果我们有一个画成一系列线段的对象，则所需做的一切只是在每一线段的端点运行间隔化算法，并在过程的每一步画出点间的线就可以了。下一节将使用这种技术把对象动画化。

7.1.1 把一条线动画化

在把用线段绘制的对象动画化之前，我们需要先考查一下画线的方法。”为了把一个对象动画化，必须画出此对象。从屏幕中抹去它，在它的新位置上画它，擦它，在下一个位置上画它，如此等等。然而，如果我们通过用背景颜色重画来从屏幕中抹去线段，那么，如果在动画对象经过的地方有其它对象，则可能很快就变成一片混乱。显然，这不是我们所希望出现的。

在第四章，我们学习过用putimage()函数的XOR_PUT选项可以“清晰地”移动一个对象。虽然我们打算使用putimage去动画化一条线段，但仍然能用同样的XOR_PUT选项“清晰地”移动横跨屏幕的线。

BGI允许我们设置画线函数，以便它们能画“异或”线。由于“异或”操作，我们可以简单地再画一次而抹去任一线段。为了打开线绘制程序的“异或”功能，必须使用setwritemode()函数，它的函数原型是：

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

mode变元可以置为copy_put或XOR_PUT。根据缺省值，BGI使用copy_put，这意味着可通过沿线段设置，每一像素为当前绘图颜色而画出这些线。如果使用XOR_PUT选项，则线段中的每一像素与屏幕映像“异或”。因此，如果当使用XOR_PUT选项时，对同一坐标调用两次线例程，则该线画出后又擦去，对原来的屏幕没有影响。

请注意，setwritemode函数只对line()，linerel()，lineto()，rectangle()和drawpoly()起作用。

7.1.2 使用间隔化技术

现在，让我们应用setwritemode()实现间隔化，为的是能动画化一组线段。例如，下面所示的代码，将如图7.1所示在屏幕上移动一个矩阵。

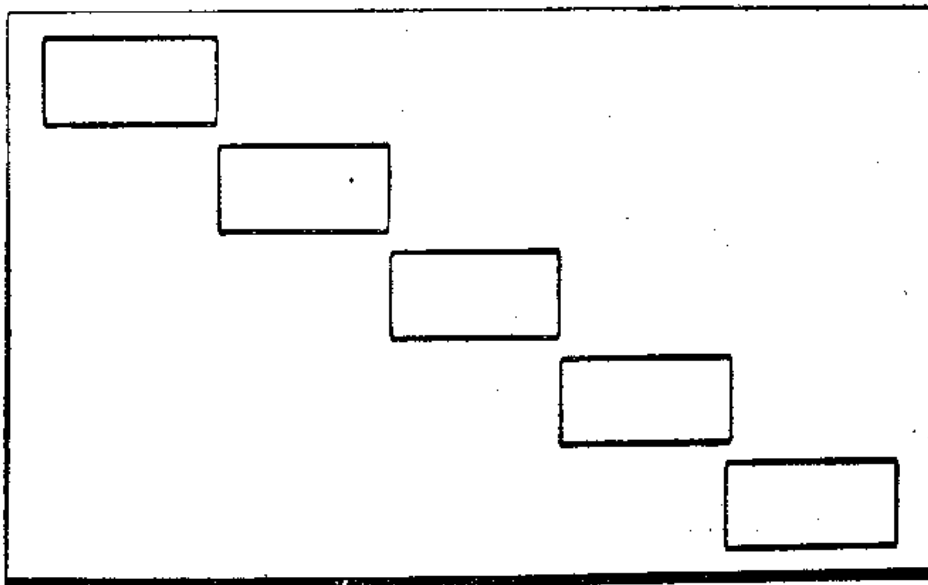


图 7.1 inbtween.cpp的时间间隔输出

矩形的开始坐标在整型数组startpoints中定义，其中数组包含一系列的x和y坐标。矩形的最终位置存放在数组stoppoints中。由于使用drawpoly()画线，因此说明该数组以便它们的第一个和最后一个坐标是一样的。还说明一个附加的数组(points)，它的大小如同startpoints和stoppoints，以便它能保存和显示动画化矩形的每一中间状态。常数NUM_STEPS定义在动画中所用的中间步数。在本例中，NUM_STEPS被置为整数值100。除了使用这些变量之外，间隔化函数类似于前些时候所提供的代码。完整的程序是：

```
// inbetween.cpp -- A simple program that demonstrates inbetweening
#include <graphics.h>
#include <conio.h>
#include <dos.h>

const int NUM_STEPS = 100; // Number of inbetweening steps
const int LENGTH = 5*2;   // Length of points array

int startpoints[LENGTH] = {0,0, 100,0, 100,20, 0,20, 0,0};
int stoppoints[LENGTH] = {400,100, 500,100, 500,120, 400,120, 400,100};
int points[LENGTH];       // Contains points to be drawn

// These statements will change a square into a triangle. Uncomment these
// lines to generate Figure 6.2.
// int startpoints[LENGTH] = {0,0, 639,0, 639,199, 0,199, 0,0};
// int stoppoints[LENGTH] = {210,120, 315,60, 420,120, 210,120, 210,120};

void inbetween(void);      // Prototype for the inbetween function

main()
{
    int gmode, gdriver=DETECT;

    initgraph(&gdriver,&gmode,"\\tc\\bgi");
    setwriteMode(XOR_PUT);
    inbetween();
    getch();
    closegraph();
    return(0);
}

// This function does the inbetweening
void inbetween(void)
{
    float step_size, inc_amount;
    int i, j;

    step_size = 1.0 / (NUM_STEPS - 1);
    for (i=0; i<NUM_STEPS; i++) {
        inc_amount = i * step_size;
        for (j=0; j<LENGTH; j++) {
            points[j] = startpoints[j] + inc_amount *
                (stoppoints[j] - startpoints[j]);
            points[j+1] = startpoints[j+1] + inc_amount *
                (stoppoints[j+1] - startpoints[j+1]);
        }
        drawpoly(LENGTH/2,points);    // Draw lines
        delay(100);                    // Wait before erasing to lessen flicker

        drawpoly(LENGTH/2,points);    // Erase lines
    }
    drawpoly(LENGTH/2,points);        // Redraw the final lines
}
```

一个有关间隔化的令人迷惑的事情是正在动画化的对象的最初和最后映像不一定是同一的。图7.2示出修改后的先前代码，以便使矩形转化为三角形。这里唯一的限制是开始和最后状态的线段数目必须相同。实际上，我们可以打破这个规则，通过在动画化对象的最后版本中添加各种图形来产生有效的效果。例如，作为练习，你可以试试把正方形转化为一个脸形。当然，这里的困难出自可绘制各种形状的适当坐标。

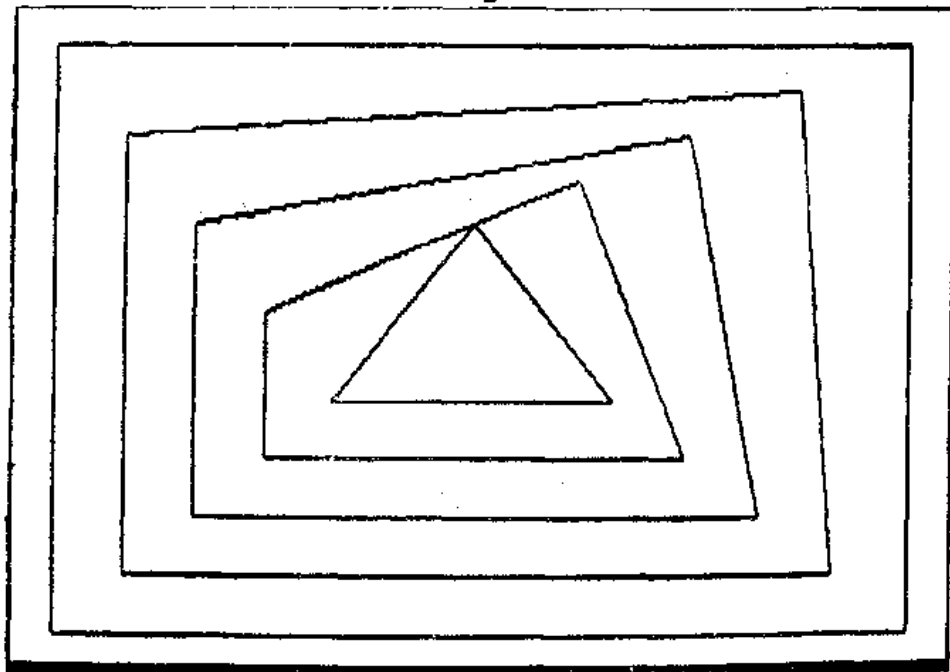


图 7.2 用间隔化把矩形转化为三角形

下面是两个针对startpoints和stoppoints的数组说明，你可以用以替代前面例子中所示的那些。这两个数组中的坐标将产生一幅图画。它如先前所描述的那样把一个矩形转化为一个三角形。由于在初始和结束状态的线段数目必须相同，而显然矩形和三角形有不同数目的顶点，故我们说明一个stoppoints，以便第一个和最后的坐标被说明三次。换句话说，在矩形中的线段之一收缩成长度为0的线，因此，虽然数组中仍然具有同样数目的线段，但三角形看起来少了一根线。

```
int startpoints[LENGTH] = {0.0, 639.0, 639.199, 0.199, 0.0};  
int stoppoints[LENGTH] = {210, 120, 315.60, 420.120, 210, 120, 210, 120};
```

7.1.3 使用getimage() 和putimage()

使用间隔化的缺点之一是动画化的对象必须每步沿路重画一次。对于复杂的对象，这个过程可能很慢。另一种方法是绘制该映像一次，然后用getimage() 和putimage() 函数把它在屏幕上移动。

第三章已介绍过getimage() 和putimage() 函数，并描述了各种对于 putimage() 可用的布置选项。在本节，我们将注重如何使用它们，而不是描述它们如何运行。（参见第三章有关这些例程的详细说明和它们怎样工作。）

我们使用getimage() 和putimage() 的第一个动画化例子，是如图7.3所示在屏幕上移动一辆自行车。这个例子的代码是，


```

// bike.cpp -- This program demonstrates animation using the BGI's
// getimage() and putimage() functions. The program draws a bike on the
// screen and then copies the image of the bike across the screen using the
// XOR_PUT option of putimage().
#include <stdlib.h>
#include <stdio.h>
#include <graphics.h>
#include <alloc.h>
#include <conio.h>
#include <dos.h>

const int STEP = 5;           // Number of pixels to move bike each time

void *bike;                   // Points to the image of the bike

void drawbike(void);           // Function prototypes
void movebike(void);

main()
{
    int gdriver = CGA;         // Use CGA high-resolution mode or
    int gmode = CGAHI;        // comparable 640 x 200 modes

    initgraph(&gdriver,&gmode,"\\tc\\bgi");
    drawbike();
    movebike();
    getch();
    closegraph();
    return(0);
}

// Draw a bicycle using calls to circle() and line(). Then capture
// the image of the bicycle with getimage().
void drawbike(void)
{
    circle(50,100,25);         // Draw the wheels
    circle(150,100,25);
    line(50,100,80,85);        // Draw part of the frame
    line(80,85,134,85);
    line(77,82,95,100);
    line(130,80,150,100);
    line(128,80,113,82);       // Handlebars
    line(128,80,116,78);
    line(72,81,89,81);         // Seat
    line(73,82,90,82);

    circle(95,100,5);          // Draw the pedals
    line(92,105,98,95);
    line(91,105,93,105);
    line(97,95,99,95);
    line(95,100,136,87);
    bike = malloc(imagesize(25,75,175,125)); // Allocate image
    if (bike == NULL) {
        closegraph();
        printf("\nFailed to allocate memory for bike image.\n");
        exit(1);
    }
    getimage(25,75,175,125,bike); // Capture image of bike
}

```

```
// Move bike across the screen in increments of STEP
void movebike(void)
{
    int i;

    for (i=0; i<getmaxx()-180; i+=STEP) {
        putimage(25+i,75,bike,XOR_PUT);        // Erase bike
        putimage(25+STEP+i,75,bike,XOR_PUT);    // Display bike
        delay(50);
    }
}
```

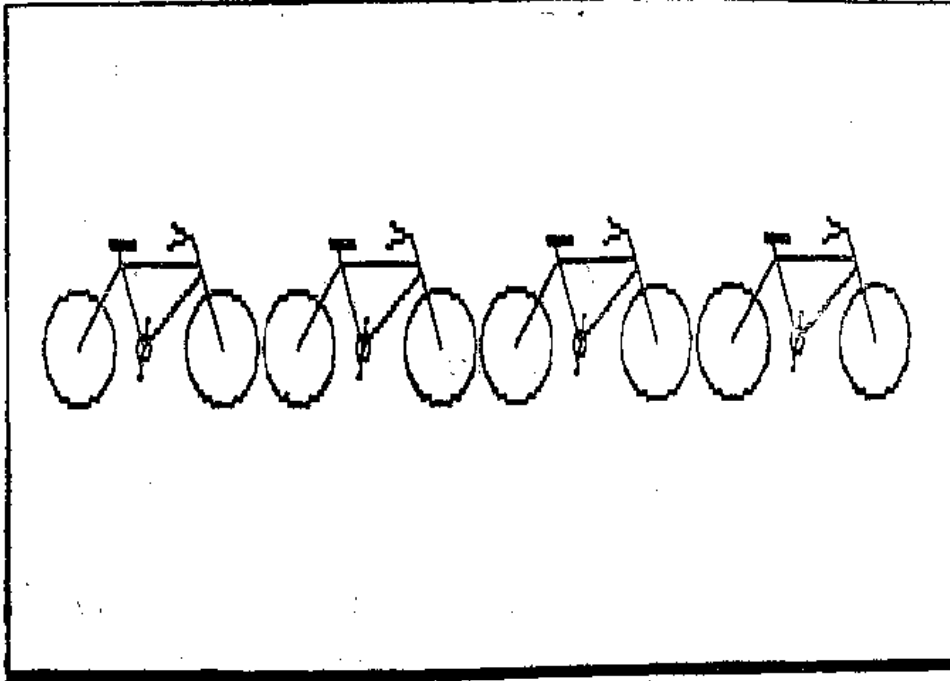


图 7.3 bike.cpp的时间间隔输出

通过在函数drawbike()中对line()和circle()的一系列调用,创建自行车的初始映像。往下,通过调用malloc()把存储分配给指针bike,以保存自行车的屏幕映像。请注意,要对指针bike作测试,并测试malloc()返回的指针是否为0。如果是,则表示存储分配失败,并退出程序,否则,通过调用getimage()检索自行车的映像。

```
bike = malloc(imagesize(25,75,175,125)); // Allocate image
if (bike == NULL) {
    closegraph();
    printf("\nFailed to allocate memory for bike image.\n");
    exit(1);
}
getimage(25,75,175,125,bike);
```

往下,调用movebike()函数,以便在屏幕上移动bike中的自行车映像。在本例中,该映像以5个像素的步从坐标(25, 75)移动到(getmaxx-180, 75)。这个运动通过movebike()中的for循环完成。它们是:

```
for (i=0; i<getmaxx()-180; i+=STEP) {
    putimage(25+i,75,bike,XOR_PUT);        // Erase bike
    putimage(25+STEP+i,75,bike,XOR_PUT);    // Display bike
    delay(50);
}
```

如你所看到的那样，for循环调用putimage()两次。第一次通过把bike“异或”在屏幕上擦去自行车映像的当前映像。第二次用XOR_PUT选项，在它的下一位置显示自行车的映像。

我们可以通过在自行车移动时旋转自行车的脚蹬改进我们的例子。基本思想是创建若干自行车映像，其中每一映像具有不同方向的一些脚蹬，然后顺序展示它们使之看起来就像脚蹬在旋转。有两种方法完成这点，例如，我们可以作出带有不同方向脚蹬的整个自行车的若干映像，然后顺序展示它们。或者，我们有一个完整的自行车映像以及一系列示出脚蹬处于不同位置的较小映像。用第二种技术，脚蹬映像必须和自行车映像重叠以建立运动。

你可能感到惊奇：为什么需要两组映像，其一针对自行车，而另一为了动画化脚蹬？一个重要的原因是(这里并不明显)，对于大型的图画，我们可加速动画的过程并通过只交换必须改变的区域映像而节省存储。此外，若干整辆自行车的映像比单个自行车映像和伴随的一组较小映像耗费更多存储。例如，在高分辨的图形模式下，它很容易耗费超额的存储。

现在，让我们回到动画化自行车脚蹬的任务，我们看到的下一个程序将动画化一辆自行车，它在屏幕上移动自行车，并旋转它的脚蹬。程序是这样完成的，它创建4个自行车的不同映像，每一都有不同的脚蹬方向，并顺序展示这些映像：

```
// bike2.cpp -- This program demonstrates animation using the BGI's
// getimage() and putimage(). The program creates four images of a
// bike--each with the pedals in a different orientation--so that
// when the images are played back it looks like the pedals are
// moving as the bicycle is moved across the screen.
#include <stdlib.h>
#include <stdio.h>
#include <graphics.h>
#include <alloc.h>
#include <conio.h>
#include <dos.h>

const int STEP = 5;           // Number of pixels to move bike each time

int S=5;
void *bike1, *bike2;          // Points to the image of the bike
void *bike3, *bike4;

void drawbike(void);           // Function prototypes
void movebikeandpedals(void);

main()
{
    int gdriver = CGA;         // Use CGA high-resolution mode or
    int gmode = CGAHI;         // comparable 640 x 200 modes

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    drawbike();
    movebikeandpedals();
    getch();
    closegraph();
    return(0);
}

// Draw a bicycle using calls to circle() and line(). Then capture
// the image of the bicycle with getimage().
void drawbike(void)
{
```

```

void *pedals;

circle(50,100,25); // Draw the wheels
circle(150,100,25);
line(50,100,80,85); // Draw part of the frame
line(80,85,134,85);
line(77,82,95,100);
line(130,80,150,100);
line(128,80,113,82); // Handlebars
line(128,80,116,78);
line(72,81,89,81); // Seat
line(73,82,90,82);
circle(50,100,25); // Draw wheels
circle(150,100,25);
line(50,100,80,85); // Draw part of frame
line(80,85,134,85);
line(77,82,95,100);
line(130,80,150,100);
line(128,80,113,82); // Handlebars
line(128,80,116,78);
line(72,81,89,81); // Seat
line(73,82,90,82);
bike1 = malloc(imagesize(25,75,175,125));
bike2 = malloc(imagesize(25,75,175,125));
bike3 = malloc(imagesize(25,75,175,125));
bike4 = malloc(imagesize(25,75,175,125));
pedals = malloc(imagesize(85,90,110,110));
if (bike1 == NULL || bike2 == NULL || bike3 == NULL ||
    bike4 == NULL || pedals == NULL) {
    closegraph();
    printf("\nFailed to allocate memory for bike images.\n");
    exit(1);
}
circle(95,100,5); // Draw base of pedals
line(95,100,136,87);
getimage(85,90,110,110,pedals); // Save screen of pedals
line(86,100,104,100); // Draw pedals in
line(85,100,87,100); // first position
line(103,100,105,100);
getimage(25,75,175,125,bike1);
putimage(85,90,pedals,COPY_PUT); // Restore pedal area
line(88,96,102,104); // Draw pedals in second
line(87,96,89,96); // position
line(101,104,103,104);
getimage(25,75,175,125,bike2);

putimage(85,90,pedals,COPY_PUT); // Restore pedal area
line(95,95,95,105); // Draw pedals in third
line(94,95,96,95); // position
line(94,105,96,105);
getimage(25,75,175,125,bike3);

putimage(85,90,pedals,COPY_PUT); // Restore pedal area
line(102,96,88,104); // Draw pedals in fourth
line(101,96,103,96); // position
line(87,104,89,104);
getimage(25,75,175,125,bike4);
free(pedals);
}

// Sequences through the four images of the bicycle in order to move
// the bicycle across the screen.

```

```

void movebikeandpedals(void)
{
    int i, times;

    times = (getmaxx()-175) / STEP;
    for (i=0; i<times; i++) {
        switch(i%4) {
            case 0 : putimage(25+i*STEP,75,bike4,XOR_PUT);
                     putimage(25+(i+1)*STEP,75,bike1,XOR_PUT);
                     break;
            case 1 : putimage(25+i*STEP,75,bike1,XOR_PUT);
                     putimage(25+(i+1)*STEP,75,bike2,XOR_PUT);
                     break;
            case 2 : putimage(25+i*STEP,75,bike2,XOR_PUT);
                     putimage(25+(i+1)*STEP,75,bike3,XOR_PUT);
                     break;
            case 3 : putimage(25+i*STEP,75,bike3,XOR_PUT);
                     putimage(25+(i+1)*STEP,75,bike4,XOR_PUT);
                     break;
        }
        delay(50); // This delay controls the speed of the bicycle
    }
}

```

7.2 在背景上动画化对象

迄今为止，我们只是用getimage()和putimage()在空白的背景上移动对象。遗憾的是，这不能代表典型的情况。在大多数动画程序中，你将大概需要在背景上放若干对象使屏幕更为生动。例如，我们可使用一个田园风景作为自行车动画程序的背景。但是，如果我们把对象放在自行车必须经过的屏幕上，在两个对象重叠时，自行车可能会改变它的颜色。可能的颜色变化的原因是动画化的对象和屏幕进行了“异或”。因此，当正被动画化的对象包含一个位值1时，如果屏幕在该位置上也有一个位值1，则把映像拷贝到屏幕将会改变其相应的像素颜色。幸好，有一个解决此问题的方法，即对此动画化的对象使用两个稍微不同的映像。这两个特殊的掩码将被组合，以便穿越背景时动画化的对象不改变颜色。

该方法使用一个和屏幕“AND”的掩码，而第二个则是和第一个掩码“XOR”的掩码。当这两个掩码按此方式组合时，对象将以它正常的颜色出现在屏幕中。为抹去对象，我们将在显示它的区域下保留屏幕映像，以便在稍后能通过把保存的屏幕拷贝回显示器而恢复该屏幕。表7.1示出如何组合AND掩码和XOR掩码中的位，以选择屏幕上特定的颜色。通过选出每一掩码中适当的值，你可以置每一屏幕位为0或1，保持它为同一值，或逆反它。

遗憾的是，这种动画技术可能很慢，因为有若干屏幕映像必须由我们处理。这可能是一个问题。首先，我们必须保存该屏幕。然后AND一个映像，接着XOR一个映像，而稍

表 7.1 使用AND和XOR掩码

AND掩码值	XOR掩码值	所得的屏幕位
0	0	0
0	1	1
1	0	不变
1	1	逆反

后再通过拷贝回保存的屏幕映像而恢复屏幕为原来状态。这可能消耗大量时间——尤其是当被动化的对象尺寸很大时。事实上，我们已经绘制的自行车映像就大到难以用这个方法有效地动画化。

因此，我们看到的下一个程序将仅在屏幕上移动一个小球，它有一个如图7.4所示的背景。该球用两个如前面描述的掩码绘制在屏幕上。实际上，球是由两个小圆填以不同颜色而做成的，这将向你展示出双掩码方法如何工作的好思路。

图7.5示出两个组合以形成球的掩码。在该程序中，这两个掩码是通过指针andmask和xormask存取的，第三个指针covered用于保存在任意给定时刻被掩码覆盖的屏幕部分。因此，在屏幕上移动圆的进程浓缩在四行的函数moveball()中：

```
putimage(35+i,85,covered,COPY_PUT);  
getimage(35+STEP+i,85,65+STEP+i,115,covered);  
putimage(35+STEP+i,85,andmask,AND_PUT);  
putimage(35+STEP+i,85,xormask,XOR_PUT);
```

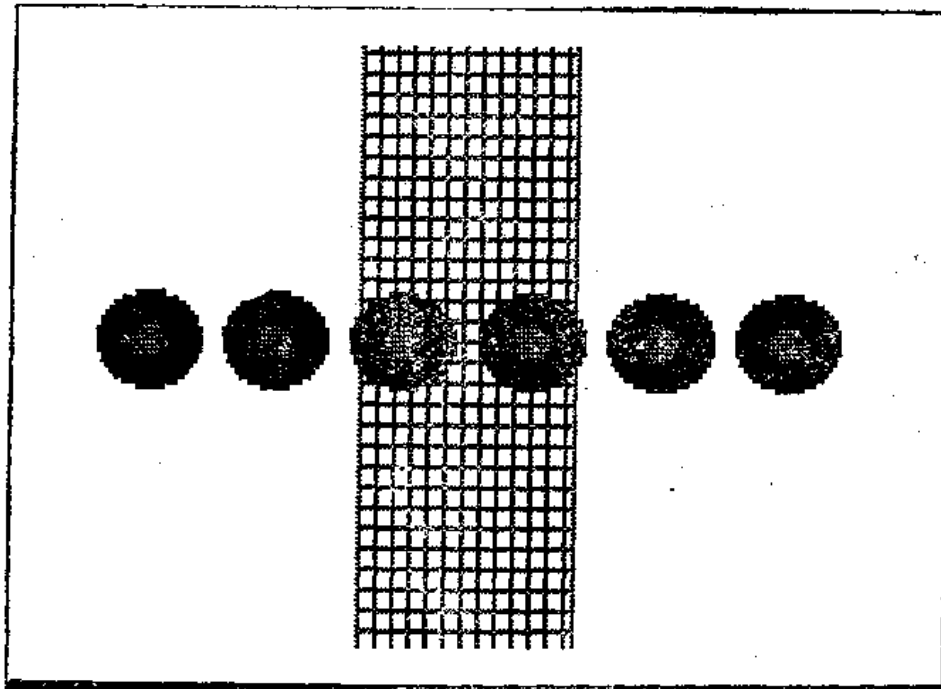


图 7.4 ball.cpp的时间间隔输出

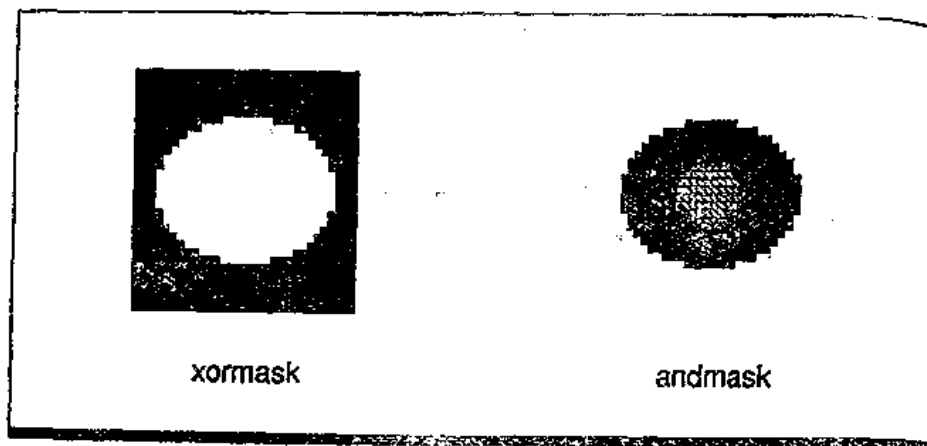


图 7.5 ball.cpp中使用的掩码

第一个语句重写球的当前位置。下一行保存球将下一次出现之处的屏幕。最后两个语句把两个掩码写到屏幕并画球。

```
// ball.cpp -- This program demonstrates animation using an
// AND mask and an XOR mask.
#include <graphics.h>
#include <conio.h>
#include <alloc.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>

const int STEP = 2;
const int DELAY = 50;

void *xormask, *andmask, *covered;

void drawball(void);           // Function prototype
void moveball(void);

main()
{
    int gmode, gdriver = DETECT;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    drawball();
    moveball();
    getch();
    closegraph();
    return(0);
}

void drawball(void)
{
    covered = malloc(imagesize(35,85,65,115));
    xormask = malloc(imagesize(35,85,65,115));
    andmask = malloc(imagesize(35,85,65,115));
    if (covered == NULL || xormask == NULL || andmask == NULL) {
        closegraph();
        printf("Not enough memory\n");
        exit(1);
    }
    getimage(35,85,65,115,covered);
    // Create the AND mask first
    setfillstyle(SOLID_FILL, getmaxcolor());
    bar(35,85,65,115);
    setcolor(0);
    setfillstyle(SOLID_FILL, 0);
    pieslice(50,100,0,360,12);
    getimage(35,85,65,115,andmask);
    // Create the XOR mask
    putimage(35,85,covered,COPY_PUT);
    setcolor(getmaxcolor());
    setfillstyle(SOLID_FILL, getmaxcolor());
    pieslice(50,100,0,360,12);
    setfillstyle(SOLID_FILL, 1);
    setcolor(1);
    circle(50,100,4);
    floodfill(50,100,1);
    getimage(35,85,65,115,xormask);
    // Erase ball and make the backdrop
```

```

putimage(35,85,covered,COPY_PUT);
setfillstyle(HATCH_FILL,2);
bar3d(100,10,150,199,0,0);

// Redraw the ball on the screen by saving the screen area to
// be overwritten, using the AND mask and then the XOR mask.
getimage(35,85,65,115,covered);
putimage(35,85,andmask,AND_PUT);
putimage(35,85,xormask,XOR_PUT);
}

// Move the ball across the screen by first overwriting the currently
// displayed ball and then moving it to the next location by using
// the AND and XOR masks.
void moveball(void)
{
    int i;

    for (i=0; i<150; i+=STEP) {
        putimage(35+i,85,covered, COPY_PUT);
        getimage(35+STEP+i,85,65+STEP+i,115,covered);
        putimage(35+STEP+i,85,andmask,AND_PUT);
        putimage(35+STEP+i,85,xormask,XOR_PUT);
        delay(DELAY);
    }
}

```

7.2.1 动画化多个对象

可以很容易扩展我们的程序以便它将一次动画化多于一个的对象。我们对此程序所作的主要的改动是向屏幕提供多于一个要移动的映像或映像序列。然后，通过顺序展示动画一系列对象和映像，很容易创建一个带有多个移动对象的景色。

7.2.2 getimage() 和 putimage() 的限制

使用getimage() 和putimage() 动画化对象有若干限制，这也许会约束到它们的使用。首先，它们可能使用许多存储——特别是当映像的大小增加时，此外，当保存或写的映像变得较大时，getimage() 和putimage() 的操作速度将衰减。这些都是getimage() 和putimage() 的共同问题。

但是，当处理动画时，使用这两个函数的最大缺点是它们不允许位图映像变比或旋转。你可以自己编写函数来执行这些操作，但BGI使用的位图对许多模式是不同的。因此，你应编写各种函数以处理每一种模式。这是一项枯燥的任务，并且是我们不想在这里着手解决的。尽管有这些问题，但getimage() 和putimage() 提供了强有力的然而简单的在图形屏幕上移动对象的方法。

7.3 用调色板动画化

通常是这样制造动画的，即当对象移过屏幕时绘制它或拷贝对象的映像并把它移过屏幕。但是，另一种有时十分生动和简单的方法是使用调色板产生动画。其基本思想是使用不同的颜色在屏幕上绘制对象。然后改变调色板中的颜色。当完成这点时，屏幕上的所

有对象立即改变它们的颜色，并且就像所有的对象都已在新位置上重画，通过把对象序列化，使得它们的颜色反映一系列的彩色变换，就可以制造出动画。

遗憾的是，并非所有图形适配器都同样支持调色板处理。事实上，EGA和VGA最有力地支持这种特色。用这两种图形适配器，你可以使用`setpalette()`函数去改变调色板中的颜色。只有某些图形适配器允许你改变调色板。如早些时候所提及的那样，通过转换调色板中的颜色，可以使得对象似乎在移动。

使用彩色调色板的一个普通的动画例子是显示下雨的山景。实际上，雨水的运动是由一系列调色板的改变而导出的。由于需要大量的时间和编程来创造一幅美丽的山景，让我们使用一个较简单的编程例子。

我们将观察的程序是`firework.cpp`。它连续在屏幕上显示一系列类烟火彩色的燃烧。在存储中保存了三种随机像素的映像，使用`putimage()`急速地绘制在屏幕上，以形成扩张的爆炸图案。程序中的技巧是使用调色板中的变化，为的是取得各式各样的燃烧彩色，这使我们不用保留许多不同的燃烧图案（每一图案带不同颜色）。

该程序被编写成可运行在EGA或VGA系统。在CGA方式下这个程序不工作，因为我们将要改变调色板，而这种技术不能完全按EGA和VGA所执行的同样方式工作。

现在，让我们仔细看看这片代码。烟火的燃烧图案由屏幕上随机地点出的像素生成。这是通过for循环完成的：

```
for (i=0; i<BLAST_SIZE*3/2; i++)  
    putpixel(random(BLAST_SIZE),random(yrange),1);
```

该循环在以(0,0)和(BLAST_SIZE, yrange)为界的区域中绘出一些彩色像素的点。这个区域的右下坐标依赖于BLAST_SIZE。在这种情况下，它是300像素宽。变量yrange就是BLAST_SIZE，它根据屏幕的长宽比缩小或放大。在这个for循环完成之后，屏幕右上的方块区域将用随机置放的像素填充。当然，我们需要一个圆形的燃烧的图案——不是一个方块的东西。因此，使用下面的代码抽出圆形的像素。

```
circle(BLAST_SIZE/2,yrange/2,BLAST_SIZE/2-10);  
rectangle(0,0,BLAST_SIZE,yrange);  
floodfill(1,1,1);  
setfillstyle(SOLID_FILL,0);  
floodfill(1,1,0);  
getimage(0,0,BLAST_SIZE,yrange,blast3);
```

这些语句使用`floodfill()`函数擦去圆形（绘制成所要求的燃烧图案的大小）和包含像素方块的四方形的所有像素。第一个语句绘制一个将定义燃烧图案大小的圆。往下，用`rectangle()`绘制有界的矩形。然后，用`floodfill()`以白色填充圆和矩形之间的区域。请注意，不使用黑色，因为已经包含它，因此在遇到少量背景像素时，`floodfill()`将认为已经填充了它。继续到下一语句，代码再一次调用`floodfill()`以重填圆和矩形之间的区域。但是，这次该区域用黑色填充。现在屏幕上剩下的唯一东西是圆形的燃烧图案。因此，最后一步是调用`getimage()`以获得燃烧图案的映像，这是通过前面所示的最后一个语句完成的。执行类似的过程以便生成两个小一点的燃烧图案，它们存放在`blast2`和`blast1`。稍后，通过顺序展示这些映像，将展现出不断扩张的燃烧图案。

当然，好的烟火显示要是没有火箭去发射是不够完美的。这样的火箭映像由以下若干

行建立。用`bar()`、`putpixel()`和两次调用`line()`来画火箭。火箭的映像保存到称为`rocket`的指针中。

跟随的`while`循环用于连续地绘制一系列烟火爆炸，直到按下某一键为止。爆炸的位置是用以下的行随机计算的：

```
x = random(getmaxx()-BLAST_SIZE);
y = random(getmaxy()/3);
```

(`x`, `y`) 坐标对标记出将用以创造烟火效果的燃烧图案映像的左上角。`x`值保持在0和`getmaxx() - BLAST_SIZE`之间，以便燃烧的映像不超出屏幕的边缘而被裁剪。`y`坐标限制在最大`y`范围（你稍后将可看到，它相应于屏幕的顶部）的1/3。

火箭由先前列出的两个语句之后的`for`循环发射。计算它的起始位置和结束位置以便火箭将飞行到写在屏幕上的燃烧图案的中部。在`for`循环之前对`setpalette()`的调用是为了确保火箭用白色绘制。

`for`循环以后的语句显示`blast1`、`blast2`和`blast3`中的燃烧图案。燃烧映像在屏幕中被显示，十分像过去已讨论过的那样，所以我们不再详细讨论它们。但是，代码的独特部分，是对`setpalette()`函数的调用。这些部分是用于改变烟火的彩色。例如，下面这一行：

```
setpalette(getmaxcolor(),random(15)+1);
```

随机地把调色板的表中的最后一项改变为当前视频模式的16种可能颜色之一（参见第三章这样的表），这是正用于给烟火上彩色的颜色项。为了提供少量的变化，设计了第二个燃烧图案`blast2`的颜色，使彩色可能改变。这依赖于跟随的`if`语句中的`random()`语句是否返回一个0值（请记住，`random()`返回0和`n-1`之间的值）。

```
if (!random(3))
    setpalette(getmaxcolor(),random(15)+1);
```

类似地，围绕用于绘制`blast3`的`putimage()`函数的`if`语句是用来在绘图时作随机限制的。通过执行这个语句，程序将生成两个不同大小的烟火图案。

你可能已经注意到，在`firework.cpp`中有一些对Turbo C++ `delay()`函数的调用。这些语句用于给烟火程序一种艺术表现力，所以烟火图案不会太快地消逝，因为不同的机器显示烟火图案的速度不同。你可能需要基于正在使用的计算机调整延迟。

程序的其余部分涉及称为`covered`的指针，它用于保存生成或显示某些烟火图案之前的屏幕。它可用于从屏幕抹去烟火并恢复原来状态。

```
// firework.cpp -- This program simulates a fireworks display by
// using the getimage()/putimage() functions as well as palette
// animation.
#include <graphics.h>
#include <stdlib.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include <stdio.h>

const int BLAST_SIZE = 200;           // Size of fireworks in pixels
const int ROCKET_HEIGHT = 7;          // Rocket is 7 pixels tall
const int ROCKET_WIDTH = 4;           // and 4 pixels wide
const int ROCKET_STEP = 2;            // Rocket moves two pixels at a time
```

```

main()
{
    int gdriver = EGA;           // Use EGA and EGALo or VGA and VGALo
    int gmode = EGALo;          // with this version of the fireworks
    int x, y, i, xasp, yasp;    // program
    int yrange, ry;
    double aspectratio;
    void *covered;              // Holds screen under fire burst
    void *blast1, *blast2, *blast3; // Three burst patterns
    void *rocket;               // The fire rocket

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    if (gdriver != EGA && gdriver != VGA) {
        closegraph();
        printf("This program requires an EGA or VGA graphics adapter.\n");
        exit(1);
    }
    randomize();
    getaspectratio(&xasp, &yasp);
    aspectratio = (double)xasp / (double)yasp;
    yrange = BLAST_SIZE * aspectratio;
    blast1 = malloc(imagesize(0,0,BLAST_SIZE,yrange));
    blast2 = malloc(imagesize(0,0,BLAST_SIZE,yrange));
    blast3 = malloc(imagesize(0,0,BLAST_SIZE,yrange));
    covered = malloc(imagesize(0,0,BLAST_SIZE,yrange));
    if (covered == NULL || blast3 == NULL ||
        blast2 == NULL || blast1 == NULL) {

        closegraph();
        printf("Not enough memory\n");
        exit(1);
    }
    getimage(0,0,BLAST_SIZE,yrange,covered);
    // Make the burst pattern by randomly filling a square with pixels
    for (i=0; i<BLAST_SIZE*3/2; i++)
        putpixel(random(BLAST_SIZE), random(yrange), getmaxcolor());

    // Then convert the square pixel pattern into a circle by erasing
    // all of the pixels outside of a circle. Use a floodfill to perform
    // this operation. Repeat this process three times, each with a
    // smaller circle and after each time capture the circular burst
    // pattern in burst1, burst2, or burst3.

    circle(BLAST_SIZE/2,yrange/2,BLAST_SIZE/2-10);
    rectangle(0,0,BLAST_SIZE,yrange);
    floodfill(1,1,getmaxcolor());
    setfillstyle(SOLID_FILL,0);
    floodfill(1,1,0);
    getimage(0,0,BLAST_SIZE,yrange,blast3); // Capture large burst pattern
    setfillstyle(SOLID_FILL,getmaxcolor()); // Make smaller pattern
    circle(BLAST_SIZE/2,yrange/2,BLAST_SIZE/4+10);
    rectangle(0,0,BLAST_SIZE,yrange);
    floodfill(1,1,getmaxcolor());
    setfillstyle(SOLID_FILL,0);
    floodfill(1,1,0);
    getimage(0,0,BLAST_SIZE,yrange,blast2); // Capture middle burst pattern
    setfillstyle(SOLID_FILL,getmaxcolor()); // Make smallest burst
    circle(BLAST_SIZE/2,yrange/2,20);
    rectangle(0,0,BLAST_SIZE,yrange);
    floodfill(1,1,getmaxcolor());
    setfillstyle(SOLID_FILL,0);
    floodfill(1,1,0);

```

```

getimage(0,0,BLAST_SIZE,yrange,blast1); // Get small burst
putimage(0,0,covered,COPY_PUT); // Erase burst pattern

// Create the rocket
rocket = malloc(imagesize(0,0,ROCKET_WIDTH,ROCKET_HEIGHT));
setfillstyle(SOLID_FILL,getmaxcolor());
bar(1,2,ROCKET_WIDTH-1,ROCKET_HEIGHT-1);
putpixel(2,1,getmaxcolor());
line(0,ROCKET_HEIGHT-2,0,ROCKET_HEIGHT);
line(ROCKET_WIDTH,ROCKET_HEIGHT-2,ROCKET_WIDTH,ROCKET_HEIGHT);
getimage(0,0,ROCKET_WIDTH,ROCKET_HEIGHT,rocket); // Rocket
putimage(0,0,rocket,XOR_PUT); // Erase rocket image

// Draw fireworks until a key is pressed
while (!kbhit()) {
    // Randomly decide where the rocket should explode. Use the
    // upper half of the screen and avoid the edges of the screen.
    // (x, y) corresponds to the top-left corner of the blast
    // images.
    x = random(getmaxx()-BLAST_SIZE);
    y = random(getmaxy()/3);
    setpalette(getmaxcolor(),15); // Make rocket WHITE
    for (ry=getmaxy()-ROCKET_HEIGHT; ry>y+yrange/2;
        ry -= ROCKET_STEP) {
        putimage(x+BLAST_SIZE/2,ry,rocket,XOR_PUT);
        delay(20);
        putimage(x+BLAST_SIZE/2,ry,rocket,XOR_PUT);
    }
    setpalette(getmaxcolor(),random(15)+1); // Randomly select burst color
    getimage(x,y,x+BLAST_SIZE,y+yrange,covered); // Save screen
    putimage(x,y,blast1,COPY_PUT); // Draw smallest burst
    delay(100); // Keep it on screen for awhile

    putimage(x,y,blast2,COPY_PUT); // Show second burst
    if (!random(3))
        setpalette(getmaxcolor(),random(15)+1); // Maybe change color
    delay(100);

    if (random(2)) { // Randomly decide to
        putimage(x,y,blast3,COPY_PUT); // show third burst
        if (!random(3)) setpalette(getmaxcolor(),random(15)+1);
    }
    delay(400+random(750)); // Wait for a while
    putimage(x,y,covered,COPY_PUT); // Erase burst
    delay(random(3000)); // Wait for next rocket
}
closegraph();
return(0);
}

```

在调色板动画中的小小的变异是用它使对象在屏幕上瞬间出现。例如，我们可以把闪电添加到第二章中的fractal程序。它将瞬间出现在屏幕上。然而，它却是实实在在始终在那的。技巧是把闪电绘成和背景一样的颜色，但是，使用不同的调色板索引。然而，使闪电短时间出现，把调色板索引颜色改变为闪电颜色。然后，恢复成为背景，交换调色板的颜色产生把闪电绘制在屏幕上的效果。但由于它已存在，故节省了绘制的时间，并形成非常快的动画技术。

7.4 使用多重屏幕页

另一种能产生快速动画效果的动画技术是使用多重存储页。这种方法利用提供若干能绘图和显示的存储的独立段（或“页”）的图形硬件。但是，像调色板技巧一样，这种方法只能在某些图形适配器（Hercules, EGA和VGA）上可行。其基本思想是使得有若干屏幕的部分或完整映像准备显示并然后交换它们。通过完成这点，可创建动画。可用的页面数目依赖于已在使用的图形方式。有关它们具有的图形模式和页面的完整的表，请参见第三章。

BGI函数setactivepage() 用于选择在何处书写图形操作。setactivepage() 的函数原型是：

```
void far setactivepage(int page);
```

其中，page是要使用的页数。请注意，活动的页不应当是当前正被显示的页。事实上，可见的页是由函数setvisualpage() 选择的。它的函数原型是：

```
void far setvisualpage(int page);
```

使用多重存储页面的另一种方式是作为草稿本。这里的思想是使用不可见页面之一作为工作区，以组合掩码或创建对象的映像。它们在稍后可以用getimage() 和putimage() 快速地拷贝到可见页面上。利用这种方式，你可以避免显示每一事情。例如，在上一个程序中，我们需要为烟火建立像素图案。这在屏幕上完成。但如果其它存储页面已经可用，我们就可以通过在非可见的页面上执行它们以隐藏这些操作。

第八章 创建鼠标工具包

使用面向对象程序 (OOP) 例如 Turbo C++ 的主要好处之一是：你可以建立功能强的对象集合，然后又可以用它来建立其它功能更强的对象。你越使用这些对象工作，就越发现这种“积木块”的编程方法在帮助你创建易于重用和扩充的工具方面向前迈进了一大步。

本章有两个重要目的。第一，开发一个称为 `mouseobj` 的类库，它提供为使鼠标与图形程序相交接所需的全部基本例程。此程序包将是众多的交互图形工具的第一个，它在第八至第十一章中开发并在其后各章的几个绘图应用程序中使用。第二，指出如何使用 OOP 技术，例如，“继承性”是贯穿本书其余部分使用的重要特性。可以分阶段建立我们的程序并其后在已建立的程序基础上加以扩充。

8.1 使用鼠标

虽然键盘是极好的输入设备，但它不总是最适用于交互图形程序。甚至对于某些有用的定位特征，如光标键，键盘也是速度缓慢且缺少在屏幕上随机选择位置的能力。在交互式图形环境中典型地要求这些操作，所需的是交互指示设备，如光笔、操纵杆或鼠标。因为鼠标在 PC 机上比其它设备更普遍，故将选择它作为主要的输入设备。虽然我们试图尽可能支持键盘，但还是针对鼠标设计我们所有的输入例程。

本章将指出怎样从你自己的图形程序中访问鼠标。我们的鼠标类库 `mouseobj` 包含若干成员函数，执行诸如初始化鼠标、确定鼠标的状态和位置，控制它在程序中的移动等操作。此外，我们将开发一个称为 `kbdmouseobj` 的类库，以便在不提供鼠标时支持键盘。此类库由鼠标类库派生并包含仿真鼠标特征的成员函数。这些鼠标和键盘类库在文件 `mouse.h`、`mouse.cpp`、`kbdmouse.h` 和 `kbdmouse.cpp` 中实现（见列表 8.1, 8.2, 8.3, 8.4）。由于我们的鼠标工具包具有面向对象的特点，你将会看到我们可以很容易用键盘仿真鼠标——这通常是很难编程的任务。

8.2 鼠标综述

在我们开始为鼠标和键盘类库编码之前，将需要涉及鼠标硬件和软件如何工作的基础。如果你从未编写过控制鼠标的代码，可能会对鼠标那样容易被支持而感到惊奇。事实上，为初始化鼠标，获得它的光标位置和移动它的光标，仅要求少量的基本函数。当使用鼠标工作时，请记住，我们的代码被设计成是用兼容 Microsoft 的鼠标工作的，而这恰巧是 PC 最通用的鼠标标准。如果你正使用不同类型的鼠标，则可能需要查阅你的鼠标硬件用户手册，并对我们提出的代码进行必要的修改，以便你的鼠标能正常地操作。

鼠标系统实际上由两个主要的元素组成：鼠标机制和称作鼠标驱动程序的存储驻留程序。鼠标驱动程序提供与鼠标通信所需的全部低级支持。此外，它负责自动维持鼠标的光

标位置和发现是否按了某些按钮。

通常，鼠标驱动程序在上电时由你的autoexec.bat文件中的语句装入存储器。一旦装入驱动程序，鼠标就可以被随后执行的任意程序使用。

在应用程序中使用鼠标惊人地简单。但是，使用鼠标时在正文和图形程序之间有一些微小的差别，因此我们把注意力集中在图形模式下的鼠标。

8.3 访问鼠标驱动程序

我们通过PC软中断33h访问鼠标和鼠标驱动程序的各种功能。鼠标驱动程序为这个中断单元的各种调用提供服务，把它们转到其内部的低级鼠标函数。所选择的特定函数依赖于中断时AX寄存器中的值。三个其它的寄存器，BX、CX和DX用于把各参数传送给鼠标例程。类似地，鼠标函数使用这四个寄存器把鼠标位置和鼠标按钮的状态返回给调用函数，图8.1示出程序如何使用中断33h来调用鼠标函数。

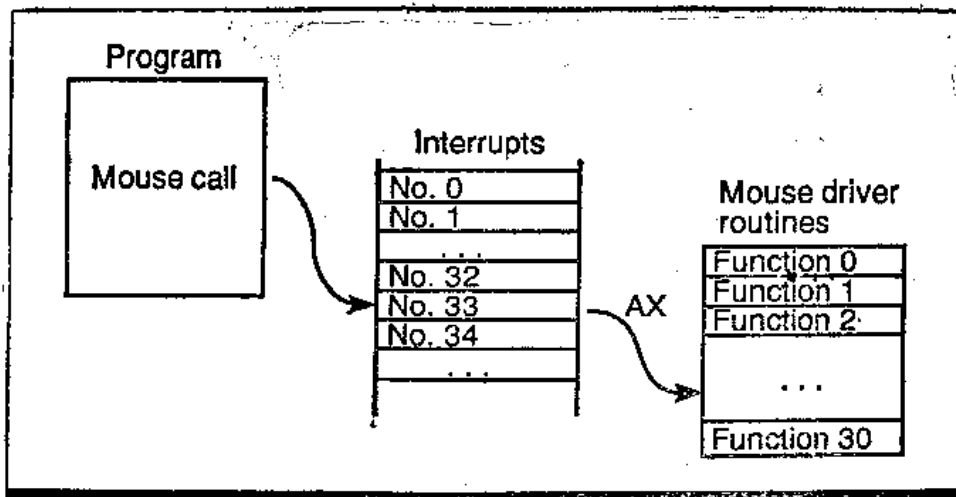


图 8.1 引用鼠标函数的次序

中断33h可以通过特定的Turbo C++函数int 86()调用，它提供直接访问微处理器的中断功能，一般形式如下：

```
int86(interrupt_number,&inregs,&outregs);
```

其中，第一个变元是要使用的中断向量，下面两个变元是对寄存器值的结构的指针。此结构的说明在bios.h标题文件中定义。

我们用于与鼠标驱动程序交互的mouseobj类程中的核心鼠标成员函数针对int 86()函数建立。成员函数mouseintr()仅用传送给它的变元装入各个寄存器，调用中断，然后把寄存器值往回拷贝到各变元中。包含在传送给mouse intr()的变元中的各值选择执行的特定鼠标函数。因此，该由mouseintr()的调用程序确保提供正确的变元值。贯穿我们的程序，将涉及到四个鼠标变元m1、m2、m3和m4。它们分别对应于寄存器AX、BX、CX和DX。下面是函数mouseintr()：

```

// This routine provides the communication between the mouse driver and
// an application program. There are several predefined mouse functions
// supported by the Microsoft mouse--see accompanying text. Parameters
// are sent back and forth to the mouse driver through the ax, bx, cx
// and dx registers.
void mouseobj::mouseintr(int &m1, int &m2, int &m3, int &m4)
{
    union REGS inregs, outregs;

    inregs.x.ax = m1;    inregs.x.bx = m2;
    inregs.x.cx = m3;    inregs.x.dx = m4;
    int86(0x33, &inregs, &outregs);
    m1 = outregs.x.ax; m2 = outregs.x.bx;
    m3 = outregs.x.cx; m4 = outregs.x.dx;
}

```

通过置m1、m2、m3和m4为不同的值，我们可以很容易执行各种操作，如初始化鼠标、读鼠标的当前位置或确定鼠标按钮的状态等。在下一节中将讨论这些操作。

请注意，此函数被赋予作为我们基础鼠标类程的mouseobj类程。我们小心地称这个类程为“基础类程”，因为它将用于派生键盘类程kbmouseobj。花儿几分钟考查一下列表8.1 (mouse.h)，你会看到“封装”技术可用于隐藏所有的数据和在类程体中鼠标处理操作的编码细节。为什么这样做呢？毕竟，我们这样就可以作为一组独立的函数为鼠标支撑例程编码。通过建立一个鼠标类程，我们能够把所有鼠标特定的编码细节放在同一个顶蓬下，并隐藏支持鼠标硬件的低层细节。因此，使用鼠标类程的应用程序不必知道鼠标如何工作的内部细节。

8.4 鼠标函数

鼠标驱动程序包含20个以上的鼠标函数，其中的大部分已列在表8.1中。我们不想使用所有这些函数，但为了便于你参引而把它们在这里列出。

现在让我们一步一步地了解实际上如何使用鼠标，以便可以开始开发我们定制的类型。

表 8.1 鼠标驱动程序函数

函数号 (放在AX)	描 述
0	重置鼠标并返回它的状态
1	示出屏幕上的鼠标光标
2	消除屏幕上的鼠标光标
3	返回鼠标位置和按钮的状态
4	把鼠标光标移到虚拟位置 (x, y)
5	检索自上次调用以来揿压按钮的次数
6	检索自上次调用以来释放按钮的次数
7	置光标的水平界限
8	置光标的垂直界限
9	定义在图形模式下使用的光标
10	置在正文模式下使用的光标

函数号 (放在AX)	描 述
11	读鼠标运动计数器
12	建立中断例程
13	启用光笔
14	关闭光笔
15	置光标的鼠标运动比率
16	隐藏区域中的鼠标
19	置允许较快鼠标运动的参数
20	交换中断例程
21	检索鼠标驱动程序状态
22	保存鼠标驱动程序状态
23	恢复鼠标驱动程序状态
29	置鼠标光标所用的CRT页号
30	取鼠标光标所用的CRT页号

8.4.1 鼠标初始化

在能够使用鼠标以前，我们必须首先把它初始化。这要通过调用0号函数（见表8.1）完成。调用此函数以后，鼠标驱动程序重置为各个缺省值，并在发现鼠标硬件和驱动程序时，在AX中返回-1。否则，返回0。因此，通过使用重置函数（函数0）能够告知鼠标是否出现。我们可以编写自己的鼠标重置成员函数为：

```
// Resets the mouse cursor to: screen center, mouse hidden, using arrow
// cursor and with minimum and maximum ranges set to full virtual screen
// dimensions. If a mouse driver exists, this function returns a 1.
// otherwise it returns a 0.
int mouseobj::reset(void)
{
    int m1, m2, m3, m4;

    m1 = RESET_MOUSE;
    mouseintr(m1, m2, m3, m4);
    return(m1);
}
```

RESET_MOUSE项是我们放在称为mouse.h的标题文件中的函数0的常数。我们将对后面几节讨论的其它鼠标函数定义类似常数（文件mouse.h在接近本章末尾与mouse.cpp一起列出）。

请注意，我们的重置例程比用函数号装入m1还多做一些工作，它调用mouseintr()，然后返回在m1中保存的状态标志。变量m2、m3和m4在此情况下忽略不管。大多数其它鼠标例程在形式上是类似的。

虽然reset()提供基本鼠标重置函数，但我们仍需要更高级的例程以处理初始化鼠标的完整任务。我们将调用成员函数init()。其中，它调用reset()且如果reset()成功，则置mouseobj变量mouseexists为TRUE。否则，mouseexists置为FALSE。稍后我们将看到，

在所有的鼠标成员函数中使用变量mouseexists,以便如果没有发现鼠标驱动程序,则可以避免对鼠标驱动程序的调用。实际上,当mouseexists为FALSE时,将切换到键盘输入。我们在下一节探讨这一功能。

显然需要扩充我们的init()函数,它的中间形式下面将示出。仍然没有提到的主要特征是当没有提供鼠标时如何处理键盘输入。正如前面提到的,这将添加到后面的节中。

```
// Call this function at the beginning of your program, but after the
// graphics adapter has been initialized. It will initialize the
// mouse and display the mouse cursor at the middle of the screen.
// Returns 0 if a mouse is not detected.
int mouseobj::init(void)
{
    int gmode;
    char far *memory = (char far *)0x004000049L;

    mouseexists = TRUE;
    if (reset()) { // If mouse reset okay, assume
        gmode = getgraphmode(); // mouse exists. Test if Hercules
        if (gmode == HERCMONOHI) { // is used. If so, patch memory

            *memory = 0x06; // location 40h:49h with 6.
            reset();
        }
        show(); // Show the mouse
        return(TRUE); // Return a success flag
    }
    else { // No mouse found--emulate a
        mouseexists = FALSE; // mouse using the keyboard
        return(FALSE); // Return a no-mouse found flag
    }
}
```

init()成员函数还通过调用鼠标函数move()把鼠标位置设在屏幕的左上角,并通过调用成员函数show()切换鼠标的显示。Move()和show()两者都是我们的鼠标类程的一部分,不久我们将会看到。

init()函数的一个奇特性是调用BGI例程getgraphmode(),它用于检验你的程序是否在Hercules卡上运行。如果是,则必须把6写到存储单元40h:49h上。这是Hercules板的特性,而对于任一其它图形适配器则不需要。

8.4.2 附加的鼠标成员函数

至此,我们已涉及到鼠标初始化和重置函数。现在将考查真正使鼠标可用的例程。表8.2为mouse.cpp包含的鼠标特定成员函数的完整表,最后两个函数getinput()和waitforinput()被设计成用鼠标和键盘两者工作。当在本章稍后讨论键盘类程时,将考查这两个例程。

我们由编写这些函数开始,并假定提供了鼠标。稍后将派生kbdmouseobj类程,以便当没有提供鼠标时支持键盘。我们的方法有趣的一面是能将使用kbdmouseobj中的成员函数创建键盘处理类程。

8.4.3 鼠标光标

我们将考查控制鼠标光标显示的下一个成员函数,我们在init()中已看到称为show()

的成员函数，该函数打开鼠标光标的显示。还有一个辅助函数hide()，它从屏幕除去鼠标光标。这里要注意的重要事情是，这两个例程仅影响鼠标光标的显示。换句话说，不管鼠标光标的显示状态是什么，鼠标驱动程序总能修改和保持光标的位置。

show()和hide()函数分别使用鼠标函数1和函数2。它们不要求任何变元或返回任何值。这两个函数是简单的，并包含在列表8.2示出的mouse.cpp源文件中。

表 8.2 mouse.cpp中的鼠标处理成员函数

函数	描述
mouseintr()	低层鼠标调用的接口
init()	初始化鼠标
reset()	重置鼠标并返回它的状态
move()	把鼠标移到位置 (x, y)
show()	显示鼠标光标
hide()	消除屏幕上的鼠标光标
getcoords()	取鼠标光标的坐标
buttonpressed()	检测自上次调用以来是否撤压了鼠标按钮
buttonreleased()	检测自上次调用以来是否释放了鼠标按钮
testbutton()	测试按钮的内部例程
inbox()	检查鼠标是否在屏幕的一个区域内
getinput()	检查是否撤按或释放了鼠标按钮或是否按了一个键
waitfoinput()	循环直到撤压了一个按钮或直到按了一个键

不难想象经常需要使用show()和hide()多次打开或关闭鼠标光标。但是，它们的用途可能比初步印象更为重要。每当你使用鼠标在屏幕上读或写任何东西时，必须总是先通过调用hide()关闭鼠标。在访问屏幕之后，你可以通过调用show()恢复鼠标显示。为什么呢？因为鼠标光标映像实际上要组合到屏幕映像中。因此，每当访问屏幕时又打开鼠标光标，你可能会冒访问到鼠标光标映像的危险，或至少不正确地修改了在鼠标光标下的某些东西。仅在访问屏幕之前和之后，使用hide()和show()，才可以向你保证鼠标将不影响屏幕上的什么东西。

一个补充的警告是，如果已经不显示鼠标，则不应调用hide()。如果你违反此规则，

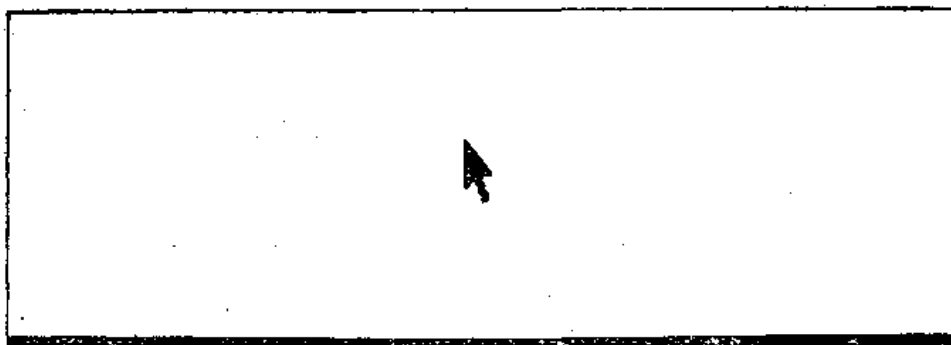


图 8.2 图形鼠标光标

则必须在对hide()的调用之后,伴随对show()的调用。这是由当显示鼠标光标时鼠标驱动程序内部确定的方法所要求的。

在图形模式下,缺省鼠标光标被显示为如图8.2中所示的那样的箭头符号。虽然可以通过调用函数9改变显示的光标类型,但我们没有在鼠标驱动程序中利用此特点。

8.4.4 鼠标位置

上一节,我们已经得知鼠标驱动程序如何控制鼠标光标映像的显示。现在让我们看看怎样可以通过驱动程序访问和控制鼠标光标的位置。

鼠标驱动程序的部分职责是保持鼠标光标的位置。此外,我们可以通过调用函数3查询鼠标驱动程序,以便返回鼠标的坐标。我们的鼠标类程包含执行此操作的称为getcoords()的成员函数。

函数3不要求传送给它任何变元,它在m3和m4返回鼠标的x和y坐标。实际上,鼠标函数3不仅返回鼠标坐标,它还在m2中返回按钮的当前状态,但是,我们不管函数的这个方面。

函数3返回的坐标在大多数情况下将对应于鼠标的屏幕坐标。但是,请注意这是“在大多数情况下”。原来,鼠标驱动程序参引虚拟坐标系中的所有鼠标坐标,而不是屏幕坐标中的。通常,这两个坐标系统在图形模式下相同,但是,当屏幕是针对320列的模式时,x方向上的虚拟坐标是不相同的。然而调整十分简单。在这些情况下,真正屏幕坐标总是虚拟坐标的一半。因此,每当当前图形模式有320列时,仅需要用2除一下鼠标驱动程序的坐标。图8.3示出320列图形模式的坐标系统之间的关系。至此,我们的getcoord()成员函数变成:

```
void mouseobj::getcoords(int &x, int &y)
{
    int m1, m2;

    m1 = GET_MOUSE_STATUS;
    mouseintr(m1,m2,x,y);
    if (getmaxx() == 319) x /= 2; // Adjust for virtual coordinates
    // of the mouse
}
```

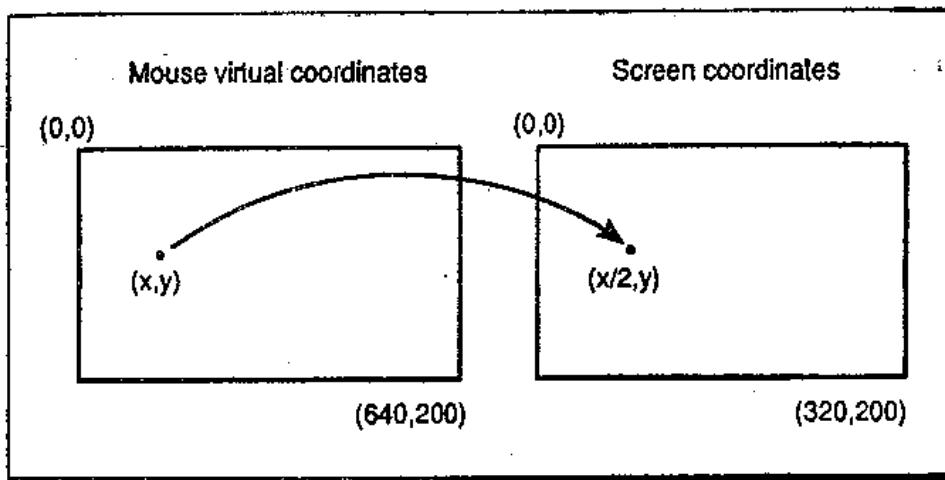


图 8.3 鼠标和CGA320×200模式坐标之间的变换

8.4.5 鼠标按钮

Microsoft鼠标还提供可通过鼠标驱动程序访问的两个按钮。请注意，虽然其它的鼠标可能有更多按钮，但通常这些按钮可以作为Microsoft鼠标的超集处理。

有三种方法检测鼠标按钮。对于每一个按钮，你可以检测它的当前状态，是否已被按下，以及是否已被释放。我们使用程序中的后两个检测按钮的成员函数并避免检测按钮的当前状态，因为用它也许不能足够快地收集所有按下的按钮信息。

鼠标函数5和函数6用于确定自从最后检测或鼠标初始化以后，鼠标按钮之一被按下或释放的次数。两个函数都可以检测每一个按钮。检测哪一个特定的按钮取决于在函数调用时参数m2中的值。如果m2是0，则检测左按钮；如果m2是1，则检测右按钮。按钮动作的次数（即对函数5按按钮的次数）返回在m2中。此外，这些函数在m3和m4返回最后按钮动作的位置（m3是x坐标，m4是y坐标）。请注意，各自独立保持左按钮和右按钮状态信息。

我们的鼠标类使用成员函数buttonpressed()和buttonreleased()中的函数5和函数6检测鼠标按钮的状态。这两个函数都是布尔函数，它们接受指定要检验哪一个按钮的单个变元。此变元可取三个值之一，该值在mouse.h中定义为LEFT_BUTTON、RIGHT_BUTTON和EITHER_BUTTON。在buttonpressed()的情况下，如果自从最后调用buttonpressed()后已按了指示的按钮，则函数返回非0值。否则，返回0。类似地，如果在它的变元表中规定的按钮自最后一次调用以后已被释放，则buttonreleased()仅返回非0值。因此，假定我们说明称为mouse的mouseobj，我们可以编写以下循环，它等待直到按下任一按钮为止：

```
while (!mouse.buttonpressed(EITHER_BUTTON)) :
```

实际上，低级的鼠标函数不允许我们问是否已按下任一按钮。我们必须各自检测左按钮和右按钮。但是，这容易完成。

由于按按钮和释放按钮函数之间有很多相似性，所以我们把它们代码组合成一个称为testbutton()的成员函数。此函数也返回一个布尔值；但是，它取两个变元：检测（按或释放按钮）的动作和检查哪一个按钮。成员函数是：

```
int mouseobj::testbutton(int testtype, int whichbutton)
{
    int m1, m2, m3, m4;

    m1 = testtype;
    if (whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON) {
        m2 = LEFT_BUTTON;
        mouseintr(m1, m2, m3, m4);
        if (m2) return(TRUE); // Return TRUE if the action occurred
    }
    if (whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON) {
        m1 = testtype;
        m2 = RIGHT_BUTTON;
        mouseintr(m1, m2, m3, m4);
        if (m2) return(TRUE); // Return TRUE if the action occurred
    }
    return(FALSE); // Return FALSE as a catchall
}
```

请注意，鼠标函数号装入到m1中，它是作为变元testtype被传送的。此外，为在左按钮和右按钮之间选择，从变元whichbutton装入鼠标变量m2。请记住，此变量可以取以下值之一：LEFT_BUTTON、RIGHT_BUTTON或EITHER_BUTTON。但是请注意，如果用户规定EITHER_BUTTON，则必须分别检测两个按钮。

testbutton() 通过mouseintr() 调用鼠标驱动程序之后，m2包含whichbutton指定的按钮的撤按动作数。因为我们仅关心按钮是否已被按或被释放，而不管它出现多少次。所以，如果m2指出按了一次或多次按钮，则testbutton()都返回TRUE。否则，testbutton()返回FALSE。

函数testbutton() 忽略你可能觉得有价值的两块信息。第一，它抛弃了自最后检验以后出现的按钮动作数。你可能发现这是有用的，虽然在紧凑的代码循环中，你不太可能足够快地记录不只一次的撤按或释放按钮。

还有，函数5和函数6都在m3和m4中返回最后按钮动作的坐标。我们将忽略这些而改用getcoords()。在按钮动作以后检索鼠标的位置；但是，按钮检测和对getcoords()的调用之间的延迟可能是重要的。因此，你可能想要把返回到m3和m4中的坐标综合到你的代码中以避免此情况。

8.4.6 在方框中的鼠标

当使用鼠标时，我们常常想能够检测鼠标光标是否在屏幕的特定区域中。为完成这点，鼠标类程中包含一个称作inbox()的成员函数。函数inbox()检查提供给它的鼠标光标坐标是否在如图8.4示出的一对屏幕坐标定界的矩形区域中。屏幕坐标对应于正检测区域的左上角和右下角。如果鼠标在矩形中，那末，inbox()将返回1；否则它将返回0，函数是：

```
int mouseobj::inbox(int left, int top, int right, int bottom, int x, int y)
{
    return((x >= left && x <= right && y >= top && y <= bottom) ? 1 : 0);
}
```

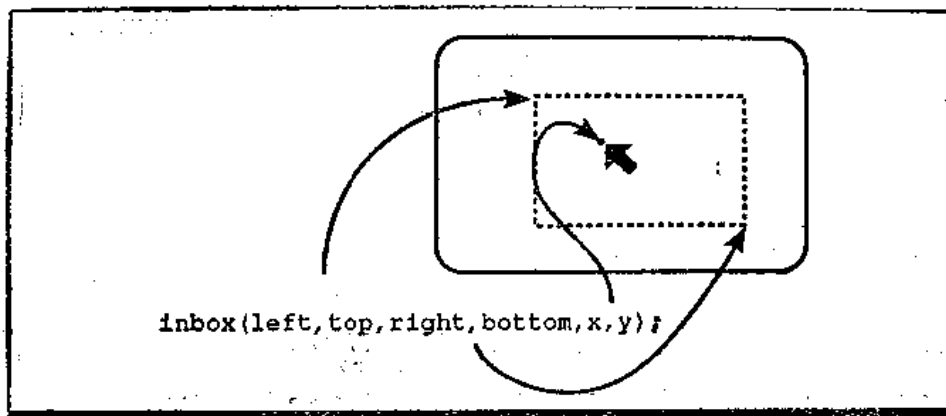


图 8.4 用inbox() 查找区域中的鼠标

8.4.7 更多的鼠标控制

引进move() 成员函数能使我们的鼠标控制函数更加完美。此函数使用鼠标 函数4把

鼠标光标移到特定的 (x, y) 屏幕位置。请注意,当图形适配器为320列模式时,在move()中传送给它的屏幕坐标必须调整为虚拟坐标。这是由下面的行完成的:

```
if (getmaxx() == 319) x *= 2;
```

8.5 增添键盘输入

虽然鼠标在图形世界中是最好的输入设备,但有时若能使用键盘也是有用的。例如,往往通过键入快速键组合而不是使用鼠标更便于选择函数。因此,我们准备提供某些处理键盘的新例程。

如果仔细观察mouseobj类程,则你会看到已包含处理键盘的两个成员函数getinput()和waitforinput()。这些函数是为使用键盘和鼠标工作而设计的。编写它们,使得首先检查键盘缓冲区以审定是否按了键盘。如果是,则这些函数立即返回在键盘缓冲区中的第一个字符。否则,例程检测适当的鼠标按钮以查看按钮是否已被按下或释放。实际上,waitforinput()做的只不过是循环中调用getinput(),直到已按按钮或用户键入一个字符为止。

由Turbo C++函数kbhit()检查键盘缓冲区。如果它返回0,则没有按按钮,getinput()继续检测鼠标按钮是否已被按下。如果已按下,则getinput()返回-1。否则,如果已按键,则getinput()返回按键的值。

8.5.1 仿真鼠标

至此,我们仅仅讲了有关使用鼠标的问题。但是,并不是每一个人都有鼠标,而且现实的情况是,好的程序必须也适应非鼠标系统。虽然可以使用提供mouseobj类程的getinput()和waitforinput()以帮助支持键盘和鼠标,你可能应当把支持键盘综合到你的应用程序中。这也许是非常难做的工作。

我们通过派生kbdmouseobj类程(见列表8.3和列表8.4)能够提供最好的解。因为此类程是从mouseobj类程派生的,它继承了所有鼠标类程的数据和函数。为支持键盘,我们随后可以修改某些继承的成员函数,例如init()。并且添加特别为用键盘控制仿真鼠标工作而设计的一些新的函数。也就是说,我们将能用新的类程而不管实际是否有连接的鼠标。我们将使用光标键围绕屏幕移动仿真鼠标光标,并用INS和DEL键模拟鼠标按钮。此方法允许我们利用已经编写的代码而不必从草稿开始。

表 8.3 kbdmouseobj中的键盘输入成员函数

成员函数	描 述
getkbinteraction()	按箭头键时修改光标的位置或掀压按钮键时置按钮掀压标志
getkb()	执行某些扩展键码的低层键盘读入
initcursor()	初始化键盘对象以便它模拟鼠标
togglecursor()	打开或关闭模拟的鼠标光标
drawcursor()	画模拟的鼠标光标

由kbdmouseobj引入的新成员函数在表8.3中列出。它们仅负责支持键盘。提供的其

它函数被设计成综合支持鼠标和键盘两者。如果考查列表8.3中的类程定义(kbdmouse.h), 将会看到支持两种设备的成员函数用virtual关键字说明。例如, 请注意init() 函数的说明是:

```
virtual int init(void);
```

这些函数实际上是从mouseobj类程继承下来的, 这里它们被说明为virtual, 因为我们将重设它们, 以便它们可以执行不同的操作。为了解如何使用此技术。让我们仔细看看init() 成员函数。

8.5.2 初始化键盘对象

为支持键盘, 你必须使用kbdmouseobj类程创建一个对象。在使用键盘对象以前, 必须记住调用init() 成员函数。此函数稍微不同于我们已编写的任一其它的函数, 因为它产生对另一个init() 成员函数 (即在mouseobj类程中定义的函数) 的调用。为了解这如何工作, 让我们考查对kbdmouseobj的完整init() 函数。

```
int kbdmouseobj::init(void)
{
    if (!mouseobj::init())           // No mouse found--emulate the mouse
    {
        cinc = 1; cminx = 0;         // The functionality is the
        cmaxx = getmaxx() - 1;       // same as the real thing
        cmaxy = getmaxy() - 1;       // Therefore, set the emulated
        cminx = 0; cminy = 0;        // cursor to the top left of the
        cdeltax = cdeltay = 0;        // screen, set its bounds to the
        cx = cmaxx/2; cy = cmaxy/2;  // full screen and initialize its
        lbuttonpress = FALSE;        // mouse movement counters, and
        rbuttonpress = FALSE;        // initialize all button states
        numlpress = 0;                // to false
        numrpress = 0;                // Set simulated mouse button
        numlrelease = 0;              // press and release counters to
        numrrelease = 0;              // zero
        initcursor();                 // Create cursor image
        cursoron = FALSE;
        show();                        // Call show() to display it
        return(0);                    // return a no mouse found flag
    }
    else
    {
        return(1);
    }
}
```

函数中的第一个代码语句揭示出如何初始化的秘密,

```
if (!mouseobj::init()) {
```

此代码检查是否已通过调用与mouseobj类程相关的init()例程安装了鼠标。由于继承性, 如果mouseobj类程名字如所示放在函数调用之前, 则我们从键盘对象访问mouseobj类程中的任一成员函数。如果找到鼠标, 则新的init() 键盘成员函数跳过建立鼠标仿真所需的代码。

请注意, 仿真鼠标要求几个要设置的新变量和要调用的函数。首先, 必须建立仿真光标, 必须初始化变量以记住鼠标光标的位置以及它的按钮状态。因为处理键盘所需的数据隐藏在kbdmouseobj类程中, 故不用担心如何跟踪这些信息。不用说, 我们采取的方法大大简化了把键盘添加到交互图形应用的工作。实际上, 唯一要讨论的是你想不想增加键

盘。如果你仅想使用鼠标,则建立一个如下的对象:

```
mouseobj mouse1;
```

但是,如果你想在提供鼠标时使用鼠标,而在不提供鼠标时使用键盘,则使用kbdmouseobj类程:

```
kbdmouseobj inpdev;
```

不管你使用哪一类程,初始化过程是相同的。只要简单记住在你的程序开头,在已初始化图形适配器以后,调用init()成员函数。

8.5.3 仿真鼠标光标

现在我们看看如何可以仿真鼠标光标。基本思想是使用getimage()和putimage()移动有关屏幕的仿真鼠标光标的映像。我们将在称为initcursor()的新的成员函数中建立模拟的光标映像。kbdmouseobj类程私有的、称为cursor的指针将指向我们实际在函数drawcursor()中画出的光标映像。通过调用malloc()在initcursor()中分配光标映像的空间。此外,在initcursor()中分配存储,使得称为undercursor的另一个私有指针可用于保存当前由仿真鼠标光标覆盖的部分屏幕。于是,移动屏幕上的仿真光标变成三步过程:

- (1) 通过使用以前保存在undercursor中的映像重写它,从屏幕中除去当前光标映像;
- (2) 使用getimage(),以保存在undercursor中的屏幕区域,其中光标是在出现的信息下面;
- (3) 使用putimage(),以便在它新的位置绘出光标。

在initcursor()中建立原始光标映像遵循这同样的过程。首先,要建立光标映像的屏幕区保存在undercursor中。然后由drawcursor()使用一系列对line()的调用绘出箭头形状的光标。其次,通过调用getimage()把光标映像保存在cursor,以便稍后把它拷贝到屏幕。最后,通过使用在undercursor中的映像重写该光标,把屏幕恢复为它的原来状态。这样,将除去光标映像并把屏幕恢复到它的原始状态。

如果你愿意用所采用的光标作试验,则只需修改drawcursor()。光标的大小由在kbdmouse.h中定义的常数CURSHEIGHT和CURSWIDTH规定为高8个像素和宽8个像素。

8.5.4 仿真鼠标位置

仿真鼠标光标的位置保持在变量cx和cy中。结果,对getcoords()的调用只不过是在使用仿真的光标时返回这些值。每当应用程序调用getinput()和用户在键盘上掀按箭头键之一时,就更新cx和cy变量。这后者的改变包含在称为getkbinteraction()的新成员函数的扩充代码部分。实质上,此例程查看是否采用了键盘动作。如果是,则调用getkb()以检索输入(它可以是扩充的扫描码)。其次,把输入与对应于各种键盘组合的代码相匹配。如果按了箭头键之一,则函数修改cx和cy。请注意,仿真的光标位置也裁剪为坐标cminx、cminy、cmaxx和cmaxy。这些裁剪参数可用于模拟鼠标函数7和函数8,它限制鼠标光标移动到屏幕上的矩形区域。最后,请注意,每当它调整仿真光标的位置时,getkbinteraction()返回0。这等于告诉调用函数不需要采取动作,或换句话说,不要掀按键

或不要按仿真按钮。

8.5.5 仿真鼠标按钮

由键盘上的INS和DEL键仿真鼠标按钮。在getkbinteraction()中的switch 语句测试是否已按了这些键,并在已经按了某个键时,设置标志lbuttonpress或rbuttonpress。如果已分别按左按钮和右按钮,则这些标志为TRUE。此外,每次按按钮时,增加两个计数器numlpress和numrpress。由例程buttonpressed()和buttonreleased()使用的这些值可发现某些被按下的按钮并把实情报告给用户。

使用此技术发现已按下按钮是容易的。但是,仿真释放按钮则是另外一回事了。为解决此问题,我们将计数为释放按钮而每隔一次地撤按的INS键或DEL键。因此,每当lbuttonpress或rbuttonpress为TRUE且按下INS键或DEL键时,就认为是释放按钮并增加专用的计数器numlrelease和numrrelease,以便记录释放按钮的次数。如前所述,由模拟的按钮例程使用的这些值来发现某些释放的按钮,并把实情报告给用户。

• 列表 8.1 mouse.h

```
// mouse.h --Includes function prototypes and constants for
// the functions in mouse.cpp. The following is a list of constants
// that correspond to the mouse functions supported in mouse.cpp.
#define MOUSEH // Prevent header file from being recompiled
#define MOUSEH
const int RESET_MOUSE=0;
const int SHOW_MOUSE=1;
const int HIDE_MOUSE=2;
const int GET_MOUSE_STATUS=3;
const int SET_MOUSE_COORD=4;
const int CHECK_BUTTON_PRESS=5;
const int CHECK_BUTTON_RELEASE=6;
const int GET_MOUSE_MOVEMENT=11;
const int LEFT_BUTTON=0; // Use left button
const int RIGHT_BUTTON=1; // Use right button
const int EITHER_BUTTON=2; // Use either button
class mouseobj {
public:
    // Internal variable set true if a mouse driver is
    int mouseexists, // detected during initialization. Used to select
    // between the mouse and keyboard emulated mouse.
    virtual void mouseintr(int &m1, int &m2, int &m3, int &m4);
    virtual int init(void);
    virtual int reset(void);
    virtual void hide(void);
    virtual void show(void);
    virtual void move(int x, int y);
    virtual void getcoords(int &x, int &y);
    virtual int buttonreleased(int whichbutton);
    virtual int buttonpressed(int whichbutton);
    int testbutton(int testtype, int whichbutton);
    int inbox(int left, int top, int right, int bottom, int x, int y);
    virtual int mouseobj::getinput(int whichbutton);
    int mouseobj::waitforinput(int whichbutton);
};
#endif
```

• 列表 8.2mouse.cpp

```
// mouse.cpp -- Routines to support a Microsoft compatible mouse.
// Mouse support is divided into two classes. The first,
// mouseobj, provides most of the functions you'll need to control
// the mouse. The other, kbmouseobj, is a class derived from
// mouseobj that overrides the mouse functions so that they are
// emulated by the keyboard. This class is used when a mouse does
// not exist. Both classes assume they are running in graphics mode.
// To move the emulated cursor, use the arrow keys on the keyboard
// and the INS and DEL keys as the left and right mouse buttons,
// respectively. The gray + and - keys can be used to change the
// amount the emulated mouse cursor is moved by.
#include <alloc.h>
#include <math.h>
#include <dos.h>
#include <graphics.h>
#include <process.h>
#include <stdio.h>
#include <conio.h>
#include "mouse.h"

const int TRUE = 1;
const int FALSE = 0;

// This routine provides the communication between the mouse driver and
// an application program. There are several predefined mouse functions
// supported by the Microsoft mouse--see accompanying text. Parameters
// are sent back and forth to the mouse driver through the ax, bx, cx,
// and dx registers.
void mouseobj::mouseintr(int &m1, int &m2, int &m3, int &m4)
{
    union REGS inregs, outregs;

    inregs.x.ax = m1;    inregs.x.bx = m2;
    inregs.x.cx = m3;    inregs.x.dx = m4;
    int86(0x33, &inregs, &outregs);
    m1 = outregs.x.ax;    m2 = outregs.x.bx;
    m3 = outregs.x.cx;    m4 = outregs.x.dx;
}

// Call this routine at the beginning of your program, but after the
// graphics adapter has been initialized. It will initialize the
// mouse and display the mouse cursor at the middle of the screen.
// Returns 0 if a mouse is not detected
int mouseobj::init(void)
{
    int gmode;
    char far *memory = (char far *)0x004000049L;

    mouseexists = TRUE;
    if (reset()) { // If mouse reset okay, assume
        gmode = getgraphmode(); // mouse exists. Test if Hercules
        if (gmode == HERCMONOHI) { // is used. If so, patch memory
            *memory = 0x06; // location 40h:49h with 6.
            reset();
        }
        show(); // Show the mouse
        return(TRUE); // Return a success flag
    }
}
```

```

    }
    else {
        mouseexists = FALSE; // No mouse found--emulate a
        return(FALSE);      // mouse using the keyboard
    }
}

// Resets the mouse cursor to: screen center, mouse hidden, using arrow
// cursor and with minimum and maximum ranges set to full virtual screen
// dimensions. If a mouse driver exists, this function returns a 1
// otherwise it returns a 0.
int mouseobj::reset(void)
{
    int m1, m2, m3, m4;

    m1 = RESET_MOUSE;
    mouseintr(m1,m2,m3,m4);
    return(m1);
}

// Moves the mouse to the location (x,y)
void mouseobj::move(int x, int y)
{
    int m1, m2;

    m1 = SET_MOUSE_COORD;
    if (getmaxx() == 319) x *= 2; // Adjust between virtual and actual
    mouseintr(m1,m2,x,y);        // coordinates if necessary
}

// Removes the mouse cursor from the screen. Call this function before you
// write or draw anything to the screen. It is also a good idea to turn off
// the mouse at the end of a program. Use show() to restore the mouse
// on the screen. The mouse movement will be maintained while the mouse is
// not visible. Due to a peculiarity of the mouse driver, make sure you
// don't call hide() if the mouse is not already visible. See text
// discussion for more on this.
void mouseobj::hide(void)
{
    int m1, m2, m3, m4;

    m1 = HIDE_MOUSE; // Invoke the hide mouse function
    mouseintr(m1,m2,m3,m4);
}

// Display the mouse cursor. Normally, you should not call this
// function if the mouse is already visible. The keyboard mouse
// is clipped to the minimum and maximum ranges in this routine.
void mouseobj::show(void)
{
    int m1, m2, m3, m4;

    m1 = SHOW_MOUSE;
    mouseintr(m1,m2,m3,m4); // Display the mouse cursor
}

// Get the current location of the mouse cursor
void mouseobj::getcoords(int &x, int &y)
{
    int m1, m2;

    m1 = GET_MOUSE_STATUS;
    mouseintr(m1,m2,x,y);
    if (getmaxx() == 319) x /= 2; // Adjust for virtual coordinates
    // of the mouse
}

```

```

// Test if a button has been released since the last call to this
// function. If so, return 1, otherwise return 0.
int mouseobj::buttonreleased(int whichbutton)
{
    return(testbutton(CHECK_BUTTON_RELEASE,whichbutton));
}

// Return a 1 if the mouse button specified has been pressed since the
// last check with this function. If the button has not been pressed,
// return a 0.
int mouseobj::buttonpressed(int whichbutton)
{
    return(testbutton(CHECK_BUTTON_PRESS,whichbutton));
}

// Called by buttonpressed() and buttonreleased() to explicitly
// test the mouse button states. The function returns TRUE if the
// specified mouse button (in whichbutton) performed the specified
// action (as indicated by testtype). Otherwise the function returns
// FALSE which means that the action tested for did not occur
int mouseobj::testbutton(int testtype, int whichbutton)
{
    int m1, m2, m3, m4;

    m1 = testtype;
    if (whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON) {
        m2 = LEFT_BUTTON;
        mouseintr(m1,m2,m3,m4);
        if (m2) return(TRUE); // Return TRUE if the action occurred
    }

    // Save the image of the screen where the cursor image will be created,
    // Clear this space and call drawcursor() to draw the cursor.
    getimage(cx,cy,cx+CURSWIDTH,cy+CURSHEIGHT,undercursor);
    setlinestyle(SOLID_LINE,0,0);
    setfillstyle(SOLID_FILL,BLACK);
    bar(cx,cy,cx+CURSWIDTH,cy+CURSHEIGHT);

    drawcursor(cx,cy);
    // Save the image of the cursor and overwrite the screen area where the
    // cursor is with the original screen image.
    getimage(cx,cy,cx+CURSWIDTH,cy+CURSHEIGHT,cursor);
    putimage(cx,cy,undercursor,COPY_PUT);
}

// Test if the mouse cursor is within the box specified. Returns TRUE
// if the mouse is in the box; otherwise the function returns FALSE.
int mouseobj::inbox(int left, int top, int right, int bottom, int x, int y)
{
    return((x >= left && x <= right && y >= top && y <= bottom) ? 1 : 0);
}

// Returns a character if a key has been pressed, or -1 if a mouse
// button has been pressed, or a zero if none of the above. If a
// mouse exists, this routine favors any keyboard action.
int mouseobj::getinput(int whichbutton)
{
    if (kbhit()) // Check if a key has been pressed
        return(getch()); // Return the character
    else {
        if (buttonpressed(whichbutton)) {

```

```

        while (!buttonreleased(whichbutton)) ;
        return(-1);
    }
    else if (buttonreleased(whichbutton))
        return(-1);
    return(0);
}

// Continue to call getinput() until a button or key has been pressed
int mouseobj::waitforinput(int whichbutton)
{
    int c;

    while ((c=getinput(whichbutton)) == 0) ;
    return(c);
}

```

• 列表 8.3kbdmouse.h

```

// kbdmouse.h -- Derived mouseobj class that emulates the mouse
// with the keyboard.
#ifndef KBDMOUSEH // Prevent header file from being recompiled
#define KBDMOUSEH
#include "mouse.h"
const int MAXINC = 32; // Largest increment amount of emulated mouse cursor
const int CURSWIDTH = 8; // The emulated cursor is 8 pixels wide

const int CURSHEIGHT = 8; // 8 pixels high
class kbdmouseobj : public mouseobj {
    int cx, cy; // Internal variables used to maintain
                // the cursor location when the mouse is not used
    int cinc; // Internal variable used for increment
                // amount of the nonmouse cursor
    void *cursor; // Points to image of the emulated mouse
    void *undercursor; // Area saved under emulated mouse
    int cursoron; // TRUE if cursor currently visible
    int cminx, cmaxx; // Minimum, maximum x coordinates for cursor
    int cminy, cmaxy; // Minimum, maximum y coordinates for cursor
    int lbuttonpress; // TRUE if simulated left or right
    int rbuttonpress; // button is pressed
    int numlpress; // Count the number of button presses
    int numrpress;
    int numlrelease; // Count the number of button releases
    int numrrelease;
    int cdeltax, cdeltay; // Keeps track of emulated mouse's movements
public:
    virtual int init(void);
    virtual void hide(void);
    virtual void show(void);
    virtual void move(int x, int y);
    virtual void getcoords(int &x, int &y);
    virtual int buttonreleased(int whichbutton);
    virtual int buttonpressed(int whichbutton);
    virtual int getinput(int whichbutton);
    int getkbinteraction(int whichbutton);
    int getkb(void);
    void initcursor(void);
    void togglecursor(void);
}

```

```

        void drawcursor(int x, int y);
    };
#endif

```

• 列表 8.4 kbdmouse.cpp

```

// kbdmouse.cpp -- Routines to emulate a mouse with a keyboard.
// To move the emulated cursor use the arrow keys on the keyboard
// and the INS and DEL keys as the left and right mouse buttons,
// respectively. The gray + and - keys can be used to change the
// amount the emulated mouse cursor is moved. Note that the routines
// will call the mouseobj member functions if a mouse exists.
#include <alloc.h>
#include <math.h>
#include <dos.h>
#include <graphics.h>
#include <process.h>
#include <stdio.h>
#include <conio.h>
#include "mouse.h"
#include "kbdmouse.h"

const int TRUE = 1;
const int FALSE = 0;

// Call this function at the beginning of your program, after the graphics
// adapter has been initialized. It will initialize the mouse and display
// the mouse cursor at the middle of the screen. If a mouse is not
// present it will cause the emulated mouse to appear and the keyboard
// will be used to move the "mouse" cursor
int kbdmouseobj::init(void)
{
    if (!mouseobj::init()) { // No mouse found--emulate the mouse
        cinc = 1; cminx = 0; // The functionality is the
        cmaxx = getmaxx() - 1; // same as the real thing
        cmaxy = getmaxy() - 1; // Therefore, set the emulated
        cminx = 0; cminy = 0; // cursor to the top left of the
        cdeltax = cdeltay = 0; // screen, set its bounds to the
        cx = cmaxx/2; cy = cmaxy/2; // full screen and initialize its
        lbuttonpress = FALSE; // mouse movement counters, and
        rbuttonpress = FALSE; // initialize all button states
        numlpress = 0; // to false
        numrpress = 0; // Set simulated mouse button
        numlrelease = 0; // press and release counters to
        numrrelease = 0; // zero
        initcursor(); // Create cursor image
        cursoron = FALSE;
        show(); // Call show() to display it
        return(0); // return a no mouse found flag
    }
    else
        return(1);
}

// Moves the mouse to the location (x,y)
void kbdmouseobj::move(int x, int y)
{
    if (mouseexists)

```

```

        mouseobj::move(x,y);
    else {
        hide();           // Erase the current mouse cursor
        cx = x; cy = y;   // Update the mouse cursor's location
        show();           // Display mouse at the new location
        cdeltax = cdeltay = 0; // Reset the mouse movement variables
        return;
    }
}

// Removes the mouse cursor from the screen. Call hide() before writing or
// drawing to the screen. It is also a good idea to turn off the mouse at
// the end of a program. Use show() to restore the mouse on the screen. The
// mouse movement will be maintained while the mouse is not visible. Due to
// a peculiarity of the mouse driver, make sure you don't call hide() if
// the mouse is not already visible. See text discussion for more on this
void kbdmouseobj::hide(void)
{
    if (mouseexists)
        mouseobj::hide();
    else // Mouse doesn't exist, so turn
        togglecursor(); // off the emulated cursor
}

// Display the mouse cursor. Normally, you should not call this
// function if the mouse is already visible. The keyboard mouse
// is clipped to the minimum and maximum ranges in this routine.
void kbdmouseobj::show(void)
{
    if (mouseexists)
        mouseobj::show(); // If the mouse doesn't exist,
    else // turn on the emulated cursor
        togglecursor();
}

// Get the current location of the mouse cursor
void kbdmouseobj::getcoords(int &x, int &y)
{
    if (mouseexists)
        mouseobj::getcoords(x,y);
    else {
        x = cx; y = cy; // The position of the emulated
    } // mouse is given by cx and cy
}

// Test if a button has been released since the last call to this
// function. If so, return 1, otherwise return 0. In the keyboard mode,
// button release is simulated by striking on the button key again.
int kbdmouseobj::buttonreleased(int whichbutton)
{
    if (mouseexists)
        return(mouseobj::buttonreleased(whichbutton));
    else {
        if ((whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON) &&
            numrelease) {
            numrelease--;
            return(TRUE);
        }
        else if ((whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON)
            && numrrelease) {

```



```

        numrelease--;
        return(TRUE);
    }
    // If there isn't already a button released, check and see if
    // the user just released one, and if so repeat the tests above.
    else if (getkbinteraction(whichbutton) < 0) {
        if (whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON)
            return(TRUE);
        else if (whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON)
            return(TRUE);
    }

    return(FALSE);    // Return a value that the button was not pressed
}

// Return a 1 if the mouse button specified has been pressed since the
// last check with this function. If the button has not been pressed,
// return a 0.
int kbdmouseobj::buttonpressed(int whichbutton)
{
    if (mouseexists)
        return(mouseobj::testbutton(CHECK_BUTTON_PRESS,whichbutton));
    else {
        if ((whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON) &&
            numlpress) {
            numlpress--;
            return(TRUE);
        }
        else if ((whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON)
            && numrpress) {
            numrpress--;
            return(TRUE);
        }
        // If there isn't already a button pressed, check and see if
        // the user just pressed one, and if so repeat the tests above.
        else if (getkbinteraction(whichbutton) < 0) {
            if (whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON)
                return(TRUE);
            else if (whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON)
                return(TRUE);
        }
        return(FALSE);    // Return a value that the button was not pressed
    }
}

// Returns a character if a key has been pressed, or -1 if an emulated
// mouse button has been pressed, or a zero if none of the above. If a
// mouse exists, this function calls the mouse routines.
int kbdmouseobj::getinput(int whichbutton)
{
    int c;

    if (mouseexists)
        return(mouseobj::getinput(whichbutton));
    else {
        c = getkbinteraction(whichbutton);
        return(c);
    }
}

// This routine is used only if the mouse does not exist. It updates

```

```

// the location of the cursor whenever an arrow key is pressed or
// will set the button pressed flags if a button key is pressed. In
// addition, it will change the cursor increment amount if the plus or
// minus keys are pressed. If the "button" specified is pressed or was

// already pressed, the function returns -1. If an arrow key is pressed
// then a 0 is returned, which means that the caller does not have to
// perform any action. Finally, if some other key is pressed, then its
// value is returned.
int kbdmouseobj::getkbinteraction(int whichbutton)
{
    int c;

    if (kbhit()) {
        c = getkb();
        hide(); // Emulated mouse may be moved
        switch(c) {
            case 0x5200 : // INS key -> Emulate left mouse button press
                lbuttonpress = (lbuttonpress) ? FALSE : TRUE;
                show();
                if (whichbutton == LEFT_BUTTON || whichbutton == EITHER_BUTTON)
                    return(-1); // Return mouse button click signal
                if (lbuttonpress) numlpress++;
                else numlrelease++;
                return(0); // Wrong mouse "button" pressed
            case 0x5300 : // DEL key emulates right mouse button
                show();
                rbuttonpress = (rbuttonpress) ? FALSE : TRUE;
                if (whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON)
                    return(-1); // Return mouse button signal
                if (rbuttonpress) numrpress++;
                else numrrelease++;
                return(0); // Wrong mouse "button" pressed
            case 0x002B : // '+' key: increase the mouse movement amount
                cinc = (cinc < MAXINC) ? cinc + 6 : MAXINC;
                break;
            case 0x002D : // '-' key: decrease the mouse movement amount
                cinc = (cinc > 1 + 6) ? cinc - 6 : 1;
                break;
            case 0x4800 : // Up key moves the cursor up by the
                if ((cy -= cinc) < cminy) cy = cminy;
                cdeltay -= cinc; // increment amount and clip; also
                break; // decrement movement amount
            case 0x5000 : // Down key
                if ((cy += cinc) > cmaxy) cy = cmaxy;
                cdeltay += cinc;
                break;
            case 0x4B00 : // Left key
                if ((cx -= cinc) < cminx) cx = cminx;
                cdeltax -= cinc;
                break;
            case 0x4D00 : // Right key
                if ((cx += cinc) > cmaxx) cx = cmaxx;
                cdeltax += cinc;
                break;
            case 0x4700 : // Home key
                if ((cy -= cinc) < cminy) cy = cminy;
                if ((cx -= cinc) < cminx) cx = cminx;
                cdeltax -= cinc; cdeltay -= cinc;
                break;
        }
    }
}

```

```

    case 0x4900 :                // PgUp key
        if ((cy -= cinc) < cminy) cy = cminy;
        if ((cx += cinc) > cmaxx) cx = cmaxx;
        cdeltax += cinc; cdeltay -= cinc;
        break;
    case 0x4F00 :                // End key
        if ((cy += cinc) > cmaxy) cy = cmaxy;
        if ((cx -= cinc) < cminx) cx = cminx;
        cdeltax -= cinc; cdeltay += cinc;
        break;
    case 0x5100 :                // PgDn key
        if ((cy += cinc) > cmaxy) cy = cmaxy;
        if ((cx += cinc) > cmaxx) cx = cmaxx;
        cdeltax += cinc; cdeltay += cinc;
        break;
    default : show(); return(c);
}
show();                        // Restore mouse to possibly new position
}

return(0);                     // Tell caller not to take any action
}

// Low-level keyboard input that retrieves some of the extended
// codes produced by the arrow keys and gray keys. This function does
// not support all of the extended key codes.
int kbdmouseobj::getkb(void)
{
    int ch1, ch2;

    ch1 = getch();
    if (ch1 == 0) {              // Arrow keys have two character sequences
        ch2 = getch();          // where the first character is a zero
        ch2 = ch2 << 8;         // Combine these two characters into one
        ch2 |= ch1;             // and return the value
        return(ch2);
    }
    else
        return(ch1);
}

// Creates the image of the simulated mouse. It calls drawcursor()
// which is a routine that you must supply to actually draw the cursor
// Note: A drawcursor() routine is provided below which draws an arrow
void kbdmouseobj::initcursor(void)
{
    // Allocate space for the emulated mouse cursor and for the space
    // below it. If not enough memory, quit the program.
    cursor = malloc(imagesize(0,0,CURSWIDTH,CURSHEIGHT));
    undercursor = malloc(imagesize(0,0,CURSWIDTH,CURSHEIGHT));
    if (cursor == NULL || undercursor == NULL) {
        closegraph();
        printf("Not enough memory for program.\n");
        exit(1);
    }

    if (whichbutton == RIGHT_BUTTON || whichbutton == EITHER_BUTTON) {
        m1 = testtype;
        m2 = RIGHT_BUTTON;
    }
}

```

```

    mouseintr(m1,m2,m3,m4);
    if (m2) return(TRUE);          // Return TRUE if the action occurred
}
return(FALSE);                    // Return FALSE as a catchall
}

// Used by the simulated mouse to turn the mouse cursor on and off.
// The viewport settings may have changed so temporarily reset them
// to the full screen while the cursor image is displayed or erased.
void kbdmouseobj::togglecursor(void)
{
    struct viewporttype vp;
    int oldx, oldy;

    getviewsettings(&vp);          // Save view settings
    oldx = getx(); oldy = gety();   // and current position
    setviewport(0,0,getmaxx(),getmaxy(),1);
    if (cursoron) {                // To erase the cursor
        putimage(cx,cy,undercursor,COPY_PUT); // overwrite it with
        cursoron = FALSE;          // the saved image of
    }                               // the screen
    else {                          // To draw the cursor
        getimage(cx,cy,cx+CURSWIDTH,cy+CURSHEIGHT,undercursor);
        putimage(cx,cy,cursor,COPY_PUT);    // first save the area
        cursoron = TRUE;                 // where the cursor
    }                                     // will be and then
    // display the cursor.
    // Reset the viewport settings and its current position
    setviewport(vp.left,vp.top,vp.right,vp.bottom,1);
    moveto(oldx,oldy);
}

// This function draws a cursor if a mouse does not exist. It draws
// a small arrow. If you want a different cursor, just change this
// routine. CURSWIDTH and CURSHEIGHT define the dimensions of cursor.
void kbdmouseobj::drawcursor(int x, int y)
{
    setcolor(getmaxcolor());
    line(x+1,y+1,x+CURSWIDTH-1,y+CURSHEIGHT-1);
    line(x+2,y+1,x+CURSWIDTH-1,y+CURSHEIGHT-2);
    line(x+1,y+2,x+CURSWIDTH-2,y+CURSHEIGHT-1);
    line(x+2,y+1,x+2,y+5);
    line(x+1,y+1,x+1,y+5);
    line(x+1,y+1,x+5,y+1);
    line(x+1,y+2,x+5,y+2);
}

```

8.6 测试你的鼠标

下面的mousetst.cpp程序将部分地测试你的鼠标代码。程序只不过是把你的系统置为图形模式，然后初始化鼠标。如果发现鼠标，则显示它，而且你可以移动鼠标光标。程序将继续直到你按了鼠标的一个按钮为止。

如果没有发现鼠标，则会出现仿真鼠标。你可以使用箭头键围绕屏幕移动它。使用灰度+和-键改变移动仿真鼠标光标的量值。程序将继续，直到你按下INS或DEL键（这些是仿真鼠标按钮）。

请注意称为mouse的对象被说明为类型kbmouseobj。然后此变量用于访问鼠标工具箱的所有功能。在你的应用程序中应有类似的语句。由于此程序在mouse.cpp和kbmouse.cpp中使用鼠标实用程序，你将需要把这些文件和mousetst.cpp连接。

```
// mousetst.cpp -- This is a very simple program to test the mouse
// packages mouse.cpp and kbmouse.cpp. It will display a mouse
// cursor and allow you to move it around the screen until a mouse
// button is pressed. If no mouse is detected, the keyboard emulated
// mouse is used. Use the arrow keys to move this mouse and the INS
// or DEL keys as the mouse buttons in order to quit the program.

#include <graphics.h>
#include "mouse.h"
#include "kbmouse.h"                // Keyboard mouse functions

main()
{
    int c, gmode, gdriver=DETECT;    // Use autodetect
    kbmouseobj mouse;

    initgraph(&gdriver,&gmode,"\\tc\\bgi"); // Initialize the screen
    if (!mouse.init()) {
        mouse.hide();                // Always turn the mouse cursor
        outtextxy(0,0,"No mouse detected"); // off while writing to the
        mouse.show();                // screen
    }
    mouse.hide();
    outtextxy(0,10,"Press a mouse button to quit");
    outtextxy(0,20,"or the INS or DEL key if a mouse isn't present");
    mouse.show();                    // Wait until the mouse routine
    do {                             // returns a negative value.
        c = mouse.waitforinput(EITHER_BUTTON); // This indicates that a button
    } while (c >= 0);                 // was pressed.
    // It's a good idea to turn the mouse off when done
    mouse.hide();
    closegraph();                    // Exit graphics mode
    return(0);
}
```

第九章 使用图符

在第八章，我们学习了如何使用图形模式的鼠标。它的部分吸引力是鼠标可以建立最直观的用户接口。（以后我们将看到鼠标如何容易在屏幕上绘图。）但是，使用鼠标的另一种方式是把它作为从屏幕选择命令的指示设备。图符是在图形的模式下表示命令的流行方式。如果你曾经用过PC图形应用程序，如画画程序，则大概知道图符的好处。

在本章，我们将仔细考查图符，并在开发用于建立、保存和编辑图符的独立图符编辑程序时扩充一组图形程序设计工具。此外，我们将提供几个样本图符图案，稍后在第十二章和十三章开发定制的图形程序时我们将要使用它们。

9.1 为什么使用图符

在很多基于图形的应用中都使用图符，例如CAD和画画程序，因为它们可以把复杂的命令表示为符号或图。因此，设计出非常直观和易于使用的图形接口。代替键入命令或从往往是模糊不清的菜单表中选择，可用鼠标指出并撤按图符来调用命令。

我们的目标是开发交互地建立图符的程序。虽然可以用手工设计图符，但这个程序的交互特性简化了绘图和编辑图符的过程。此程序也可用作工具以帮助我们的试验不同的图符设计。

程序iconed.cpp（见列表9.1）非常类似于我们在第四章建立的填充图案编辑程序。差别之一是我们将使用在第八章开发的kbmouseobj类以改进用户接口。在这方面，你将能看到在前几章中开发的工具如何应用于增强其它程序。本章的第一部分集中于开发图符编辑程序，第二部分提供在后面几章的程序使用的几个图符图案。

9.2 表示图符

图符可以用很多不同的方法表示。这里我们主要关心的是开发按BGI支持的不同的图形模式显示时好看的图符。由于现有的屏幕分辨率各式各样，所以按一种模式绘画的图符若按另一模式绘制时可能有不同的外观。通常，建立一个在所有图形模式下看起来一样的图符图案几乎是不可能的。

我们将对我们的图符进行标准化，以使它们全是16像素×16像素。这样的尺寸似乎在大多数情况下都可很好地工作（至少是在我们使用的图形模式下）。不过，我们必须补偿屏幕的长宽比，以便当用不同的图形模式显示时，使图符看上去比例正确。但是，我们打算补偿图符的尺寸，所以图符将以不同的大小出现，并在每一图形模式下形状稍有不同。

从内部看，用我们的图符编辑程序把图符图案表示为二维的字节数组。此数组的说明如下所示：

```

const int ICONWIDTH = 16;           // An icon is a 16 x 16 pattern
const int ICONHEIGHT = 16;
unsigned char icon[ICONHEIGHT][ICONWIDTH]; // Holds 16 x 16 icon pattern

```

常数ICONHEIGHT和ICONWIDTH分别规定了图符的长度和宽度，并说明为16。因此，图符数组中的每一单元代表图符图案中的1个像素。我们将仅使用黑白图符，所以，这些字节单元将仅取1或0值。如果值是1，则对应的图符像素以白色绘出，否则它留下背景颜色。如果你要求，也可以把彩色添加到图符中；但是请记住，不是所有的图形模式都可以产生同一的颜色。

9.3 保存图符

为简化我们的应用程序和使图符易于维护，我们将把每一图符图案保存在它自己文件中。此外，我们将以正文格式存放图符，以便能很容易查看它们。图符文件中的格式由两个主要部分组成——标题和实体。标题设置在文件的顶部并且是规定图符的宽和高的单一行。因为我们的图符全是16像素×16像素的，所以第一行总是以这两个数字开头。余下的文件包含组织它们的图符图案，使得图符的每一行在一单独行上。因此，16×16图符每行有16个数，其中图符映像的每一像素用0或1表示。图9.1示出一个样本图符和用于存放它的文件。如果你比较这两者，会看到在图符中每一像素集和文件中的每一个1值之间有一一对应的关系。

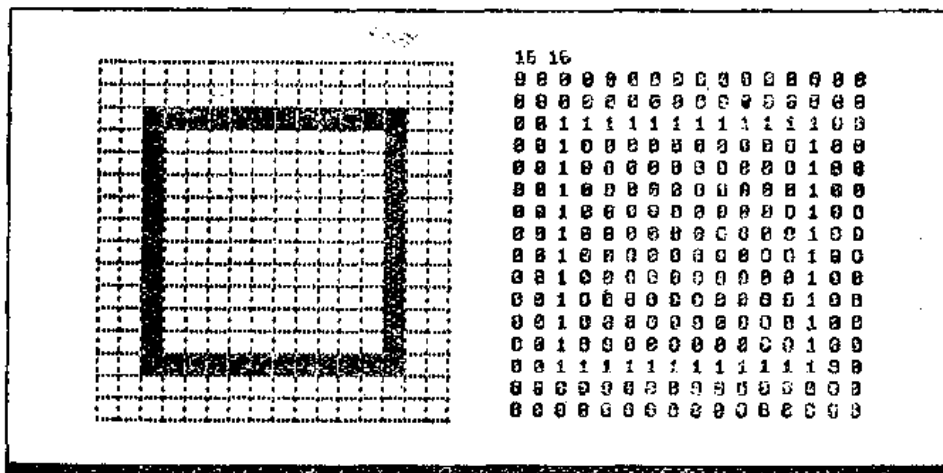


图 9.1 图符和表示它的文件

当讨论图符文件的格式时，让我们看看函数save_icon()，它用于把图符图案书写到磁盘上。为简化用户的输入，假定程序以正文方式调用save_icon()函数，它首先提示你键入要书写的图符文件的文件名。然后它调用fopen()打开此文件。如果文件不能被打开或建立，则fopen()返回NULL，并且函数立即返回而不执行任何其它动作。但是，如果文件被打开，则函数使用以下语句，把存放在数组icon中的图符图案书写到文件中，

```

// Write the header to the file:
fprintf(iconfile, "%d %d\n", ICONWIDTH, ICONHEIGHT);
for (j=0; j<ICONHEIGHT; j++) {           // Write the icon

```

```

    for (i=0; i<ICONWIDTH; i++)          // pattern to a file
        fprintf(iconfile,"%x ",icon[j][i]); // one row at a time
    fprintf(iconfile,"\n");
}
fclose(iconfile);

```

此代码中的第一个fprintf()把包含图符的宽和高的标题数据写到文件中。跟着的嵌套for循环通过书写在单独行上的每一行把图符图案拷贝到文件中。书写完图符以后，调用fclose()关闭该文件。

9.4 读图符文件

读图符文件到图符编辑程序的过程类似于书写文件，并由函数read_icon()执行。像save_icon()那样，为简化正文输入，此函数被设计成按正文模式运行。

函数从初始化图符图案开始，使得它包含所有的0，如下所示：

```

for (j=0; j<ICONHEIGHT; j++) { // Initialize the icon array
    for (i=0; i<ICONWIDTH; i++) // to all zeroes
        icon[j][i] = 0;
}

```

这种技术确保图符图案以一个全黑的图符开始。然后通过显示一个程序标志和问你是否想编辑现存图符文件使read_icon()继续。如果你的回答是yes（除字母n以外的字母），则read_icon()假定你想要编辑现有的图符并询问你用以存放它的文件名。请记住，在此过程中，以正文模式执行数据键入操作以便简化代码。但是，你可以试试把在第四章开发的gtext.cpp正文处理实用程序综合到函数中，目的在于避免在图形和正文之间变换。

一旦确定了文件名，则read_icon()试图打开它以便读。如果此操作成功，则读包含标题的文件第一行。只要此行包括等于ICONWIDTH和ICONHEIGHT的两个值，则read_icon()继续。如果发现其它的数，则文件无效且read_icon()打印信息并返回到函数main()。

用两个for循环把图符图案从文件读到数组icon，就像在save_icon()把图符数据书写到文件中那样。差别在于这里的fscanf()函数用于从文件读图符图案。一旦已把图符图案读到数组icon，则我们可以显示它。在本章后面将看到这是如何完成的。

9.5 交互编辑程序

现在已知道如何读和写图符文件，我们准备看看图符编辑程序。图9.2示出正用于编辑图符的编辑程序。请注意，在显示时有两个主要的区域。屏幕左边是正编辑的图符的放大表示。所有的编辑实际上在此窗口里执行。屏幕的右边是以它的实际大小示出的图符图案的当前状态。

你可以通过在放大的图符图案的某些大图符位上撒按鼠标来编辑图符。当每次经过这些大的像素之一时你撒按鼠标，像素的状态就从白转换到黑或从黑转换到白。此外，每次鼠标动作之后，在屏幕的右边修改图符的当前状态。

你可以通过转换各个想在放大的图符图案中改动的像素来使用图符编辑程序，直到你对正常大小的图符外观感到满意为止。为退出编辑过程，你可选择ESC键。一旦这样做，

将清除屏幕，并问你是否想把当前在编辑程序中的图符图案保存到你选择的文件中。如果是，你将被提示键入文件名，然后程序将保存图符图案并终止。否则，程序就简单地结束。现在让我们考查图符编辑程序的不同成分。

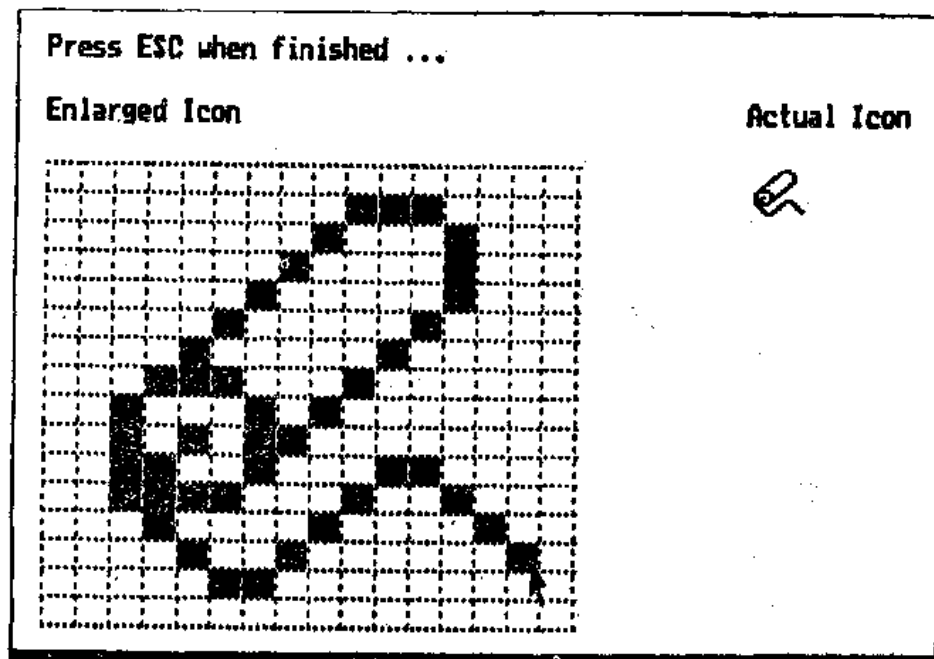


图 9.2 使用中的图符编辑程序

9.5.1 建立屏幕

iconed程序中的前10个语句初始化图符编辑程序的各个方面。进程由读图符文件开始，如果必要的话，初始化图形模式，初始化鼠标，生成放大的图符图案，并显示图符图案的初始状态。使用的函数调用如下：

```
read_icon();           // If a person wants an icon file to
init_graphics();       // be read, read it, otherwise
mouse.init();          // simply initialize the screen with
draw_enlarged_icon();  // an empty big icon pattern
show_icon();           // Draw the icon
```

必须按此顺序调用这些语句。特别是，我们需要保持正文模式中的read_icon()函数（在图形初始化之前），以便简化用户的交互动作。此外，鼠标初始化必须总是在图形初始化之后，最后两个语句的排序大概更明显些。在本节的后面我们将考查这两个例程。

下面的三个语句显示使用outtextxy()函数的一系列标志。请注意，围绕屏幕输出语句的一对mouse.hide()和mouse.show()函数，必须保证当屏幕正用显示的信息修改时关闭鼠标光标。

9.5.2 建立放大的图符

函数draw_enlarged_icon()生成屏幕左边的网格，在其上编辑放大的图符图案。网格由17个水平和垂直虚线组成，标出如图9.3中所示的图符图案的16×16网格，这些网格由下面的两个for循环画出：

```

mouse.hide();           // Draw vertical and horizontal dashed
for (i=0; i<=ICONHEIGHT; i++) // Lines to make the big icon pattern
    line(BIGICONLEFT,BIGICONTOP+i*(BIGBITSIZE+NORM_WIDTH),right,
        BIGICONTOP+i*(BIGBITSIZE+NORM_WIDTH));
for (i=0; i<=ICONWIDTH; i++)
    line(BIGICONLEFT+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),BIGICONTOP,
        BIGICONLEFT+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),bottom);
mouse.show();

```

再说一遍，请注意，无论把什么书写到屏幕以前，首先要通过调用mouse.hide()关闭鼠标，而稍后通过调用mouse.show()来恢复，

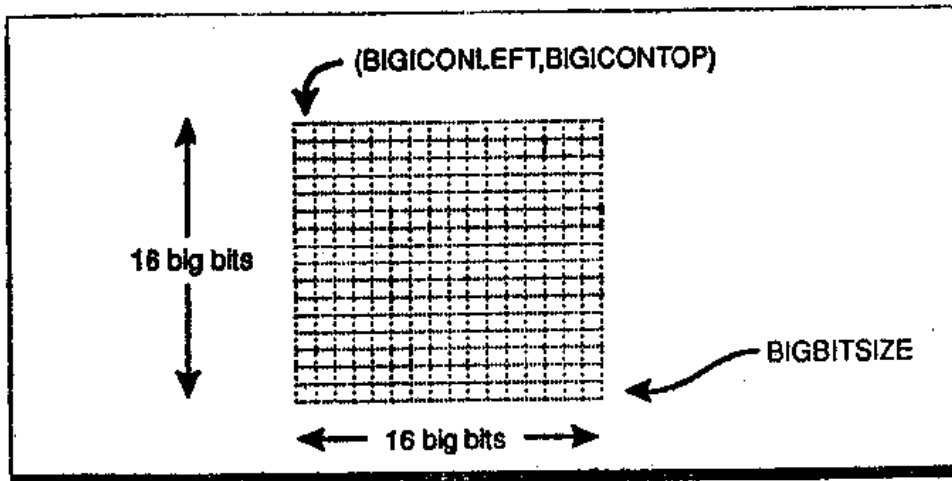


图 9.3 在iconed.cpp中使用的编辑网格

到目前为止，你可能发现draw_enlarged_icon()依赖很多常数。表9.1中示出这些常数以及它们的含义。你可能想要引用此表以帮助你了解draw_enlarged_icon()函数。

表 9.1 画放大的图符图案时所用的常数

常数	描 述
BIGICONLEFT	大图符图案起始的列
BIGICON	大图符图案的顶行
BIGBITSIZE	在放大的图符图案中的大像素的尺寸
ICONWIDTH	图符的宽度
ICONHEIGHT	图符的高度
DOTTED_LINE	来自graphics.h, 指定线型
NORM_WIDTH	来自graphics.h, 线的像素宽度

draw_enlarged_icon()的最后一行调用函数init_biglit(), 它建立一个放大的图符像素的映像。为了转换在放大图符图案中的图符像素，我们将把“异或”init_bigbit()函数产生的映像放到放大的图符图案的矩形区域中。现在看看此函数。

在init_bigbit()的开头，通过下面的嵌套for循环建立放大位之一的映像。

```

for (j=bby+1; j<=bby+BIGBITSIZE; j++) {           // Draw the big bit one
    for (i=bbx+1; i<=bbx+aspect*BIGBITSIZE; i++) // pixel at a time
        putpixel(i,j,getmaxcolor());
}

```

C
+
+
这两个循环在放大的图符图案的左上角画出一块像素，大小为BIGBITSIZE。实际上，放大图符的宽度BIGBITSIZE由屏幕的长宽比调整（用包含在变量aspect中的屏幕长宽比乘以它）。此变量在init_graphics()中由以下语句计算：

```
getaspectratio(&xasp,&yasp);  
aspect = yasp / xasp;
```

我们这样做是为了调整图符编辑程序可能使用的不同图形模式的长宽比。

一旦建立了放大的图符像素映像，则把它拷贝到数组bigbit中。但是，必须首先调用malloc()分配此映像的空间。分配空间以后，调用getimage()以便拷贝大像素的映像，如下面所示：

```
bigbit = malloc(imagesize(bbx,bby,bbx+aspect*BIGBITSIZE,bby+BIGBITSIZE));  
getimage(bbx+1,bby+1,bbx+aspect*BIGBITSIZE,bby+BIGBITSIZE,bigbit);
```

往后，当编辑时想转换放大图符像素之一时，则仅需要把存放在bigbit的这个像素图案映像“异或”到网格的适当位置上。在必须除去的放大图符像素的左上角有一个大像素。这是通过调用带XOR_PUT置换的putimage()来完成的，如下所示：

```
putimage(bbx+1,bby+1,bigbit,XOR_PUT);
```

完成这些操作以后，通过调用mouse.show()和init_bigbit()最后恢复鼠标光标。

9.5.3 显示原始图符

屏幕初始化进程的下一步是显示在程序开始时读入的图符图案。请记住，如果没有读入图符文件，则在read_icon()中图符数组被初始化为全0。函数show_icon()用于显示图符图案的当前状态。此函数由顺序通过数组icon的两个嵌套for循环组成。对于每一存放1的字节单元，在放大的图符中转换相应的大位。而小的图符图案同样也更改。设置一个大图符像素是在适当的单元上“异或”大图符像素映像的问题。在内层for循环里设置的putimage()函数执行这一步。

为设置在小图符图案中的适当像素，要求调用toggle_icons_bit()。该函数接受在icon数组中的一个像素的逻辑下标作为变元，并通过测试它是否等于背景颜色来检验与icon数组下标相对应的像素。依据它的当前值，转换小的图符像素。在由ICONLEFT上指出的列上显示小的图符。它的顶行与BIGICONTOP相符。请注意，虽然图符表示为16×16模式，但小的图符可以画得宽些。这是通过把在其宽度上画出的像素数乘以屏幕的长宽比来完成的。请回忆一下，这种技术用于调整我们使用的大多数图形模式的长宽比。

9.5.4 与用户进行交互

初始化屏幕之后，图符编辑程序可以开始工作。当编辑图符时，可接受两类输入：

- (1) 按左鼠标按钮——转换指向的当前图符像素（如果有的话）。
- (2) 按ESC退出该程序。

在函数main()中的while循环支持此用户交互，如下所示：

```

while ((c=mouse.waitforinput(LEFT_BUTTON)) != ESC) {
    // Get input from mouse/keyboard. If input is ESC, quit program.
    if (c < 0) { // If input < 0 then a mouse button
        mouse.getcoords(x,y); // has been pressed, get current
        toggle_bigbit(x,y); // coordinates and toggle big bit
    } // Loop until person types Esc
}

```

循环中心围绕着使用mouseobj的成员函数waitforinput()（我们在第八章开发的）如果按了指定的按钮（在此情况下是左鼠标按钮），则此函数返回一个负值。否则，它返回按键的值。编写while循环，使得程序将继续，直到按ESC键为止。如果按左鼠标按钮，则例程将进行到if语句。此时，getcoords()检索鼠标光标的当前位置，并把它传送给函数toggle_bigbit()。该函数改变鼠标指向的图符像素的设置。

9.5.5 转置图符像素

转置正在编辑的图符中的像素包含以下三个操作：

- (1) 必须改变图符数组中的像素的值。
- (2) 必须转置放大图符图案中的大像素映像。
- (3) 必须更新小图符图案中的图符的像素。

这些动作的每一个都是通过调用toggle_bigbit()来调动的。此函数取两个变元，对应于要改变的放大图符位的屏幕坐标。此屏幕坐标由按下按钮时鼠标光标的位置确定。鼠标坐标来自mouseobj的成员函数getcoords()。

在判断应转置哪一个图符位时（如果有的话），涉及下面所示的toggle_bigbit()的函数体：

```

for (j=0; j<ICONHEIGHT; j++) {
    line1 = BIGICONTOP+j*(BIGBITSIZE+NORM_WIDTH);
    line2 = BIGICONTOP+(j+1)*(BIGBITSIZE+NORM_WIDTH);
    if (line1 <= y && y < line2) {
        for (i=0; i<ICONWIDTH; i++) {
            col1 = BIGICONLEFT+aspect*(i*(BIGBITSIZE+NORM_WIDTH));
            col2 = BIGICONLEFT+aspect*((i+1)*(BIGBITSIZE+NORM_WIDTH));
            if (col1 <= x && x < col2) {
                mouse.hide(); // Toggle the big bit using
                putimage(col1+1,line1+1,bigbit,XOR_PUT); // the XOR
                mouse.show(); // feature of putimage()
                toggle_icons_bit(i,j); // Toggle the corresponding
                return; // pixel in the small icon
            }
        }
    }
}

```

这由函数中的两个for循环完成，它们顺序通过大图符图案的单元并检测传送给toggle_bigbit()的坐标是否落在大图符图案的任一行或任一列中。如果是，则用putimage()函数“异或”图符图案的当前单元中早些时候产生的bigbit的映像。这一过程负责转置在大图符图案中的图符像素。现在，我们必须修改小图符图案中的对应位。幸好，设计的例程能使由for循环确定的行数和列数对应于可以存取图符数组中相同位的下标。存储在变量i和j中的这些值被传送到另一函数toggle_icon_bit()，以便真正改变icon数组值并更新屏幕上的小图符

此函数在本章介绍过。

9.5.6 退出图符编辑程序

控制用户交互的while循环继续到按ESC键为止。一旦该过程完成，鼠标光标就失效，屏幕返回到正文模式，并提示你保存当前图符编辑程序中的图符。如果你用与字母n不同的任何字母应答，则程序进入save_icon()函数，并将向你提示输入文件名以存储图符（save_icon函数也在本章较早讨论过）。

9.5.7 编译此程序

为编译图符编辑程序你需要以下一些文件：

主文件	标题文件	章
mouse.cpp	mouse.h	8
kbdmouse.cpp	kbdmouse.h	8
iconed.cpp	None	9

9.5.8 样本图符

图9.4中的图符全都是用本章的图符编辑程序开发的。我们在后面的程序中将使用许多这样的图符图案。为使混淆减至最少，我们提供你也应采用的这些图符的专用文件名。这将保证与我们今后开发的任一代码最大兼容。

• 列表9.1 iconed.cpp

```
// iconed.cpp -- An icon editor. This program enables you to
// interactively create an icon, edit an existing one, or save
// an icon pattern to a file so that it can later be used by
// a graphics program. Mouse interaction is supported. Supports
// CGAHI, EGA, and VGA modes. The program uses icon files that
// are of the format:
//      ICONWIDTH ICONHEIGHT
//      one row of icon pattern
//      next row of icon pattern
//      .
//      last row of icon pattern
#include <stdio.h>
#include <graphics.h>
#include <stdarg.h>
#include <alloc.h>
#include <conio.h>
#include <process.h>
#include "mouse.h"           // The mouse and keyboard routines
#include "kbdmouse.h"

const int BIGICONLEFT = 20; // The left side of the big icon pattern
const int BIGICONTOP = 50;  // The top side of the big icon pattern
const int BIGBITSIZE = 8;   // The big bits are 8 pixels in size
const int ICONWIDTH = 16;   // An icon is a 16 x 16 pattern
const int ICONHEIGHT = 16;
const int ICONLEFT = 400;    // The small icon pattern is located here
```

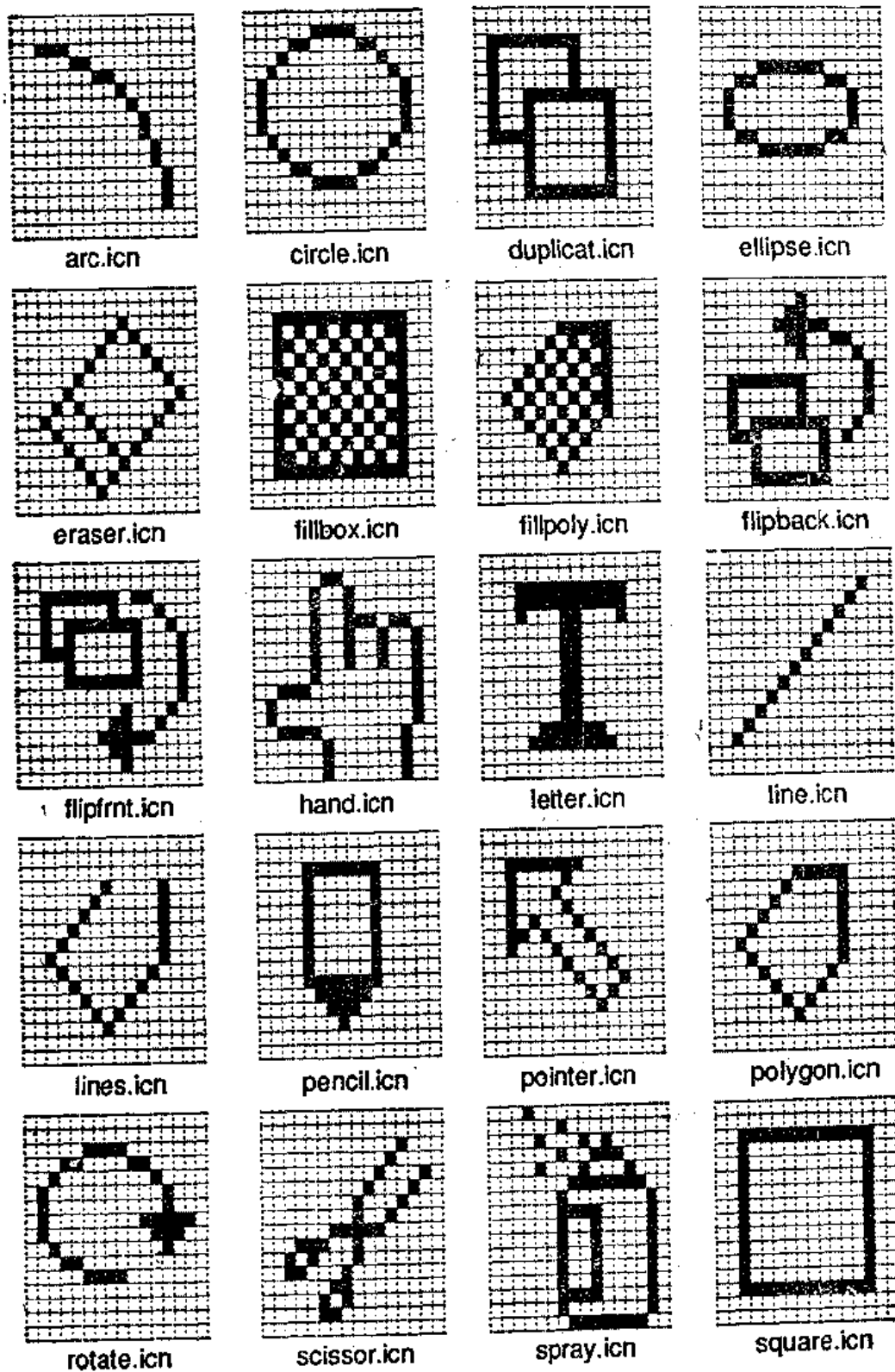


图 9.4 样本图符

```

const int ESC = 27;          // The value of the ESC key

// These are the functions in iconed.cpp
void draw_enlarged_icon(void);
void toggle_bigbit(int x, int y);
void toggle_icons_bit(int x, int y);
void init_bigbit(void);
void toggle_cursor(int x, int y);
void save_icon(void);
void read_icon(void);
void init_graphics(void);
void show_icon(void);

// The global variables
void *bigbit;                // Points to image of a big bit
unsigned char icon[ICONHEIGHT][ICONWIDTH]; // Holds 16 x 16 icon pattern
kbdmouseobj mouse;
int aspect;

main()
{
    int x, y, c;

    read_icon();              // If a user wants an icon file to
    init_graphics();          // be read, read it, otherwise

    mouse.init();             // simply initialize the screen with
    draw_enlarged_icon();     // an empty big icon pattern
    show_icon();              // Draw the icon
    mouse.hide();             // When using the mouse, first turn
                                // it off before writing to screen
    outtextxy(BIGICONLEFT,10,"Press ESC when finished ...");
    outtextxy(BIGICONLEFT,BIGICONTOP-20,"Enlarged Icon");
    outtextxy(ICONLEFT,BIGICONTOP-20,"Actual Icon");
    mouse.show();             // Redraws mouse to the screen
    while ((c=mouse.waitforinput(LEFT_BUTTON)) != ESC) {
        // Get input from mouse/keyboard. If input is ESC, quit program.
        if (c < 0) {           // If input < 0 then a mouse button
            mouse.getcoords(x,y); // has been pressed, get current
            toggle_bigbit(x,y);   // coordinates and toggle big bit
        }                      // Loop until user types ESC
    }                          // Turn mouse off and then get out
    mouse.hide();             // of graphics mode to make user
    closegraph();             // input easier for filename
    printf("Do you want to save this icon to a file? (y) ");
    if (getch() == 'n')        // Save the icon to a file if the
        save_icon();          // user types anything except 'n'
    return(0);
}

// This routine draws an enlarged view of the icon pattern being
// edited. The user clicks on the big bits in this pattern to toggle
// the corresponding pixels on and off in the actual icon. The
// enlarged icon is drawn at BIGICONLEFT, BIGICONTOP, to right,
// bottom.
void draw_enlarged_icon(void)
{
    int i, right, bottom;

    setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
    right = BIGICONLEFT+ICONWIDTH*aspect*(BIGBITSIZE+NORM_WIDTH);

```

```

bottom = BIGICONTOP+ICONHEIGHT*(BIGBITSIZE+NORM_WIDTH);
mouse.hide(); // Draw vertical and horizontal dashed
for (i=0; i<=ICONHEIGHT; i++) // lines to make the big icon pattern
    line(BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),BIGICONTOP,
        BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),
        BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH)).right);
for (i=0; i<=ICONWIDTH; i++)
    line(BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),BIGICONTOP,
        BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH)),bottom);
mouse.show();
init_bigbit(); // Create the big bit image
}

// Create the image of a single big bit. This image will be used to
// toggle the big bits later in the program whenever the user clicks
// on the big icon pattern.
void init_bigbit(void)
{
    int bbx, bby, i, j;

    bbx = BIGICONTOP; // Create the big bit image in the top-left
    bby = BIGICONTOP; // corner of the big icon already drawn
    mouse.hide(); // Hide the mouse before drawing to the screen
    for (j=bby+1; j<=bby+BIGBITSIZE; j++) { // Draw the big bit one
        for (i=bbx+1; i<=bbx+aspect*BIGBITSIZE; i++) // pixel at a time
            putpixel(i,j,getmaxcolor());
    }
    // Set aside enough memory for the big bit image and then use
    // getimage() to capture its image
    bigbit = malloc(imagesize(bbx,bbx+aspect*BIGBITSIZE,
        bby+BIGBITSIZE));
    getimage(bbx+1,bby+1,bbx+aspect*BIGBITSIZE,bby+BIGBITSIZE,bigbit);
    // Erase the big bit by exclusive ORing it with itself
    putimage(bbx+1,bby+1,bigbit,XOR_PUT);
    mouse.show(); // Turn the mouse back on
}

// When the user clicks on the big icon pattern, toggle the
// appropriate big bit and the corresponding pixel in the icon
// pattern. This routine accepts screen coordinates that specify where
// the mouse button was pressed. The two for loops test to see which
// logical bit in the icon pattern must be toggled. Use putimage() to
// toggle the bigbit. Calls toggle_icons_bit() to toggle the
// appropriate pixel in icon pattern.
void toggle_bigbit(int x, int y)
{
    int i, j, line1, line2, col1, col2;

    for (j=0; j<ICONHEIGHT; j++) {
        line1 = BIGICONTOP+j*(BIGBITSIZE+NORM_WIDTH);
        line2 = BIGICONTOP+(j+1)*(BIGBITSIZE+NORM_WIDTH);
        if (line1 <= y && y < line2) {
            for (i=0; i<ICONWIDTH; i++) {
                col1 = BIGICONTOP+aspect*(i*(BIGBITSIZE+NORM_WIDTH));
                col2 = BIGICONTOP+aspect*((i+1)*(BIGBITSIZE+NORM_WIDTH));
                if (col1 <= x && x < col2) {
                    mouse.hide(); // Toggle the big bit using
                    putimage(col1+1,line1+1,bigbit,XOR_PUT); // the XOR
                    mouse.show(); // feature of putimage()
                    toggle_icons_bit(i,j); // Toggle the corresponding
                    return; // pixel in the small icon
                }
            }
        }
    }
}

```



```

    }
}

// This routine toggles a single pixel in the icon pattern. It changes
// the pixel's color and its value in the icon pattern array. The
// arguments x and y are between 0 and ICONWIDTH or ICONHEIGHT. Plot
// 2 pixels wide to adjust for the aspect ratio of many of the
// graphics modes. You may want to remove this. The array icon saves
// the icon pattern. If one of its locations is 1, then the
// corresponding pixel in the icon is to be drawn on.
void toggle_icons_bit(int x, int y)
{

```

```

    int i;

    mouse.hide();
    // Switch the color of the pixel
    if (getpixel(aspect*x+ICONLEFT,BIGICONTOP+y) != BLACK) {
        for (i=0; i<aspect; i++)
            putpixel(aspect*x+i+ICONLEFT,BIGICONTOP+y,BLACK);
        icon[y][x] = 0;
    }
    else {
        // Draw all pixels on with the max color
        for (i=0; i<aspect; i++)
            putpixel(aspect*x+i+ICONLEFT,BIGICONTOP+y,getmaxcolor());
        icon[y][x] = 1;
    }
    mouse.show();
}

```

```

// This routine writes the icon pattern to a file. The user is
// prompted for the filename to write the file to. The format of
// the file is given at the top of this program
void save_icon(void)
{

```

```

    char filename[80];
    FILE *iconfile;
    int i, j;

    printf("\nEnter the file name to store the icon in: ");
    scanf("%s", filename);
    if ((iconfile = fopen(filename, "w")) == NULL) {
        printf("Could not open file.\n");
        return;
    }

    // Write the header to the file:
    fprintf(iconfile, "%d %d\n", ICONWIDTH, ICONHEIGHT);
    for (j=0; j<ICONHEIGHT; j++) {
        for (i=0; i<ICONWIDTH; i++)
            fprintf(iconfile, "%x ", icon[j][i]);
        fprintf(iconfile, "\n");
    }
    fclose(iconfile);
}

```

```

// This routine reads an icon file into the icon pattern array and
// calls show_icon() to turn the appropriate pixels on. If the header
// of the file does not match the ICONWIDTH and ICONHEIGHT values,
// then the file is invalid and no icon will be read in. If the
// user does not want a file read in or there is an error reading

```

```

// the icon file, the function will return with the icon pattern
// initialized to all blanks
void read_icon(void)
{
    char filename[80];
    FILE *iconfile;
    int i, j, width, height;

    for (j=0; j<ICONHEIGHT; j++) { // Initialize the icon array
        for (i=0; i<ICONWIDTH; i++) // to all zeroes
            icon[j][i] = 0;
    }
    printf("\n\n----- ICON EDITOR ----- \n\n");
    printf("Do you want to edit an existing icon? (y) ");
    if (getch() == 'n') return;
    printf("\nEnter the name of the file to read the icon from: ");
    scanf("%s", filename);
    if ((iconfile = fopen(filename, "r")) == NULL) {
        printf("Cannot open file. -- Press any key to continue\n");
        getch();
        return;
    }
    // Read the first line of the icon file. It should contain two
    // numbers that are equal to ICONWIDTH and ICONHEIGHT.
    fscanf(iconfile, "%d %d", &width, &height);
    if (width != ICONWIDTH || height != ICONHEIGHT) {
        printf("Incompatible icon file. -- Press any key to continue\n");
        return;
    }
    for (j=0; j<ICONHEIGHT; j++)
        for (i=0; i<ICONWIDTH; i++)
            fscanf(iconfile, "%x", &icon[j][i]);
    fclose(iconfile);
}

void init_graphics(void)
{
    int xasp, yasp, gmode, gerror, gdriver = DETECT;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    if ((gerror = graphresult()) < 0) {
        printf("\nFailed graphics initialization: gerror=%d\n", gerror);
        exit(1);
    }
    getaspectratio(&xasp, &yasp);
    aspect = yasp / xasp;
}

// Show the icon pattern that is in the icon array
void show_icon(void)
{
    int x, y;

    for (y=0; y<ICONHEIGHT; y++)
        for (x=0; x<ICONWIDTH; x++) {

            // For every icon bit that is on, display it in
            // the big and small icons
            if (icon[y][x] == 1) {
                outimage(BIGICONLEFT+aspect*(x*(BIGBITSIZE+NORM_WIDTH))+1,

```

```
        BIGICONTOP+y*(BIGBITSIZE+NORM_WIDTH)+1,
        bigbit.XOR_PUT);
toggle_icons_bit(x,y);
    )
)
```

第十章 图形中的上弹窗口

本章介绍常常在交互图形环境中出现的另一有用特征——上弹窗口。像在第九章探讨的图符一样，上弹窗口在最近几年已流行。它们是有吸引力的，因为它们能用于铺设信息，建立下拉菜单和充实其它形式的I/O程序。

在本章中，我们的目标是开发上弹窗口程序包，它可以用在由BGI支持的任一图形模式中。这个称为gpopup.cpp的窗口程序包允许我们用两条简单命令在图形模式中上弹和除去窗口。我们这里开发的窗口工具将与在十二、十三、十四章的程序一起使用。

10.1 基本方法

由于低级BGI屏幕工具的帮助，在Turbo C++中易于支持上弹窗口。在本节，我们将向你说明建立和除去上弹窗口的各个步骤。在列表10.1和10.2中示出了gpopup窗口程序包gpopup.h和gpopup.cpp的源文件。

有关我们开发的窗口系统最独特的是什么？那就是面向对象。系统的主要积木块是类程gwindows。此类程包含用于管理显示和除去窗口的它自己的窗口堆栈，存储为处理图形窗口所需数据的一组变量，以及为建立、显示和除去上弹窗口的一组专用的成员函数。

10.1.1 介绍gwindows类程

既然我们的窗口系统的核心是gwindows类程，现在我们将介绍它：

```
class gwindows {
public:
    gwindows(void);
    int gpopup(int left, int top, int right, int bottom, int bordertype,
               int bordercolor, int backfill, int backcolor);
    int gunpop(void);
    void unpopallwindows(void);
    virtual void paintwindow(int left, int top, int right, int bottom);
private:
    GRAPHICSWINDOW *wstack[NUMWINDOWS]; // The window stack
    int wptr; // Points to next free stack location
    int savewindow(int left, int top, int right, int bottom);
};
```

第一个成员函数gwindow() 是用以初始化窗口对象的构造器。它负责执行一个重要的任务——设置堆栈指针wptr，使得它指向堆栈的底部。gpopup() 成员函数是系统的真正工作间，因为它负责分配图形上弹窗口的存储和把有关窗口的信息存放在堆栈上。gpopup() 的配对函数是gunpop()，它重分配窗口的存储和把它从屏幕除去。如果你想要除去当前显示的所有窗口，则可以调用unpopallwindows() 成员函数。为实际绘画窗口，应使用paintwindow()。

gwindows类程的私有区段部分由两个变量和一个支持窗口堆栈的成员函数组成。当

我们考查窗口堆栈如何工作时，将更为详细地探讨这些成分。

10.1.2 上弹窗口

图10.1描绘了建立上弹窗口的过程。如图中所示，基本思想是保留被上弹窗口覆盖的屏幕区以及某些可能改变的屏幕和绘图参数，以便过后可用此信息把屏幕恢复为建立上弹窗口以前的状态。我们将把此信息保存在堆栈，使得我们可以建立上弹窗口的各层。遗憾的是，堆栈方法限制存取窗口。也就是说，窗口应以它们建立的逆序除去。因此，我们不能任意地把由若干其它窗口覆盖的窗口移到屏幕的前面。实际上，可以增加此特征，但为保持代码简单而省略了。

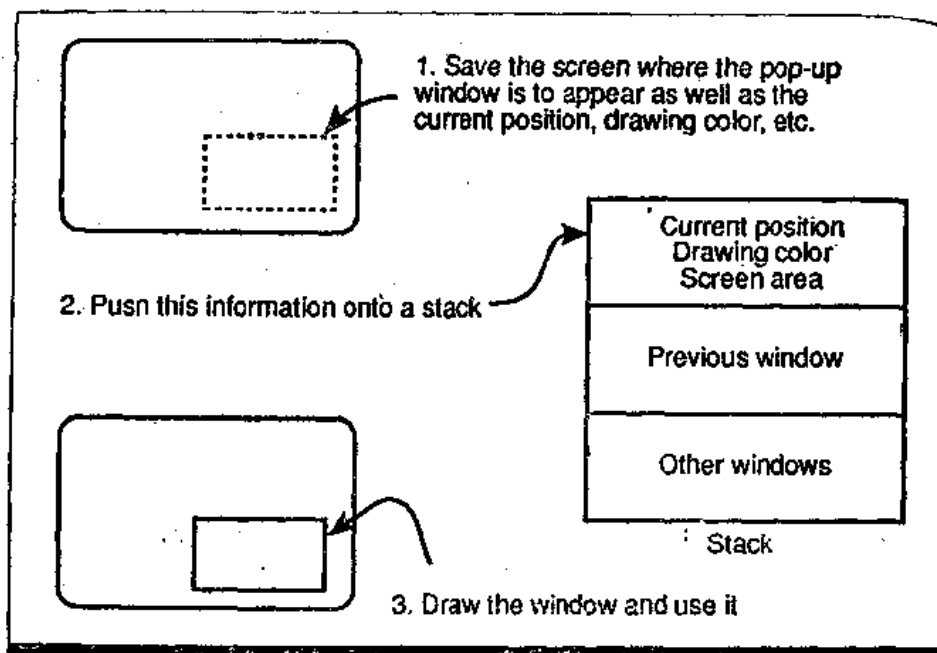


图 10.1 建立上弹窗口的步骤

类似地，为从屏幕除去上弹窗口，我们必须执行以下两步：

- (1) 通过从堆栈上托所存放的屏幕映像重写上弹窗口。
- (2) 恢复保存在堆栈上的屏幕参数。

此过程之后，就像从未有过上弹窗口那样的情况重新出现。

我们使用由BGI提供的getimage ()和putimage () 函数就可保存和恢复屏幕映像。由于这些函数可在BGI支持的所有图形模式下工作，所以上弹窗口也可以这样。保存的屏幕参数包括像当前绘画位置和颜色那样的属性。我们必须保留这些属性，因为上弹窗口可能改变它们。

10.1.3 使用堆栈

维护上弹窗口所使用的主要数据结构是堆栈。为简化起见，我们把堆栈作为单维数组如图10.2示出的那样实现，其中数组的每一元素存储早先讨论的单个上弹窗口的信息。请注意，堆栈指针实际上是数组的下标，通常指向堆栈的下一个可用单元。每次下推一个新窗口到堆栈时它增加1。因此，如果在堆栈上有四项，则堆栈指针将指向第四个元素(请

记住，C++中的数组以0开始）。

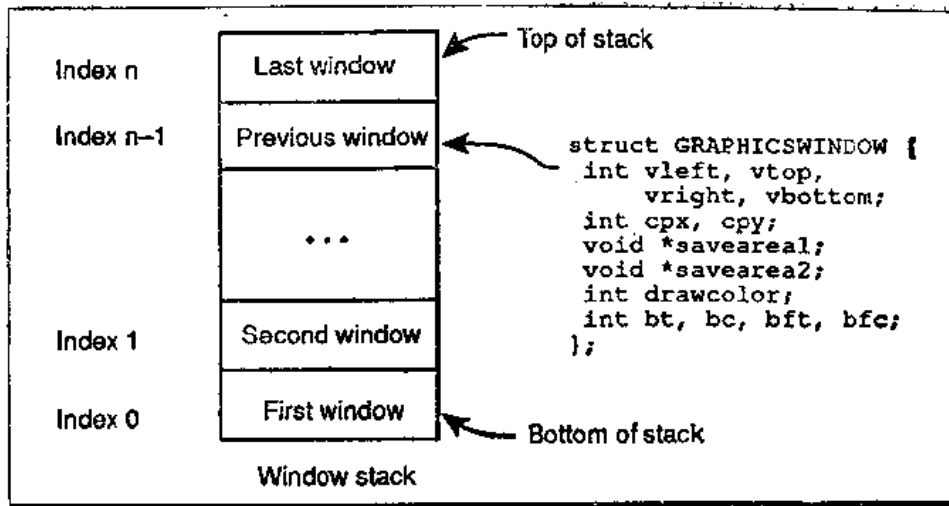


图 10.2 窗口堆栈是一个指向结构的指针数组

数组中的每一元素定义为对结构的指针。结构用于保存观察端口的设置、当前位置、绘图参数和窗口下面的屏幕区域。此结构在gpopup.cpp中定义为：

```
struct GRAPHICSWINDOW {           // Structure to save graphics settings
    int vleft,vtop,vright,vbottom; // Pixel boundaries of parent window
    int cpx,cpy;                   // Cursor location in parent window
    void *savearea1;               // Pointers used to save the area
    void *savearea2;               // that is overwritten by the window
    int drawcolor;                 // Current drawing color
    int bt, bc, bft, bfc;          // Border type and color. Background
                                   // type and color.
};
```

这个结构用于保存当窗口上弹时可能改变的任何东西。当上弹窗口从屏幕除去时，这些属性用于恢复屏幕设置为原始状态。在结构的顶部开始，vleft、vtop、vright和vbottom保存在建立上弹窗口时当时的观察端口边界。类似地，cpx和cpy保存当前位置的坐标。savearea1和savearea2变量是保存的屏幕区的指针。正好在书写窗口以前分配保存屏幕的存储。最后，在结构成员drawcolor中保存绘图颜色，而在变量bt、bc、bft和bfc中保存边界的类型和颜色（包括背景）。取决于你想如何使用上弹窗口程序包，你可能需要修改GRAPHICSWINDOW，使得它也保存当前的线型、填充图案和其余的BGI绘图参数。

回忆一下我们的堆栈，它是指向类型结构GRAPHICSWINDOW的指针数组，在gwindows类程的私有段中定义为：

```
GRAPHICSWINDOW *wstack[NUMWINDOWS]; // The window stack
```

我们将等待为这些结构分配空间，直到需要它们时为止。按此方法，不会浪费从不使用的存储。还请注意，我们已说明10个指针（NUMWINDOWS是常数，置为10）。这意味着一次仅能有10个上弹窗口。对大多数应用而言，这应该足够了。如果你开发的应用程序需要同时打开更多的窗口，那末你应适当地增加此数目。堆栈指针wptr是指向wstack数组的下标，它也在类程gwindows中定义。我们将在下一节说明如何使用这个指针。图10.3

示出当新窗口添加到屏幕上时堆栈如何变化。

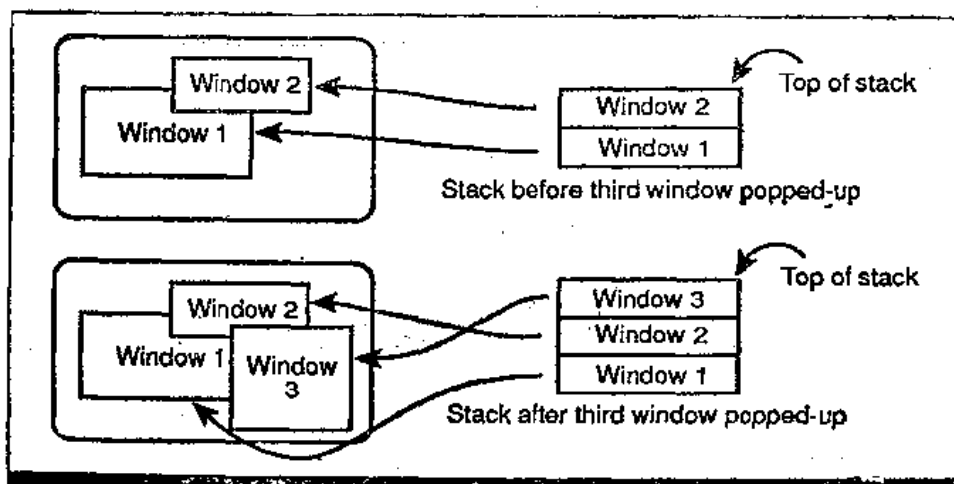


图 10.3 当把窗口添加到屏幕上时堆栈的变化

10.1.4 初始化窗口程序包

像本书中的许多其它工具一样，我们必须在使用任一上弹窗口例程以前调用初始化函数。如我们看到的，在gpopup.cpp中的窗口初始化函数是构造器gwindows()，它负责初始化指向堆栈数组第一个元素的堆栈指针：

```
wptr = 0;
```

10.1.5 上弹例程

现在让我们考查一下用于上弹窗口的成员函数gpoup()。它在文件gpoup.h中原型化为：

```
int gpoup(int left, int top, int right, int bottom, int bordertype,  
          int bordercolor, int backfill, int backcolor);
```

正如你可以看到的，gpoup()取异常多的八个变元。前四个指定上弹窗口的两个对角，并且必须是全屏幕坐标。而不是当前窗口的坐标。后四个变元定义在上弹窗口中使用的图形参数。这些都在表10.1中示出。如果你希望能通过调用gpoup()改变某些其它的图形参数，则可以容易地把它们添加到它的变元表中。例如，你可以把背景颜色或线型添加到由gpoup()保存的绘图参数的表中。

最后，如果成功地创建窗口，则gpoup()返回1；否则，返回0，于是指示在上弹窗口时失败。

现在让我们看看如何使用gpoup()上弹窗口，它从屏幕左上方向它中间伸展，有一条实边界和一个实填充图案。假定我们说明称为winobj的gwindows对象，则我们可以使用行：

```
winobj.gpoup(0,0,getmaxx()/2,getmaxy()/2,SOLID_LINE,WHITE,  
             SOLID_FILL,WHITE);
```

表 10.1 gpopup() 的变元

变元	描述
border type	定义用于画上弹窗口边界的线型
border color	置用于画边界的绘图颜色
backfill	定义用于填上弹窗口的填充图案
backcolor	置填充图案所用的颜色

10.1.6 仔细考查gpopup()

至此, 我们已有上弹窗口例程的功能性描述, 让我们仔细观察它是如何工作的。在本节中, 我们将考查gpopup() 成员函数的部分, 但是为完整列表例程, 涉及到接近本章末尾列出的gpopup.cpp源代码。

用gpopup() 所做的第一件事是检测堆栈是否填满。因为我们使用数组实现堆栈, 所以这仅仅是检查堆栈指针wptr是否已到达数组wstack末尾的简单问题。如果已到达, 则gpopup() 返回0, 指出它不能上弹窗口。请记住, 堆栈指针通常指向堆栈中下一个可用的单元; 因此, 检测堆栈是否填满是用下面的语句:

```
if (wptr >= NUMWINDOWS) return(0);
```

一旦我们知道对另一窗口堆栈有足够的空间时, 就必须对用于保存屏幕的当前状态的GRAPHICSWINDQW结构之一分配空间。这用以下两行完成:

```
wstack[wptr] = new GRAPHICSWINDOW;
if (wstack[wptr] == 0) return(0); // Not enough memory
```

请注意, 我们必须检查从调用0值的new返回的指针, 以确保存储分配没有失效。如果有问题, 则gpopup() 立即用0值返回给它的调用者。

进程中的下一步是保留观察端口和绘图参数, 因为当我们上弹窗口时可能改变它们。这在直接跟着分配GRAPHICSWINDOW结构的一些行中完成。

下面我们将通过调用成员函数savewindow()保存由上弹窗口重写的屏幕区域, 这将在下一节中讨论。但是, 首先要将观察端口设置为全屏幕, 因为我们的上弹窗口坐标是相对于全屏幕而不是当前的观察端口给出的。这允许我们把上弹窗口的位置指定在屏幕的任何地方, 而不管当前有效的是什么样的视图。gpopup() 中的if语句调用保存重写的屏幕区域的savewindow()。如果savewindow() 返回0, 则操作失败, 于是不能建立上弹窗口。如果是这种情况, 则必须恢复观察端口设置 (因为它们在调用savewindow()之前已被改变) 和必须释放已分配的GRAPHICSWINDOW结构。一旦这些完成, gpopup() 通过返回0报告失效。执行这些操作的代码是:

```
if (savewindow(left,top,right,bottom) == 0) {
    // Save screen failed. Restore viewport settings
    // and current position
    setviewport(oldview.left,oldview.top,oldview.right,
               oldview.bottom,1);
    moveto(oldx,oldy);
}
```



```

delete(wstack[wptr]):
return(0);
} // Return failure flag

```

10.1.7 保存屏幕

调用savewindow() 成员函数以保存由上弹窗口重写的屏幕区域。我们将使用getimage() 以获得此屏幕映像的副本。实际上, 我们将把要拷贝的屏幕区域划分成两个相等的部分并使用“两次”调用getimage(), 每个区域一次。之所以这样做, 是因为在某些高分辨率模式下, 如果仅使用一次调用getimage(), 则为了保存屏幕映像可能要求过多的存储(大于64K)。因此, 首先我们必须为存储屏幕映像的两个指针(savearea1和savearea2) 分配存储空间。这由调用malloc() 完成, 使用imagesize() 以确定存储的大小。

```

halfpoint = (top + bottom) / 2; // Divide the screen region
wstack[wptr]->savearea1 = // to be saved into two parts
    malloc(imagesize(left,top,right,halfpoint));
wstack[wptr]->savearea2 =
    malloc(imagesize(left,halfpoint+1,right,bottom));

if (wstack[wptr]->savearea1 == NULL || wstack[wptr]->savearea2 == NULL)
    return(0); // Not enough memory to save screen

```

再说一遍, 有可能系统超出存储和malloc() 失败。如果失败, 则savearea1或savearea2将是NULL, 并且savewindow() 将立即带0值返回。否则函数继续并作两次调用getimage() 以保存屏幕, 然后返回值1以指示savewindow() 成功。

```

// Save the screen image where the window is to appear
getimage(left,top,right,halfpoint,wstack[wptr]->savearea1);
getimage(left,halfpoint+1,right,bottom,wstack[wptr]->savearea2);
return(1); // Return a success flag

```

请记住, 使用上弹窗口函数时, 如果你试图保存太大的窗口, 则将失败。由于使用的存储指针仅是远指针而不是巨指针, 故每一个可以处理的最大区的大小是64K。于是允许我们存储至多128K屏幕。这看起来好像是许多存储, 但是EGA和VGA的较高分辨率模式可能很容易超过这个值。

10.1.8 建立上弹窗口

现在让我们回到gpopup() 成员函数的讨论。我们停留在刚刚为保存屏幕映像而调用savewindow() 的地方。往下, 把用于画上弹窗口的绘图参数保存在堆栈上。然后通过调用paintwindow() 成员函数画上弹窗口。它依次调用bar3d() 画出大多数窗口, 而不是调用bar() 或rectangle(), 因为bar3d() 支持边界和全范围填充图案的两类线型, 像从前几次所做的那样, 我们对bar3d() 使用深度0, 以便画出矩形区域。

请注意, paintwindow() 定义为虚拟函数。我们这样做, 是因为想使它便于你重设。当前它支持两类窗口: 带有简单边缘的窗口和三维外观窗口。如果你需要建立自己的窗口类型, 可以容易地建立你的paintwindow() 版本。

画完窗口以后, 以前保存在临时变量中的绘图参数的当前状态被拷贝到当前wstack结构。此外, 填充类型、线型和绘图颜色全都恢复为在绘窗口图以前的值。然后增加堆栈指针使得它指向堆栈上的下一单元, gpopup() 返回作为成功标志的1。

10.1.9 消除上弹窗口

由成员函数`gunpop()`处理从屏幕除去上弹窗口。它负责恢复被上弹窗口重写的原始屏幕映像和重置各种屏幕参数。

`gunpop()`成员函数不取任何变元并在调用时通常返回1。但是,如果堆栈是空的,则意味着没有留下任一要除去的窗口,`gunpop()`将不采取动作并返回0值。这个测试通过检查堆栈指针`wptr`是小于或等于0来完成。实际上,真正仅需测试它是否等于0,但显然这些负的堆栈指针将指示有问题,

除去上弹窗口进程的下一步是减小堆栈指针。因为正常地它指向堆栈的下一自由单元,减小堆栈指针迫使它到达显示的最顶部窗口。然后把指向此窗口的结构的指针拷贝到`W`(一个将用以参引此结构的临时指针)。

现在我们已存取窗口结构,该结构包含建立窗口以前的屏幕状态。首先,我们通过两次调用`putimage()`并使用`COPY-PUT`恢复屏幕映像。记住,我们在“两个”缓冲器(`savearea1`和`savearea2`)中保存屏幕。

```
putimage(0,0,w->savearea1.COPY_PUT);
putimage(0,(currview.bottom-currview.top)/2+1,w->savearea2.COPY_PUT);
```

请记住,当前窗口的左上坐标用作放`savearea1`映像的位置,另一个屏幕映像`savearea2`放在当前窗口下半边。因为它保存放上弹窗口的屏幕的较低部分。

`gunpop()`下面的几行重置观察端口、当前位置和建立上弹窗口以前的各种绘图参数。

最后,释放在`savewindow()`和`gpopup()`中分配的屏幕映像和窗口结构的存储。`gunpop()`返回1指示除去上弹窗口成功。

10.1.10 消除所有窗口

有时同时除去所有的上弹窗口比较方便。为此目的,包括有`unpopallwindows()`成员函数。它包含单行的行:

```
while (gunpop());
```

该行循环到`gunpop()`从屏幕除去所有上弹窗口为止。这时它将返回0。

• 列表10.1 `gpopup.h`

```
// gpopup.h -- Header file for gpopup.cpp
#ifndef GPOPUPH // Prevent recompilation
#define GPOPUPH
const int NUMWINDOWS = 10; // Allow for 10 pop-up windows
const int THREEED = 10; // Selects three-dimensional windows
struct GRAPHICSWINDOW { // Structure to save graphics settings
    int vleft,vtop,vright,vbottom; // Pixel boundaries of parent window

    int cpx,cpy; // Cursor location in parent window
    void *savearea1; // Pointers used to save the area
    void *savearea2; // that is overwritten by the window
    int drawcolor; // Current drawing color
    int bt, bc, bft, bfc; // Border type and color. Background
}; // type and color.
```

```

class gwindows {
public:
    gwindows(void);
    int gpopup(int left, int top, int right, int bottom, int bordertype,
               int bordercolor, int backfill, int backcolor);
    int gunpop(void);
    void unpopallwindows(void);
    virtual void paintwindow(int left, int top, int right, int bottom);
private:
    GRAPHICSWINDOW *wstack[NUMWINDOWS]; // The window stack
    int wptr;                          // Points to next free stack location
    int savewindow(int left, int top, int right, int bottom);
};
#endif

```

• 列表10.2 gpopup.cpp

```

// gpopup.cpp -- Pop-up graphics window package. This is a set of
// utilities that provides pop-up windows in graphics mode. The
// routines use Turbo C++'s BGI tools to simplify the code. Most of
// the graphics settings are saved before a new window is put
// up and they are restored when the window is closed. The window data
// is maintained in a stack that is part of the class, gwindows.
// The stack is implemented as an array in order to keep things simple.
// Note that these functions allow you to pop up window regions larger
// than 64K by breaking them up into smaller regions. This is useful
// for saving large pop-up windows in high-resolution graphics modes.
// To use this package, you should declare a window object of type
// gwindows. Using this object you can call gpopup() and gunpop()
// to pop up and unpop windows. Compile with large memory model.
#include <graphics.h>
#include <alloc.h>
#include "gpopup.h" // Header file for window package

// The constructor initializes the stack pointer
gwindows::gwindows(void)
{
    wptr = 0;
}

// Pushes the graphics window onto the window stack. This is an internal
// routine to save the area where the window is supposed to appear. This
// routine will return one if successful and zero if not. The latter case
// may occur if there isn't enough memory to save the screen area or
// the stack is full. Note that the saved region is divided into two
// sections so that you can save large-screen regions in high-
// resolution modes.
int gwindows::savewindow(int left, int top, int right, int bottom)
{
    int halfpoint;

    halfpoint = (top + bottom) / 2; // Divide the screen region
    wstack[wptr]->savearea1 = // to be saved into two parts
        malloc(imagesize(left, top, right, halfpoint));
    wstack[wptr]->savearea2 =
        malloc(imagesize(left, halfpoint+1, right, bottom));
    if (wstack[wptr]->savearea1 == NULL || wstack[wptr]->savearea2 == NULL)
        return(0); // Not enough memory to save screen
    // Save the screen image where the window is to appear
}

```

```

    getimage(left,top,right,halfpoint,wstack[wptr]->savearea1);
    getimage(left,halfpoint+1,right,bottom,wstack[wptr]->savearea2);
    return(1); // Return a success flag
}

// Call this function to pop up a window in graphics mode. It
// returns one if successful and zero if not. It will fail if there
// is not enough memory.
int gwindows::gpopup(int left, int top, int right, int bottom,
    int bordertype, int bordercolor, int backfill,
    int fillcolor)
{
    struct viewporttype oldview;
    int oldx, oldy, savecolor;
    struct linesettingstype saveline;
    struct fillsettingstype savefill;

    if (wptr >= NUMWINDOWS) return(0); // Stack is full
    wstack[wptr] = new GRAPHICSWINDOW;
    if (wstack[wptr] == 0) return(0); // Not enough memory
    // Save the view settings and current position so that we can temporarily
    // switch the viewport to the full screen coordinates
    getviewsettings(&oldview);
    oldx = getx();
    oldy = gety();
    setviewport(0,0,getmaxx(),getmaxy(),1);
    // Save current drawing parameters before drawing window.
    // This does not support user-defined line styles or user-defined
    // fill patterns.
    getlinesettings(&saveline);
    savecolor = getcolor();
    getfillsettings(&savefill);
    if (savewindow(left,top,right,bottom) == 0) {
        // Save screen failed. Restore viewport settings
        // and current position
        setviewport(oldview.left,oldview.top,oldview.right,
            oldview.bottom,1);
        moveto(oldx,oldy);

        delete(wstack[wptr]);
        return(0); // Return failure flag
    }
    wstack[wptr]->bc = bordercolor;
    wstack[wptr]->bt = bordertype; // Save window style
    wstack[wptr]->bft = backfill;
    wstack[wptr]->bfc = fillcolor;
    paintwindow(left,top,right,bottom);
    setviewport(left,top,right,bottom,1); // Set viewport to window
    // Save the current state of all settings on the stack
    wstack[wptr]->vleft = oldview.left;
    wstack[wptr]->vtop = oldview.top;
    wstack[wptr]->vright = oldview.right;
    wstack[wptr]->vbottom = oldview.bottom;
    wstack[wptr]->cpx = oldx;
    wstack[wptr]->cpy = oldy;
    wstack[wptr]->drawcolor = savecolor;
    // Restore drawing parameters
    setlinestyle(saveline.linestyle,saveline.upattern,saveline.thickness);
    setcolor(savecolor);
    setfillstyle(savefill.pattern,savefill.color);
    wptr++; // Increment the stack pointer
}

```

```

    return(1);                                     // Return success flag
}

// Draw the window using the styles specified. This member function
// is made virtual so that you can override it to define your own
// window styles.
void gwindows::paintwindow(int left, int top, int right, int bottom)
{
    int i;

    if (wstack[wptr]->bt == THREED) {
        // Bottom and left edge are dark gray. Top and left sides
        // are drawn white.
        for (i=0; i<3; i++) {
            setcolor(EGA_DARKGRAY);
            line(right-i,top+i,right-i,bottom-i);
            line(left+i,bottom-i,right-i,bottom-i);
            // Left and top are white
            setcolor(EGA_WHITE);
            line(left+i,top+i,right-i,top+i);
            line(left+i,top+i,left+i,bottom-i);
        }
        setcolor(wstack[wptr]->bfc);
        setlinestyle(SOLID_LINE,0,NORM_WIDTH);
        // Fill in the window
        setfillstyle(wstack[wptr]->bft,wstack[wptr]->bfc);
        bar3d(left+3,top+3,right-3,bottom-3,0.0);
    }
    else {
        // Set the graphics parameters for the pop-up window to be displayed.
        // Then make the window. Use bar3d() so that there will be a border.

        setcolor(wstack[wptr]->bc);
        setlinestyle(wstack[wptr]->bt,0,NORM_WIDTH);
        setfillstyle(wstack[wptr]->bft,wstack[wptr]->bfc);
        bar3d(left,top,right,bottom,0.0);
    }
}

// Remove the current graphics window from the screen
int gwindows::gunpop(void)
{
    struct viewporttype currview;
    struct GRAPHICSWINDOW *w;

    if (wptr <= 0) return(0);                     // If at bottom of stack, quit
    wptr--;
    w = wstack[wptr];                             // Get the top-most window
    // Restore the screen image
    getviewsettings(&currview);                   // Get the window size
    putimage(0,0,w->saveareal,COPY_PUT);
    putimage(0,(currview.bottom-currview.top)/2+1,w->savearea2,COPY_PUT);
    // Restore the view settings to those values before the window
    // was created
    setviewport(w->vleft,w->vtop,w->vright,w->vbottom,1);
    moveto(w->cpx,w->cpy);
    setcolor(w->drawcolor);
    free(w->saveareal);                             // Free stack structure and screen
    free(w->savearea2);
    delete(wstack[wptr]);
    return(1);                                     // Return a success flag
}

```

```
// Remove all popped up windows that currently exist. This is a handy
// routine to clean up the screen in certain situations.
```

```
void gwindows::unpopallwindows(void)
{
    while (gunpop());
}
```

10.2 使用窗口程序包

现在已讨论了所有基本的知识，让我们探索如何使用窗口程序包。这是一个非常简单的过程。首先，应当在你的应用程序中包含gpopup.h标题文件，以便所有的函数原型可用于编译程序。还必须在你的应用程序中说明类型gwindows的窗口对象。回忆一下，它的构造器将为你初始化窗口堆栈，于是窗口对象将准备好可供使用。每当你想要上弹窗口时，如早先在本章中讨论那样调用gpopup()。请记住，上弹窗口坐标是相对于整个屏幕而不是当前观察端口指定的。这使你能对上弹窗口的布局进行广泛控制。为除去窗口，应调用gunpop()，你必须为你建立的每一个窗口调用一次gunpop()。

还要记住在堆栈上保持上弹窗口。因此，当你调用gunpop()时，最新的窗口将总会首先除去。

一旦建立上弹窗口，观察端口将会改变以反映出新窗口的情况。如图10.4所解释的，新上弹窗口的左上角将与单元(0,0)相对应，当前的位置将置为这一点，并开始裁剪。

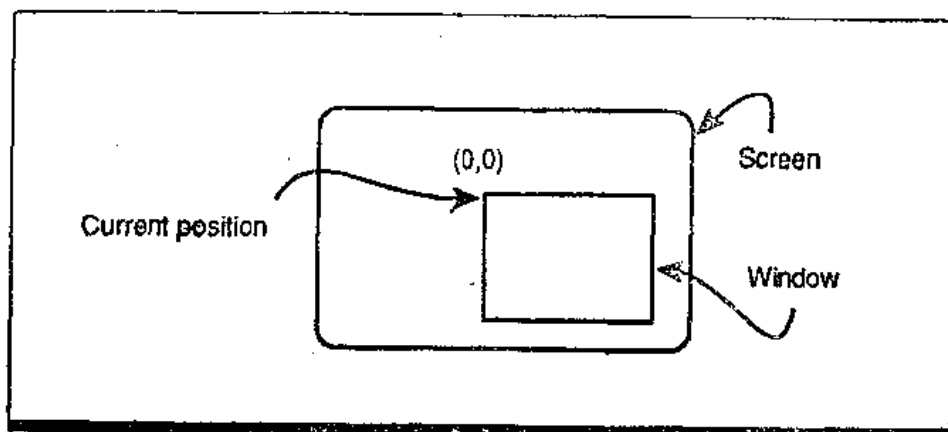


图 10.4 上弹一个窗口后的当前位置

10.3 测试程序

在结束本章之前，将提供一个测试程序，它检测我们的图形窗口程序包的能力。它展示如何使上弹窗口程序包工作。测试程序poptest.cpp（列表10.3）建立了如图10.5所示的三个上弹窗口，然后把它们从屏幕除去。在显示或除去每一窗口之后，你必须按键以便程序继续进行。

正如所编写的那样，程序使用由你的视频适配器提供的缺省模式。如果你想要试试特定的图形模式，则改变赋予main()中gdriver和gmode的值。最后，请注意程序如何存取每一窗口中的坐标。例如，在第一个上弹窗口中的中心画一个圆，为了确定窗口的中心

点，必须调用getviewsettings()，然后从视图设置计算窗口的中心。这简明地展示出窗口中的坐标是相对于当前上弹窗口存取的。

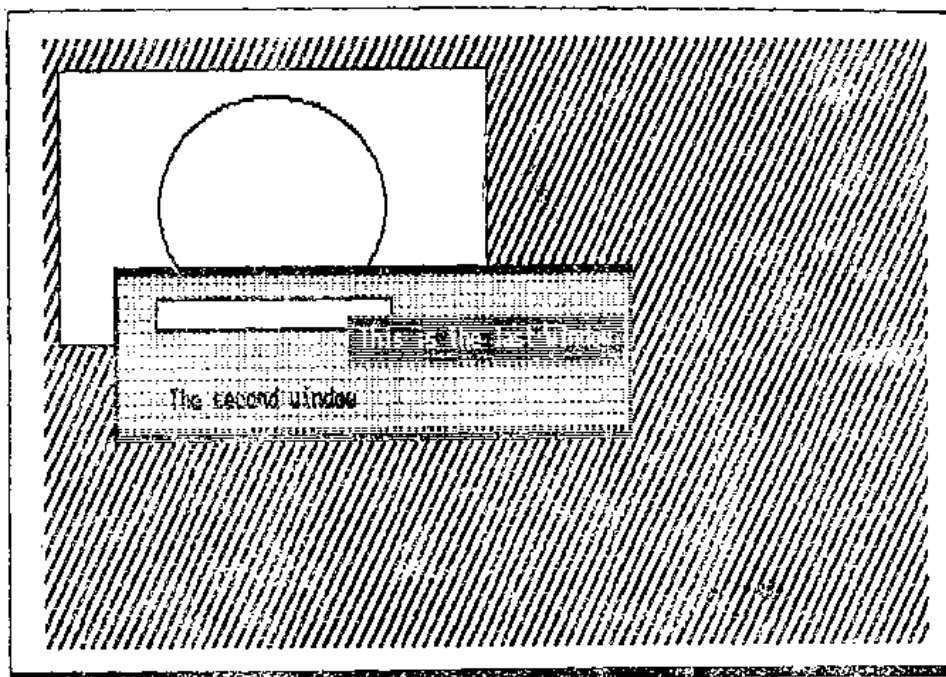


图 10.5 poptest.CPP的输出

```
// The view settings are now equal to the pop-up window boundaries
getviewsettings(&view);
// Draw a circle in the window
circle((view.right-view.left)/2,(view.bottom-view.top)/2,
       (view.left+view.right)/4);
```

为编译和连接poptest.cpp，你可以使用包含以下文件的规划文件：

主文件	标题文件	章
gropup.cpp	gpopup.h	10
poptest.cpp	None	10

列表10.3poptest.cpp

```
// poptest.cpp -- Tests the graphics pop-up window routines. This
// program displays three overlapping pop-up windows on a patterned
// background. The windows are drawn one at a time and then removed one
// at a time. Press any key to step through the various phases of this
// demonstration. This routine demonstrates how the screen is saved and the
// use of the screen coordinates in a pop-up window.
#include <graphics.h>
#include <conio.h>
#include "gpopup.h"                                // Header file for the graphics
                                                    // pop-up window package
gwindows w;                                         // Declare a window object

main()
```

```

int gmode, gdriver = DETECT;          // Use a mode with several colors
struct viewporttype view;
char message[] = " This is the last window ";
int halfwidth, height;

initgraph(&gdriver, &gmode, "\\tc\\bgi");
setfillstyle(SLASH_FILL, BLUE);       // Draw a background that
setlinestyle(DASHED_LINE, 0, WHITE);  // covers the whole screen
bar(0, 0, getmaxx(), getmaxy());

// Pop-up a window in the top-left portion of the screen. Use a solid
// line border in the maximum color and fill the pop-up window
// with color 0.
w.gpopup(10, 10, getmaxx()/2, 100, SOLID_LINE, getmaxcolor(), SOLID_FILL, 0);
// The view settings are now equal to the pop-up window boundaries
getviewsettings(&view);
// Draw a circle in the window
circle((view.right-view.left)/2, (view.bottom-view.top)/2,
       (view.left+view.right)/4);
getch();                             // Wait until a key is pressed

// Draw a second pop-up window with a three-dimensional border
// and the window's background set to color EGA_LIGHTGRAY
w.gpopup(50, 75, getmaxx()*2/3, getmaxy()*2/3, THREED, 1,
        HATCH_FILL, EGA_LIGHTGRAY);
setfillstyle(WIDE_DOT_FILL, BLACK);
bar3d(30, 10, 200, 20, 0, 0);        // Draw a filled box in the window;
setcolor(getmaxcolor());
outtextxy(40, 40, "The second window");
getch();                             // Wait for a key to be pressed

// The last pop-up window displays a message at the center of the
// screen. This type of pop-up window is good for displaying error
// messages in graphics mode.
halfwidth = textwidth(message)/2;    // Base the size of the window
height = textheight(message);        // on the string it'll display
w.gpopup(getmaxx()/2-halfwidth, getmaxy()/2-height, getmaxx()/2+halfwidth,
        getmaxy()/2+height, SOLID_LINE, 1, SOLID_FILL, 2);
setcolor(0);
settextjustify(CENTER_TEXT, CENTER_TEXT); // Center message and
getviewsettings(&view);                // display it
outtextxy((view.right-view.left)/2, (view.bottom-view.top)/2, message);
getch();                             // Wait for a key to be pressed

w.gunpop();                          // Remove the last window
getch();                             // Wait until a key is pressed
w.gunpop();                          // Remove the middle window
getch();                             // Wait until a key is pressed
w.gunpop();                          // Remove the first window
closegraph();                        // Exit graphics mode
return(0);
}

```


第十一章 交互式绘图工具

通过把绘图函数放到程序中来用BGI绘图是非常简单的。但是在画画或CAD应用中，我们需要使用鼠标或键盘“交互式地”绘图。正如你所看到的，BGI不包含交互绘图例程，这样我们必须建立自己的例程。因此，在本章我们要花些时间构造一组类程，它们提供橡皮筋的线段、喷涂的效果、多边形绘图、正文表项等等。

这些类程将成为用于第十二章和第十三章的画画和CAD程序中的主要绘图工具。但是，你将发现，这些交互式绘图工具用在按你自己需要而建立的其它交互式图形应用中过于一般。

通过建立使用OOP资源的工具，我们达到这种灵活性。特别是，每一绘图工具作为一个独立的类程实现。当展开各种类程的细节时，你会看到OOP是共享代码的强有力手段并为扩充工具箱敞开大门——而这正是我们在第十三章CAD程序中所做的。

11.1 交互式图形程序包

我们开发的交互式绘图程序包含13个绘图工具，其中每一个工具作为独立的类程实现（见表11.1）。所有工具共享公共的祖先——drawingtool类程。图11.1示出类程的层次。

表 11.1 包含在draw.cpp中的交互绘图类程

类程	描 述
drawingtool	所有交互式绘图类程的基础类程
penciltool	支持徒手绘图
erasertool	擦除部分屏幕
spraycantool	提供喷涂效果
linetool	交互式画线类程
rectangletool	交互地画矩形
fillrectangletool	交互地画填充的矩形
polygontool	交互式画多边形类程
fillpolygontool	允许用户画填充的多项式
circletool	交互地画圆
ellipsetool	交互地画椭圆
arctool	交互地画弧
texttool	支持正文输入
clearwindow	清除绘图窗口

首先，类程的数目看起来似乎有点使人不知所措。但是，当你研究代码时，便会发现大多数这些类程都是比较小的并且是按简单、可管理的步骤在其它类程之上建立的。

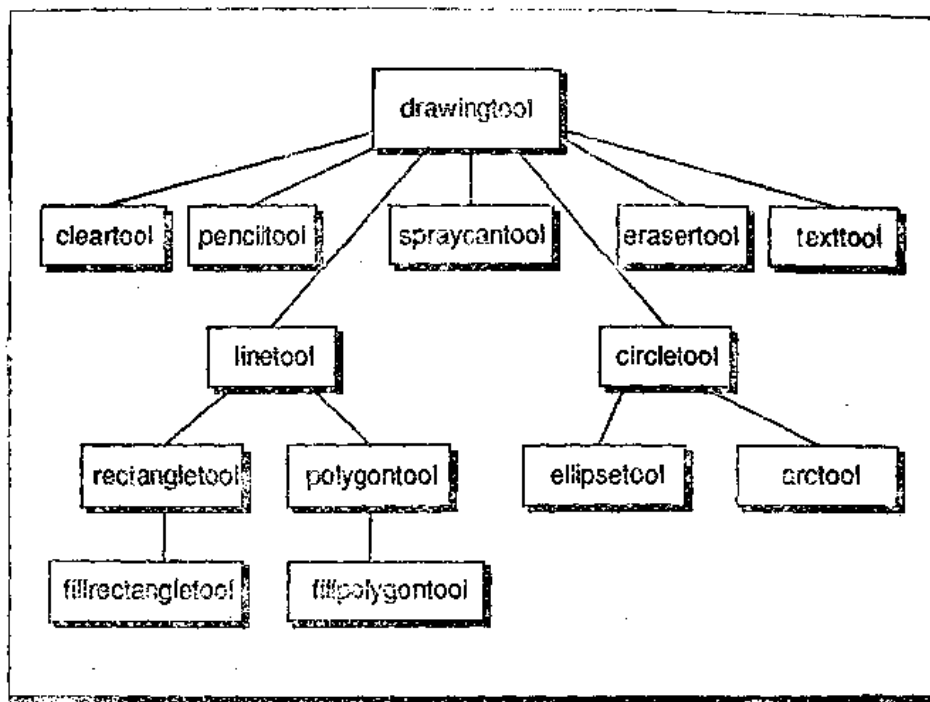


图 11.1 在draw.cpp中提供的交互式工具层次

drawingtool类概括了用户交互作用的基本流程，并为绘图例程提供公共程序设计接口。由于所有的工具由同一“基础”类建立，故能够在应用程序中充分利用称为“多态性”的OOP技术。我们将在第十二章和第十三章的画画和CAD程序中使用此技术。

绘图类程的定义包含在draw.h（列表11.1）中，而成员函数的代码可以在draw.cpp中（列表11.2）找到。

11.1.1 绘图约定

每一交互式绘图类程都遵从若干约定。第一，类程假定将使用在第八章开发的鼠标类程。由于我们的鼠标类程支持鼠标和键盘，故绘图例程可使用任一输入设备。但是，发现鼠标提供最好的性能。

第二，有由绘图工具使用的一系列全程变量。必须你的应用程序中定义这些变量并赋予适当的值。在表11.2中列出了一些变量的表。你必须在调用draw.cpp中的任一交互式绘图函数以前，在应用程序中把值赋给列在表中的所有全程绘图变量。

特别是，绘图函数把它们的输出限制在一个预定义的窗口。因此，当调用绘图函数之一时，要把当前的观察端口置为绘图窗口并允许裁剪。当绘图动作结束时，恢复观察端口为全屏幕。因此，必须通过置4个全程变量wl、wt、wr和wb为所要求的绘图窗口的左、顶、右和底边，在应用程序中初始化绘图窗口的边界。

此外，为初始化绘图函数所用的绘图参数，必须设置5个全程变量。这5个全程变量在表11.2中列出。它们根据窗口的尺寸指定各种绘图参数，它们应当置为在graphics.h中定义的适当的宏常数。例如，globalfillstyle可取值SOLID_FILL、EMPTY_FILL或任何其它在graphics.h中定义的填充图案值。

最后，绘图程序包draw.cpp假定你已在你的应用程序中定义了一个称为Mouse的kb-

表 11.2 在使用draw.cpp的应用程序中定义的全程变量

变量	描 述
Wl	绘图窗口的左列
Wt	绘图窗口的顶
Wr	绘图窗口的右边
Wb	绘图窗口的底
globaldrawcolor	定义绘图函数使用的绘图颜色
globalfillcolor	定义使用的填充颜色
globalfillstyle	定义使用的填充类型
globallinestyle	定义使用的线型
globaltextstyle	定义使用的正文类型
maxx	屏幕的最右坐标
maxy	屏幕的最低坐标
mouse	draw.cpp中使用的鼠标对象

dmouseobj对象。请记住，kbdmouseobj类是在第十一章开发的键盘和鼠标程序包中定义的，如果提供鼠标就使用鼠标，如果不提供鼠标就使用键盘。

11.1.2 仔细考查draw.cpp工具

现在我们准备大略看看在draw.cpp中提供的交互式绘图类程的层次。这些类程中的一些与其它类似，因此我们不想给它们同样的关注；但是，我们提供如何使用每一个类程的简短描述。

下面示出的drawingtool类程提供基本外壳，它规定我们与绘图工具一起使用的程序设接口。它的定义是：

```

class drawingtool {
public:
    int x, y, oldx, oldy;           // Current and last mouse position
    int left, top, right, bottom;    // Bounds of the figure
    int drawcolor;                  // Color to be used in figure
    virtual void draw(void);         // High-level drawing function
    virtual void performdraw(void);  // Defines user interaction
    virtual void startdrawing(void); // Sets up for drawing
    virtual void updatedrawing(void); // Interactively draws figure
    virtual void finishdrawing(void); // Finalizes drawing
    virtual void hide(void) { }      // Removes figure from screen
    virtual void show(void) { }      // Displays or updates figure
    // The following member functions will be defined in the CAD program
    virtual void display(void) { }
    virtual void getbounds(int &l, int &t, int &r, int &b) { }
    virtual void translate(double transx, double transy) { }
    virtual void rotate(double angle) { }
    virtual drawingtool *dup(void) { return this; }
};

```

drawingtool类程以四个例示变量X、Y、oldx和oldy开始，它们记录鼠标的当前和以前的位置。当出现像按按钮的事例时，将修改这些变量。

现在让我们考查drawingtool中的每一成员函数，以便更好地了解如何使用它们。最顶部的成员函数draw()是最高层的例程，当我们想要交互绘制图表时，则调用该例程。我们假定，这是选择特定的绘图函数时调用的成员函数。但是，如果考查代码，你将会注意到draw()做非常少的工作。它仅置观察端口坐标和允许裁剪绘图窗口的大小，调用performdraw()，然后把观察端口坐标恢复为全屏幕。现在，你可能怀疑performdraw()是否能真正绘图。不十分会。但是它给予我们有关如何在drawingtool类程中的用户交互作用的最好的线索，让我们考查一下。

等待用户按左鼠标按钮的无限while循环建立performdraw()函数：

```
while (!mouse.buttonpressed(LEFT_BUTTON)) :
```

一旦按左按钮，就通过调用mouse.getcoords()检索鼠标坐标。用例示变量x和y中保存的这些坐标对绘画观察端口进行检查，以便看看用户是否超出绘图窗口按键。如果是，则performdraw()经过return语句退出：

```
mouse.getcoords(x,y);
if (x < wl || y < wt || y > wb || y > wr) return;
startdrawing();
updatedrawing();
finishdrawing();
```

否则，通过调用三个函数startdrawing()、updatedrawing()和finishdrawing()，函数往下继续。在进行绘图时，这三个成员函数实际上是一个整体。

为什么绘图进程在这三个函数之间细划分？因为我们将从每一其它工具导出绘图工具。我们需要基础类程尽可能灵活，使得我们能仅重设那些需要修改代码的部分。用这种方法，能够共享大量的代码而仅需在我们的派生类程中修改小块的代码。

让我们考查这三个新函数中的每一个。首先，startdrawing()负责为绘图进程建立任一变量。例如，当画一条线时，startdrawing()将把当前位置移到按鼠标的位置。

以类似的方式，finishdrawing执行某些在绘制图表之后必须做的清除动作。例如，能在画出它的外部草图之后填充多边形的内部。但是，基础类程中此函数为空白。派生类程当需要时可重设它。

updatedrawing()函数是工作间，它是实际进行交互作用的地方。它的代码是：

```
void drawingtool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) { // Button must be pressed
        mouse.getcoords(x,y); // Get current location of mouse
        if (x != oldx || y != oldy) { // Only update figure if the mouse
            hide(); // has moved. Remove part or all
            show(); // of the figure. Draw the figure.
            oldx = x; oldy = y; // Update saved position of mouse
        }
    }
}
```

此函数还建立起一个while循环，但这次是只要按左鼠标按钮，循环就继续下去（回想一下，一旦初始按下鼠标按钮，则用performdraw()开始绘图过程）换句话说，只要按压鼠标按钮，则绘图就继续下去。但是，为避免不必要的屏幕书写，仅当改变鼠标的位置

置, `updatedrawing ()` 才继续进行绘画。这使得当实际上不需要修改什么时, 屏幕不致于闪烁。这由 `if` 语句规定, 它比较鼠标的最新位置和它的最后记录位置:

```
if (x != oldx || y != oldy) {    // Only update figure if the mouse
    hide( );                     // has moved. Remove part or all
    show( );                     // of the figure. Draw the figure
    oldx = x; oldy = y;          // Update saved position of mouse
}
```

但是在什么地方绘图呢? 对此我们需要返回到 `updatedrawing ()` 调用的 `show ()` 和 `hide ()` 成员函数。这些函数用于绘图和从绘图窗口擦去图表。

`show ()` 成员函数可能不言自明, 但 `hide ()` 是什么呢? 请记住, 在交互式绘图时, 用户可以改变图表的大小。因此, `hide ()` 函数用于擦去旧图表, 使得 `show ()` 可以出现, 并用它的新尺寸重画图表。这是在先前说明的 `if` 语句的地方进行的。如果不改变鼠标的位置, 则图表的大小也不改变, 因此我们迂回 `hide ()` 和 `show ()`。成员变量 `oldx` 和 `oldy` 记录鼠标的最后位置, 以及图表的最后的位臵和大小。`oldx` 和 `oldy` 的初值在 `startdrawing ()` 设置, 然后每次通过 `updatedrawing ()` 中的 `while` 循环进行修改。

请回忆一下, `oldx` 和 `oldy` 绘出鼠标的最后位置并在 `startdrawing ()` 初始化和在 `updatedrawing ()` 更新。

11.1.3 用笔绘图

现在我们已经展开 `drawingtool` 类程的组成成分, 让我们看看如何用类程派生出专用的绘图工具。我们从 `penciltool` 类程开始, 该类程用于绘如图 11.2 中示出的徒手画的图表。

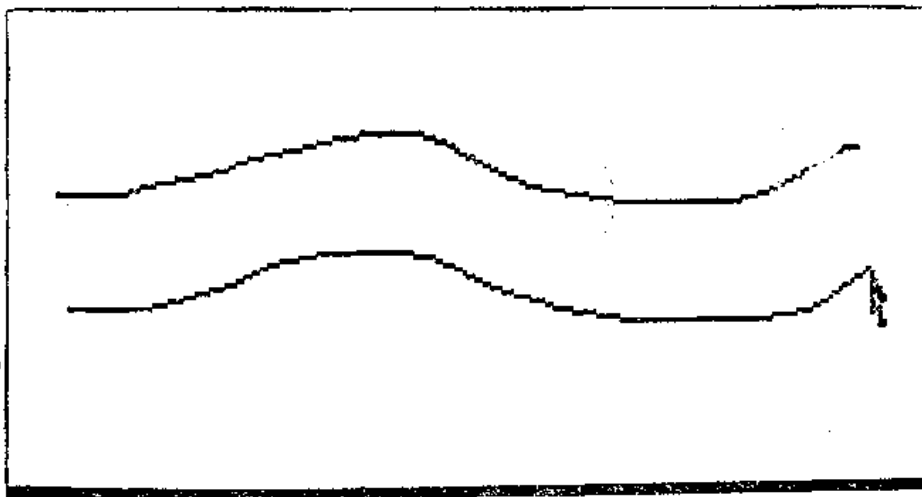


图 11.2 用 `penciltool` 徒手画曲线

使用 `penciltool`, 只要按左鼠标按钮, 你可以画连续的曲线。每当释放鼠标按钮时, 绘图停止。为恢复绘图, 只要再按下左按钮即可。为退出函数, 你在绘图窗口以外撤按左鼠标按钮。你大概将要选择另一绘图函数。我们看看在第十二章开发画画程序时怎样使用此技术。

派生一个工具, 像从 `drawingtool` 派生出 `penciltool`, 是指在需要替换继承的类程函数

或补充它们时重设成员函数。我们从最高层开始,往下进行并作必要的添加。因此, pencil-tool类程定义指出,为实现笔工具我们已重设哪些函数。它的定义是:

```
class penciltool : public drawingtool { // A curve drawing class
public:
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void show(void);
};
```

首先, draw () 函数必须被重设,以便我们可以设置做笔绘图操作时使用的绘图颜色。此值必须重置于表11.2中列出的适当的全局变量中,因为它们可能被应用程序中别的函数改变。同样原因,我们重设其它类程中的draw ()。一旦重置绘图颜色,则调用从drawingtool继承的draw () 函数以便继续绘图,

```
void penciltool::draw(void)
{
    setcolor(globaldrawcolor); // Switch to the global drawing color
    drawingtool::draw();       // Draw with the pencil
}
```

往下,重设startdrawing以便初始化绘图动作(当按左鼠标按钮时,调用此函数)。通过从鼠标当前位置到它以前的位置画一条线建立徒手画的曲线。因此,当鼠标移动时,线增长。为启动此进程, startdrawing () 调用moveto () 以便把当前位置置为由类程变量x和y给出的鼠标的位置。

```
moveto(x-wl, y-wt)
```

请注意,传送到moveto () 和lineto () 的坐标由绘图观察端口的左边界和顶边界调准。这必须做,因为moveto () 使用相对于当前观察端口的坐标,而绘图窗口和鼠标坐标x和y则总是以全屏幕坐标给出的。

最后,重设show () 函数以便完成调用lineto () 的画线工作:

```
void penciltool::show(void)
{
    // Hide mouse before drawing
    mouse.hide();           // Draw a line from last position
    lineto(x-wl, y-wt);     // Restore the mouse cursor
    mouse.show();
}
```

如此代码所说明的,为使鼠标光标不致干扰屏幕,应调用环绕lineto () 的mouse.hide () 和mouse.show ()。

11.2 擦 除

现在我们已知如何绘制图表,让我们看看如何才可擦除它们。为此,介绍类程eraser-tool,该类程设计成在鼠标光标之下,把矩形屏幕区域置为背景颜色。通过按住左鼠标按钮并到处移动鼠标,你可以擦去你想擦除的任一绘图窗口的区域。

除了它是用背景颜色把填充的小条形画到屏幕而不是使用一系列线以外,eraser-tool类程与penciltool相似。为生成这些小的清除块,类型置为SOLID_FILL,填充与绘图颜

色置为背景颜色。这通过重设draw()而完成,就像我们对penciltool所做的那样。此外,在调用继承的draw()函数以前嵌入以下语句:

```
setcolor(getbkcolor());           // Use the background color
setfillstyle(SOLID_FILL,getbkcolor()); // to erase the screen
```

每当定位鼠标光标并同时按下鼠标按钮时,该函数就给出这些小的实心条形体。擦除条形体的像素尺寸由ERASERSIZE规定,ERASERSIZE恰好是在draw.h中的erasertool类程之前定义的常数,大小为5:

```
const int ERASERSIZE = 5;           // Half the size of the eraser
```

你可以试图使擦除器的大小为一个变量,使得用户可以交互地改变擦除器的大小。于是,通过使用BGI的bar()函数,在show()中出现擦除动作:

```
bar(x-wl,y-wt,x-wl+ERASERSIZE,y-wt+ERASERSIZE); // Erase region
```

11.3 喷涂效果

函数spraycantooll通过随机设置在矩形区域里的像素为当前绘图颜色提供简单的喷涂效果。

当接左鼠标按钮时,如图11.3所示,将出现喷涂动作。你较长时间在一个位置上按住鼠标,将会填充更多的屏幕单元。为暂时停住喷涂,应释放左鼠标按钮。为退出喷涂函数,应把鼠标移出绘图观察端口并撤按左按钮。

由于撤按鼠标时喷涂效果不断地产生(甚至当鼠标不移动时),我们必须重设update-drawing()。回想一下,如果已移动鼠标,则drawingtool的updatedrawing()的实现仅调用hide()和show()。

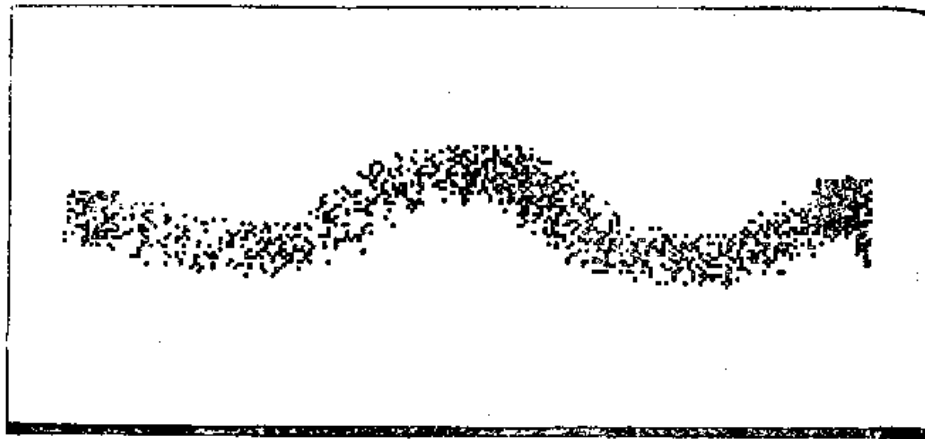


图 11.3 使用喷涂壶工具

由重设的函数show()中的两个for循环产生喷涂效果:

```
for (i=0; i<8; i++)
    putpixel(x-random(SPRAYSIZE)+5-wl,
             y-random(SPRAYSIZE)+5-wt,globaldrawcolor);
for (i=0; i<8; i++)
```

```
putpixel(x-random(SPRAYSIZE-2)+3-w1,
        y-random(SPRAYSIZE-2)+3-wt,globaldrawcolor);
```

每一for循环负责在鼠标光标周围的矩形区域内绘8个像素。第一个for循环绘远离鼠标光标的像素，而第二个for循环绘较近的像素。请注意，由每一for循环迭代画出的特定像素是使用TurboC++函数random() 随机地选择的。这均匀地分布像素。喷涂区域的大小部分由在spraycantoal中说明的常数SPRAYSIZE确定。但是，在两个for循环中的其它整数值同样也影响喷涂的分布和位置，你也许会想用其它的喷涂模式进行实验。

11.4 画线

linetool类用于画线段。通过指向开始画线的地方，按住左鼠标按钮，然后拖曳鼠标到想要的结束点，可让你画出一条线。当释放鼠标时，线不动。但是，当按按钮时，linetool继续从按按钮的鼠标初始位置画线，直到鼠标当前位置。因此，当鼠标在屏幕周围移动时，线将会按需要收缩和伸张。画线的过程在图11.4中示出。这种类型的线称为橡皮筋线，因为线段可以像橡皮筋那样灵活地出现，为画更多的线，你可以重复这一简单过程。

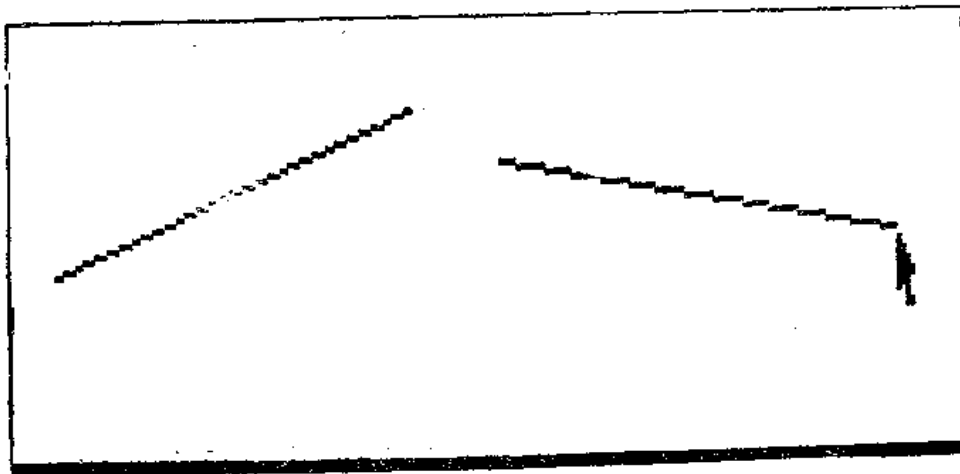


图 11.4 使用linetool对象

正如你可能希望的，linetool类似于以前讨论的函数。linetool做的第一件事是重设draw()，以便设置绘图颜色和线型：

```
setlinestyle(globallinestyle,0,NORM_WIDTH);
setcolor(globaldrawcolor);
```

请注意，线的宽度限制为NORM_WIDTH。这样做是为简化代码。你可能想要修改draw.cpp，使得线宽度是像globallinestyle和globaldrawcolor那样的变量。

linetool的一个重要方面是橡皮筋线效果。使用由BGI setwritemode() 函数（见第七章）提供的“异或”特征的技术，允许我们在屏幕周围绘画和移动线，而不必永久地影响它所写的东西。一旦设置了“异或”模式，则通过用另一条线在它们上面书写，就可以从屏幕擦除这些线。通过下一行代码，按左按钮之后将在startdrawing()中打开“异或”特征。

```
setwritemode(XOR_PUT);
```


在此之后，它们通过“异或”把所有的线画到屏幕上。有时你会看到，如果线改变颜色它和屏幕的其它图表交叉时“异或”的效果。

在图 11-1 中，通过世界坐标 (X, Y) 和屏幕坐标 (Xs, Ys) 的转换，每个点被下

画出多边形的封闭边，如果使用fillpolygontool，还可填充多边形的内部。

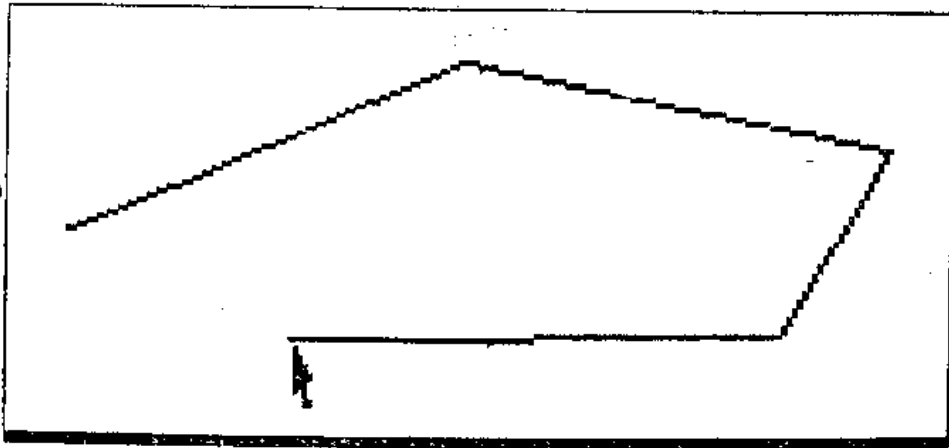


图 11.5 用polygontool画多边形

既然用户交互作用不同，我们重设updatedrawing()，以便可以用等待闭合信号（按右按钮）的do循环替换原先使用的while循环。在polygontool中，do循环用于测试是否按了右按钮。此外，在do循环内部，添加if语句测试左按钮。如果发现按左按钮，则保存鼠标坐标，使当前橡皮筋线冻结不动，并在前一线的端点处开始画新的橡皮筋线。但是，如果按右按钮，则通过把它往回连接到finishdrawing()中的第一个坐标闭合多边形。

还有，通过带XOR_PUT调用setwritemode()，“异或”技术可用于产生橡皮筋效果。为显示正确的颜色，在用户闭合多边形之后，必须用置为COPY_PUT的书写模式重画完整的多边形。为此，多边形类跟踪用于绘制多边形的屏幕坐标，这些坐标保存在polygontool类包含的poly数组中。为简化起见，数组被说明为MAXPOLYSIZE的大小，polygontool中的常数置为40。

11.6 画 矩 形

现在我们开发两个伙伴类，以便交互地画矩形和填充矩形——rectangletool和fillrectangletool。这些工具是围绕BGI的rectangle()和bar3d()命令建立的。

我们再一次使用BGI的“异或”特征建立橡皮筋效果。但是，如何使用工具的填充矩形版本完成此项工作？由于bar3d()不受setwritemode()任选项的影响，所以我们使用虚线的空矩形来定矩形的大小，然后当通过调用bar3d()设置大小时填充它。（请记住，如果深度置为0，则从bar3d()可以画填充的矩形。）这可以看出fillrectangletool应当从rectangletool派生出，这恰好是我们要做的。此外，我们从linctool派生rectangletool，以便可以利用它的用户交互作用和当地建立线型的代码，由于这点，我们需要在rectangletool类中重设的函数只是show()和hide()。它们调用rectangle()绘制矩形，但是hide()使用由(oldx, oldy)规定的鼠标先前的位置。而show()则使用新的鼠标位置(x, y)。请记住，这是我们正使用“异或”特征在画线和擦除线之间进行转换。

为画填充的矩形，仅需添加少量代码。首先，重设draw()和startdrawing()，目的在于置填充设置和置线型为虚线。（请记住，我们使用虚线的矩形直到矩形是想要的大

小。)其次,我们重设`finishdrawing()`,使得一旦设置矩形大小(当释放左鼠标按钮时),就用填充矩形画在矩形的轮廓上。这通过调用`bar3d()`来完成:

```
bar3d(x1-w1,y1-wt,x-w1,y-wt,0,0); // Draw the filled rectangle
```

如前所述那样,直到按左按钮时才开始绘图。此外,退出绘图函数的唯一方法是在绘图窗口以外撒按鼠标。

11.7 画 圆

在行中派生的下一类是 `circletool` 类,它允许我们交互地画如图 11.6 中所示的圆。通过指定圆中的中心点开始用 `circletool` 画圆。这通过把鼠标移到要求的位置并按左鼠标按钮完成。然后,当按住左鼠标按钮时,你可以拖曳鼠标向左和向右,以便改变圆的大小。

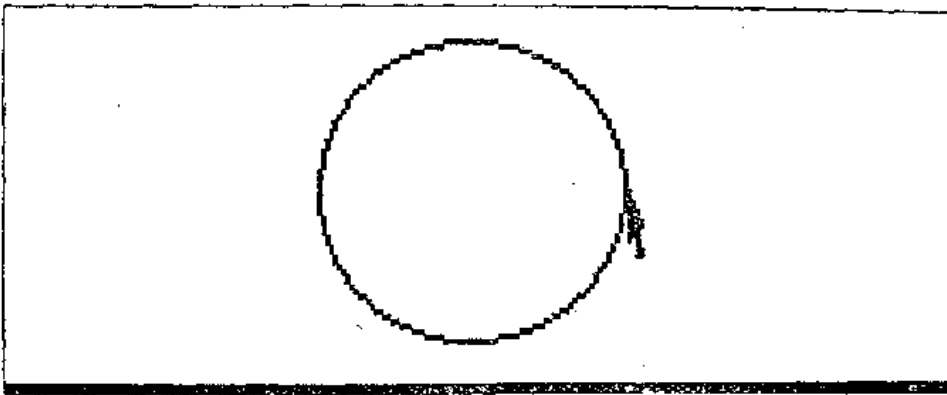


图 11.6 用 `circletool` 画圆

你可能不晓得我们如何建立橡皮筋的圆?毕竟, `circle()` 函数不受 `setwritemode()` 的影响。我们可以采取两种方法。例如,可以书写支持“异或”的我们自己的 `circle()` 函数。但是我们不想采用此方法。

取而代之,我们使用 `getimage()` 和 `putimage()` 来帮助我们产生为画圆所需要的橡皮筋效果。这通过保存画圆处的屏幕映像完成,以便后来可以通过使用 `putimage()` 把保存的映像往回拷贝到屏幕来除去屏幕的圆。实际上,我们只不过是上弹内部带有图表的窗口。让我们从仔细观察 `circletool` 开始。

你注意一下有关 `circletool` 的第一件事,它比先前的任一类程更长且大概更有威力。但是,分段来看,该类程实际上与以前讨论的那些相似。

为什么在 `circletool` 中的代码比我们以前看到更复杂?这有两个主要原因。第一,我们想要通过使用 `getimage()` 和 `putimage()` 模拟橡皮筋效果。但是,由此必须自己处理映像的裁剪。第二,使用 `getimage()` 和 `putimage()` 是较高分辨率模式中需要解决的问题,因为很多这些模式要求 64K 以上以保存整个绘图窗口,而不能使用对 `getimage()` 的单个调用。代替的是,我们把绘图窗口分成两个区域各自独立地保存每一个。屏幕区域保存到由 `covered1` 和 `covered2` 指向的两块存储区, `covered1` 和 `covered2` 是 `circletool` 中说明的新实例变量。

重设 `draw()` 函数,使得我们可以分配和重分配为保存屏幕而要求的存储。一旦分

配存储, 则调用继承的draw() 函数以执行绘图操作。完成绘图以后, 释放由 covered 1 和 covered 2 规定的存储区。

如果没有足够存储保存屏幕映像, 则调用函数 mallocerror()。当前, mallocerror() 异常终止程序。你可能想要修改这个错误处理程序, 以使程序更加与用户友好。

以 startdrawing() 成员函数开始绘图过程。请注意, 它调用 getimage() 两次, 以保存定位在光标上的屏幕。虽然它仅有效地保存一点, 但这将开始我们的橡皮筋进程。

由 updatedrawing() 成员函数来控制画圆。当移动鼠标时, 通过取圆的中心 X 坐标和鼠标的当前位置之间的差的绝对值, 不断计算圆的半径:

```
absradiusx = abs(centerx - x);
```

absradiusx 的结果后来作为在成员函数 show() 中调用 BGI 函数 circle() 时的半径变元。

updatedrawing() 的其余复杂性体现在裁剪由 getimage() 和 putimage() 使用的屏幕区域的界。请记住, 这两个函数不裁剪相对于当前观察端口的它们的映像。我们必须自己执行裁剪, 裁剪过程在图 11.7 中解释。

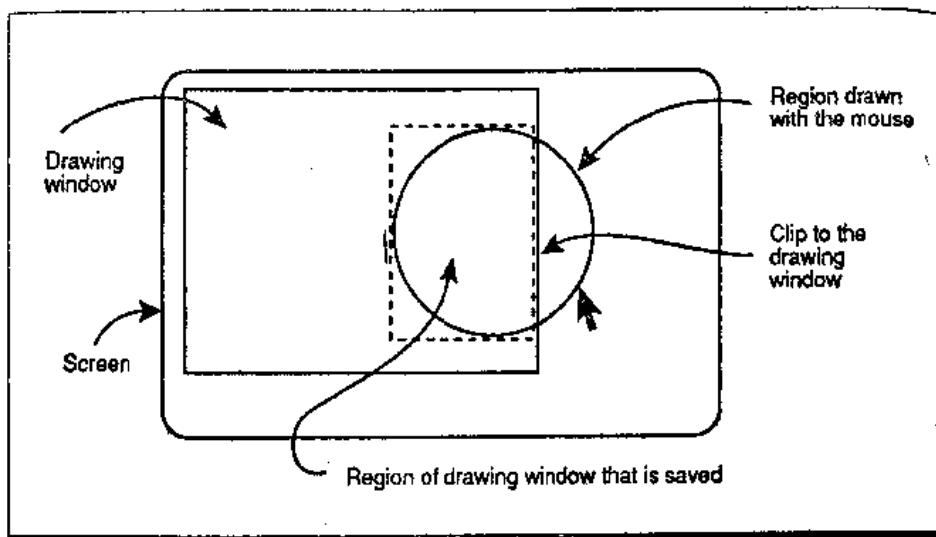


图 11.7 对橡皮筋的圆进行裁剪

用在 updatedrawing() 中的四个 if-else 语句完成裁剪, 它们查看该圆是否扩展到绘图窗口的边缘之外。

由于屏幕的长宽比在 x 和 y 方向上可能是不同的, 此函数可能卷紧过多保存的绘图窗口, 特别是在 y 方向上, 但是, 问题不大。

注意有关 circletool 的另一个问题是, 如果半径是 0, 则它不画圆。这是必要的, 因为 BGI 函数 circle() 将用 1 像素的半径画半径为 0 的圆。由于这不是我们想要的, 因此, 我们通过检查 absradiusx 变量是否有 0 值来避免在 show() 中画这样的圆。我们应当对 ellipse() 和 arc() 函数也进行同样类型的检查。

11.8 画椭圆

画椭圆是由非常类似于 circletool 的 ellipsetool 类程完成的。二者之间的主要差别是,

ellipsetool允许你改变椭圆的x和y半径。当使用circletool时，是通过左右拖曳鼠标而改变x半径的，现在y半径也可以通过上下拖曳鼠标而改变。

由于ellipsetool如此类似于circletool类程，我们从它派生出ellipsetool。我们继承来自circletool的代码并重分配为保存屏幕所要求的存储。但是，仍需要重设updatedrawing（），使得我们可以跟踪椭圆的y坐标的变元。同样地，重设show（）使得可以画椭圆而不是圆。

11.9 画 弧

arctool类程允许我们交互地画一系列的弧。遗憾的是BGI arc（）函数不适用于交互式画弧函数。主要的困难是它不便于交互地规定arc（）函数需要的所有参数，如中心点、半径和扫描角。我们回避此问题并建立为画弧用的非常简单的函数。

我们的类程arctool总是在三点钟的位置上开始画弧，但是它允许你改变弧的扫描角。类似地，当沿角向上移动鼠标，与此同时按左按钮将减少弧的扫描角。图11.9解释画弧的过程。

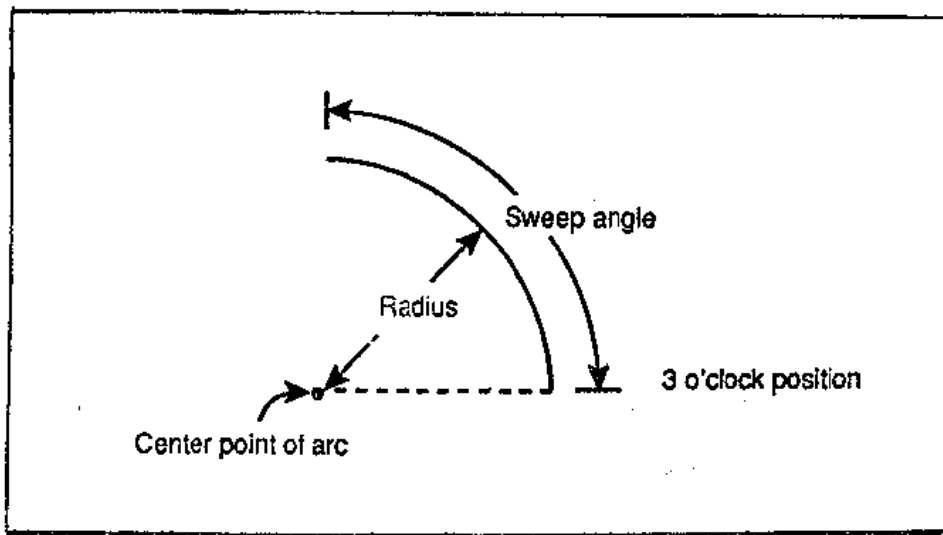


图 11.8 弧的成分

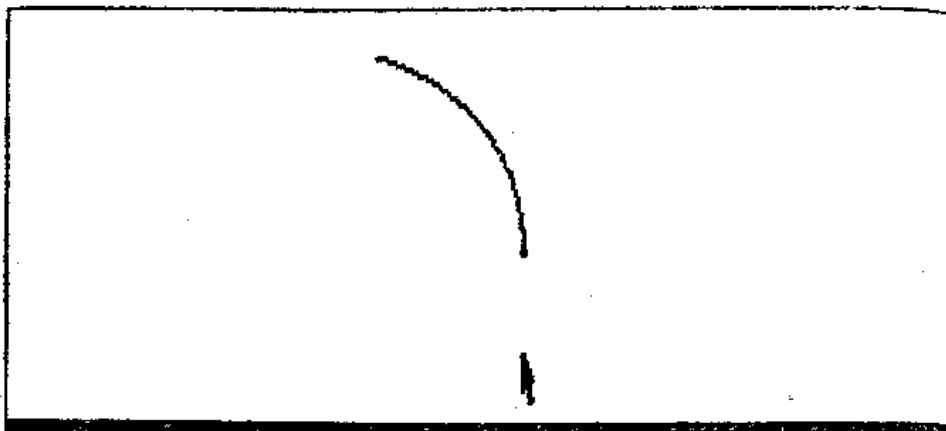


图 11.9 用arctool画弧

像对`circletool`和`ellipsotool`那样，我们通过计算鼠标当前的位置和弧的中心点之间的差的绝对值来跟踪鼠标的运动。此外，`getimage()`和`putimage()`用于产生我们在一些其它绘图例程中使用的“异或”橡皮筋效果。`arctool`类程保存足够大的矩形区以包含全360度的弧。通过从`circletool`派生`arctool`，这些特征可用于`arctool`。

11.10 杂项绘图支援

我们已涉及到包含在`draw.cpp`中的所有绘图类程，但还没有结束。工具箱包含两个附加的类程：`texttool`和`clearwindow`。

第一个类程`texttool`允许你在绘图窗口中键入正文。为使用`texttool`，你可在想要开始键入正文的地方掀按左鼠标按钮。于是，一个垂直的条形将作为光标出现。此时，如果你键入正文，它将从垂直条形开始自左至右显示。当你键入完一块正文时，必须再掀按一次左鼠标按钮。

通过画“异或”线（正文的高度）产生垂直光标。例如，第一次在`startdrawing()`中画光标，可用语句。

```
setwritemode(XOR_PUT);
mouse.hide();
line(x-wl,y-wt,x-wl,y+textheight("S")-wt);
mouse.show();
setwritemode(COPY_PUT);
```

在大多数情况下，由`updatedrawing()`函数处理键入的正文。当键入字符时，修改正文光标位置并把字符添加到`string`，一个新的变量添加到`drawingtool`，以便该正文可在稍后用于应用程序。

类程`texttool`还允许你通过按Enter键键入一系列回车。对此特征的检测可在`updatedrawing()`的while循环中找到。它检测回车值CR，然后用以下语句模拟回车。

```
if (c == CR) {
    y += textheight("S") + 2;
    x = leftx;
    moveto(x-wl,y-wt);
    if (len < MAXSTRING-1) string[len++] = '\n';
}
```

如果发现回车，则按照正文的高度向下移动正文光标，并使当前位置置于原始`x`地位`leftx`，由`y`给出新的计算高度。

当前`texttool`不支持回退、垂直正文、或BGI的很多其它功能强的正文特征。

`clearwindow`类程也是补充的绘图工具。它用于擦除整个绘图窗口。这通过设置绘图窗口的观察端口，然后调用`clearviewport()`完成。虽然我们不需要把此操作封装在一个对象中，但这样做使得它可以有和其它绘图函数一致的用户接口。这在我们的应用程序中是重要的。

列表11.1 draw.h

```
// draw.h - Header file for the interactive drawing utilities
// used in the paint and CAD programs.
```

```

#ifndef DRAWH
#define DRAWH
#include "kbmouse.h"

// The drawingtool class is the base class which defines the basic format
// of the drawing tools
class drawingtool {
public:
    int x, y, oldx, oldy;           // Current and last mouse position
    int left, top, right, bottom;    // Bounds of the figure
    int drawcolor;                  // Color to be used in figure
    virtual void draw(void);         // High-level drawing function
    virtual void performdraw(void);  // Defines user interaction
    virtual void startdrawing(void); // Sets up for drawing
    virtual void updatedrawing(void); // Interactively draws figure
    virtual void finishdrawing(void); // Finalizes drawing
    virtual void hide(void) { }      // Removes figure from screen
    virtual void show(void) { }      // Displays or updates figure
    // The following functions will be defined in the CAD program
    virtual void display(void) { }
    virtual void getbounds(int &l, int &t, int &r, int &b) { }
    virtual void translate(double transx, double transy) { }
    virtual void rotate(double angle) { }
    virtual drawingtool *dup(void) { return this; }
};

class penciltool : public drawingtool { // A curve drawing class
public:
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void show(void);
};

const int ERASERSIZE = 5;           // Half the size of the eraser
class erasertool : public drawingtool { // An eraser class
public:
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void show(void);
};

const int SPRAYSIZE = 15;           // Size of the spray area
class spraycantool : public drawingtool { // A spraycan class
public:
    virtual void draw(void);
    virtual void updatedrawing(void);
    virtual void show(void);
};

class linetool : public drawingtool { // Draws single lines
public:
    int x1, y1, x2, y2;             // Endpoints of line drawn
    int linestyle;                  // Line style used for lines
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void finishdrawing(void);
    virtual void hide(void);
    virtual void show(void);
};

class rectangletool : public linetool { // Draws a rectangle
public:

```

```

    virtual void hide(void);
    virtual void show(void);
};

class fillrectangletool : public rectangletool { // A filled rectangle
public:
    int fillstyle;           // Fill pattern used in rectangle
    int fillcolor;          // Color used in fill pattern
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void finishdrawing(void);
};

const int MAXPOLYSIZE = 40; // Can have this many edges
class polygontool : public linetool { // Draws a polygon
public:
    int poly[MAXPOLYSIZE], numpts; // Not used here, but will be later

    int fillstyle;           // Fill pattern used
    int fillcolor;          // Color used in fill pattern
    virtual void startdrawing(void);
    virtual void updatedrawing(void);
    virtual void finishdrawing(void);
};

class fillpolygontool : public polygontool { // Draws filled polygon
public:
    virtual void startdrawing(void);
    virtual void finishdrawing(void);
};

class circletool : public drawingtool { // Draws a circle
public:
    int halfx;              // These three variables are used to
    void *covered1, *covered2; // save an image where circle is drawn
    int cleft, cright, cbottom, ctop, oldright; // Clipped figure
    int centerx, centery, absradiusx, oldleft, oldtop; // Figure's dimensions
    virtual void draw(void);
    virtual void startdrawing(void);
    virtual void updatedrawing(void);
    virtual void hide(void);
    virtual void show(void);
};

class ellipsetool : public circletool { // Draws an ellipse
public:
    int absradiusy;         // The y radius of ellipse. The
    virtual void updatedrawing(void); // x radius is inherited from
    virtual void show(void); // the circletool class.
};

class arcctool : public circletool { // Draws an arc
public:
    int radius, angle;      // Radius and sweep angle of arc
    virtual void updatedrawing(void);
    virtual void show(void);
};

const int MAXSTRING = 80; // A string can be this long
class texttool : public drawingtool { // Text entry class
public:

```



```

    int leftx, lefty;
    char string[MAXSTRING];           // Top-left location of text.
    int len;                          // String entered
    int textstyle;                    // Length of string
    void draw(void);                  // Font used
    void startdrawing(void);
    void updatedrawing(void);
    void finishdrawing(void);
};

class clearwindowtool : public drawingtool {
    virtual void draw(void); // Erases drawing window
};
#endif

void mallocerror(void); // Function called if memory allocation fails

// The fill parameters, drawing colors, and so forth are all kept as global
// variables. From time to time the current drawing, fill, and line
// styles may be changed, so before using any drawing function make sure
// the various drawing parameters are reset to these global values.
extern int globalfillstyle; // The fill style that should be used
extern int globalfillcolor; // Holds the fill color that should be used
extern int globaldrawcolor; // Holds the drawing color to use
extern int globallinestyle; // Holds the line style that should be used
extern int globaltextstyle; // The text font to use
extern int wl, wt, wr, wb; // Boundaries of the drawing window
extern int maxx, maxy; // Boundaries of the full screen
extern kbdmouseobj mouse; // A mouse object is used for input

```

列表11.2 draw.cpp

```

// draw.cpp -- Interactive drawing utilities used in the Paint and
// CAD programs.
#include <stdio.h>
#include <graphics.h>
#include <alloc.h>
#include <stdlib.h>
#include <time.h>
#include "gtext.h"
#include "kbdmouse.h" // Emulated mouse utilities
#include "draw.h" // Function prototypes for draw.cpp

// The fill parameters, drawing colors, and so forth are all kept as global
// variables. From time to time the current drawing, fill, and line
// styles may be changed, so before calling any drawing function make sure
// the various drawing parameters are reset to these global values.
int globalfillstyle; // The fill style that should be used
int globalfillcolor; // Holds the fill color that should be used
int globaldrawcolor; // Holds the drawing color to use
int globallinestyle; // Holds the line style that should be used
int globaltextstyle; // Specifies the text font to use
int wl, wt, wr, wb; // Window bounds where drawing is to be done
int maxx, maxy; // Maximum dimensions of the screen
kbdmouseobj mouse; // A keyboard mouse is used for interaction

// The high-level function that is called to draw a figure. It sets
// the viewport to the drawing window, calls performdraw() to
// perform the drawing action, and then restores the viewport to
// the full screen,

```

```

void drawingtool::draw(void)
{
    struct viewporttype view;

    getviewsettings(&view);           // Save the viewport coordinates
    setviewport(wl,wt,wr,wb,1);       // Set viewport to drawing window
    drawcolor = globaldrawcolor;      // Save the drawing color
    performdraw();                    // Draw the figure
    // Restore viewport coordinates to those saved earlier
    setviewport(view.left,view.top,view.right,view.bottom,view.clip);
}

// Drawing begins when the left mouse button is pressed. Then the
// coordinates of the mouse are checked to see if the mouse was pressed
// outside of the drawing window. If so, the drawing function ends.
// Otherwise, the drawing takes begins by calling startdrawing().
void drawingtool::performdraw(void)
{
    while (1) {
        while (!mouse.buttonpressed(LEFT_BUTTON)) : // Wait for button press
            mouse.getcoords(x,y); // Get location where button was pressed
        if (x < wl || y < wt || y > wb || y > wr) return; // Exit drawing
        startdrawing(); // Setup for the drawing
        updatedrawing(); // Do the drawing
        finishdrawing(); // Cleanup drawing as needed
    }
}

// Initializes the drawing style and various variables used by the
// drawing function. By default it saves the last location of the mouse.
void drawingtool::startdrawing(void)
{
    oldx = x; oldy = y;
}

// Complete the figure being drawn. Most of the derived classes will
// draw while the left mouse button is pressed. Note that the figure is not
// updated if the mouse has not been moved.
void drawingtool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) { // Button must be pressed
        mouse.getcoords(x,y); // Get current location of mouse
        if (x != oldx || y != oldy) { // Only update figure if the mouse
            hide(); // has moved. Remove part or all
            show(); // of the figure. Draw the figure.
            oldx = x; oldy = y; // Update saved position of mouse
        }
    }
}

// Perform any final operations that must be made to finish the drawing.
// For instance, figures that are drawn with exclusive-OR lines must
// be redrawn with the exclusive-OR feature disabled so that the figure's
// color will come out correctly. By default, do nothing.
void drawingtool::finishdrawing(void) { }

// Use the global drawing color for drawing with the pencil class
void penciltool::draw(void)
{
    setcolor(globaldrawcolor); // Switch to the global drawing color
    drawingtool::draw(); // Draw with the pencil
}

```

```

}

// Initialize drawing with pencil by moving the current position to
// the current location of the mouse. Adjust for the fact that the
// mouse coordinates are given with respect to the whole screen and
// the pencil tool's coordinates are relative to the drawing window.
void penciltool::startdrawing(void)
{
    drawingtool::startdrawing();    // Call drawingtool's startdrawing()
    moveto(x-wl,y-wt);              // Set the current position
}

// Draw a line from the last position of the mouse to its current position
void penciltool::show(void)
{
    mouse.hide();                   // Hide mouse before drawing
    lineto(x-wl,y-wt);              // Draw a line from last position
    mouse.show();                   // Restore the mouse cursor
}

// Use a solid, background color to erase a region of the screen
void erasertool::draw(void)
{
    setcolor(getbkcolor());         // Use the background color
    setfillstyle(SOLID_FILL,getbkcolor()); // to erase the screen
    drawingtool::draw();            // Draw with the eraser
}

// Erase the spot pointed to when the left mouse button is pressed
void erasertool::startdrawing(void)
{
    drawingtool::startdrawing();
    show();
}

// Erase a rectangular region of the screen where the mouse is located
void erasertool::show(void)
{
    mouse.hide();                   // Remove the mouse from screen
    bar(x-wl,y-wt,x-wl+ERASERSIZE,y-wt+ERASERSIZE); // Erase region
    mouse.show();                   // Restore the mouse cursor
}

// Initialize the random function to be used by the spraycan tool
// before performing the spray painting
void spraycantool::draw(void)
{
    randomize();                    // Initializes random function
    drawingtool::draw();            // Use spraycan
}

// Override drawingtool's updatedrawing() function so that the spraycan
// paints as long as the left mouse button is pressed--even if the
// mouse has not been moved.
void spraycantool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) {
        mouse.getcoords(x,y);       // Get current mouse location
        show();                     // Spray paint
    }
}

```

```

// Spray a small region of the screen. Two separate for loops are
// used to randomly paint a series of pixels using the global
// drawing color.
void spraycantooll::show(void)
{
    // Remove the mouse before painting on the screen
    mouse.hide();
    for (int i=0; i<8; i++) // Randomly draw 8 pixels
        putpixel(x-random(SPRAYSIZE)+5-wl,
            y-random(SPRAYSIZE)+5-wt,globaldrawcolor);
    for (i=0; i<8; i++) // Randomly draw another 8 pixels
        putpixel(x-random(SPRAYSIZE-2)+3-wl,
            y-random(SPRAYSIZE-2)+3-wt,globaldrawcolor);
    mouse.show(); // Restore the mouse cursor to the screen
}

// Set the line styles and color before drawing
void linetool::draw(void)
{
    setlinestyle(globallinestyle,0,NORM_WIDTH); // Use global line style
    setcolor(globaldrawcolor); // Use global draw color
    linestyle = globallinestyle; // Save line style used
    drawingtool::draw(); // Draw one or more lines
}

// Use exclusive-ORed lines while the line is initially drawn
void linetool::startdrawing(void)
{
    drawingtool::startdrawing();
    setwritemode(XOR_PUT); // Use exclusive-OR lines while drawing
    x1 = x; y1 = y; // This will be the stationary point of the line
}

// Draw a line from fixed point where the mouse was pressed (which
// was saved in startdrawing()) to its current location. Save the
// current location of the mouse in (x2,y2).
void linetool::show(void)
{
    mouse.hide(); // Hide the mouse before drawing
    line(x1-wl,y1-wt,x-wl,y-wt); // Draw the line
    mouse.show(); // Restore the mouse to the screen
    x2 = x; y2 = y; // Save the line's other endpoint
}

// Erase the line by drawing it again. This works since the exclusive-OR
// mode is being used.
void linetool::hide(void)
{
    mouse.hide(); // Hide the mouse cursor
    line(x1-wl,y1-wt,oldx-wl,oldy-wt); // Erase the line
    mouse.show(); // Restore the mouse cursor
}

// The size of the line has been fixed, so switch out of exclusive-OR
// mode and draw the line in permanently by drawing it again.
void linetool::finishdrawing(void)
{
    setwritemode(COPY_PUT); // Get out of exclusive-OR mode
    show(); // Draw the line in permanently
}

```

```

// Drawing a rectangle is similar to drawing a line, except that
// rectangle() is called rather than line().
void rectangletool::show(void)
{
    mouse.hide(); // Hide the mouse cursor
    rectangle(xl-wl,y1-wt,oldx-wl,oldy-wt); // Draw the rectangle
    mouse.show(); // Show the mouse cursor
}

// Remove the current rectangle by drawing it again. This function
// assumes that the exclusive-OR feature is being used.
void rectangletool::hide(void)
{
    mouse.hide(); // Hide the mouse cursor
    rectangle(xl-wl,y1-wt,x-wl,y-wt); // Erase the rectangle
    mouse.show(); // Show the mouse cursor
}

// Setup for drawing a filled rectangle by selecting fill settings
// before performing the drawing operation
void fillrectangletool::draw(void)
{
    setfillstyle(globalfillstyle,globalfillcolor);
    rectangletool::draw();
}

// Use a dashed rectangle while setting the size of the filled rectangle
void fillrectangletool::startdrawing(void)
{
    setlinestyle(DASHED_LINE,0,NORM_WIDTH);
    rectangletool::startdrawing();
}

// Draw the filled rectangle
void fillrectangletool::finishdrawing(void)
{
    setlinestyle(globallinestyle,0,NORM_WIDTH);
    setwritemode(COPY_PUT); // Switch out of exclusive-OR mode
    mouse.hide();
    bar3d(xl-wl,y1-wt,x-wl,y-wt,0,0); // Draw the filled rectangle
    mouse.show();
}

// Save the starting location of the polygon. These coordinates are
// saved in the same array, poly.
void polygontool::startdrawing(void)
{
    linetool::startdrawing();
    fillcolor = globalfillcolor;
    fillstyle = globalfillstyle;
    poly[0] = x-wl; poly[1] = y-wt; // Save current location of mouse
    numpts = 2; // One coordinate pair is saved
}

// Override updatedrawing() so that the left mouse button is used to
// specify the location of vertices and the right mouse button ends
// the drawing of the polygon.
void polygontool::updatedrawing(void)
{
    do {
        mouse.getcoords(x,y); // Get current location of mouse
    }
}

```

```

    if (x != oldx || y != oldy) { // If mouse hasn't moved, update
        hide(); // current edge of polygon
        show();
        oldx = x; oldy = y; // Remember last location of edge
    }
    if (mouse.buttonreleased(LEFT_BUTTON)) {
        mouse.buttonpressed(LEFT_BUTTON);
        setwritemode(COPY_PUT); // Draw line in permanently
        show();
        setwritemode(XOR_PUT); // Switch back to exclusive-OR mode
        x1 = x; y1 = y; // Save the new vertex of the line
        poly[numpts++] = x-wl; poly[numpts++] = y-wt;
    }
    // End drawing when right mouse button is pressed
} while (!mouse.buttonpressed(RIGHT_BUTTON));
}

// Close off polygon by drawing a line back to its beginning
void polygontool::finishdrawing(void)
{
    poly[numpts++] = x-wl;
    poly[numpts++] = y-wt;
    poly[numpts++] = poly[0];
    poly[numpts++] = poly[1];
    setwritemode(COPY_PUT); // Draw line in permanently
    mouse.hide();
    line(x1-wl,y1-wt,x-wl,y-wt);
    line(x-wl,y-wt,poly[0],poly[1]); // Close polygon
    mouse.show();
}

// Prepare for drawing a filled figure before calling the polygon routine
void fillpolygontool::startdrawing(void)
{
    setfillstyle(globalfillstyle,globalfillcolor);
    polygontool::startdrawing();
}

// Draw a filled polygon
void fillpolygontool::finishdrawing(void)
{
    poly[numpts++] = x-wl;
    poly[numpts++] = y-wt;
    poly[numpts++] = poly[0];
    poly[numpts++] = poly[1];
    setwritemode(COPY_PUT); // Draw line in permanently
    mouse.hide(); // Hide the mouse cursor
    fillpoly(numpts/2,poly); // Draw the filled polygon
    mouse.show(); // Show the mouse cursor
}

// The getimage() and putimage() functions are used to create a rubber-
// banding circle effect by popping up a "window" with a circle in it.
// This function allocates and frees the memory used to save the screen
// where the circle is displayed. Since some high-resolution modes
// consume more than 64K, the drawing window is saved in two pieces,
// covered1 and covered2.
void circletool::draw(void)
{
    halfx = (wr + wl) / 2; // Split drawing window into two
    covered1 = malloc(imagesize(wl,wt,halfx,wb)); // Allocate memory

```

```

covered2 = malloc(imagesize(halfx,wt,wr,wb));
if (covered1 == NULL || covered2 == NULL) mallocerror();
setcolor(globaldrawcolor);          // Prepare for drawing
drawingtool::draw();                 // Draw one or more circles
free(covered2);                      // Free the memory allocated earlier
free(covered1);
}

// Start drawing by saving the point where the mouse cursor is located
void circletool::startdrawing(void)
{
    drawingtool::startdrawing();
    centerx = x; centery = y;
    oldleft = x; oldtop = y; oldright = x;

    mouse.hide();
    getimage(oldleft-wl,oldtop-wt,oldleft-wl,oldtop-wt,covered1);
    getimage(oldleft-wl,oldtop-wt,oldleft-wt,oldtop-wt,covered2);
    mouse.show();
}

// Draw a circle while the left mouse button is pressed. Need to
// calculate the bounds of the drawing window to be saved. These
// values are stored in cleft, cright, ctop, and cbottom. The radius
// of the circle is changed as the mouse is moved left and right.
void circletool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) {
        mouse.getcoords(x,y);
        absradiusx = abs(centerx - x); // Calculate new radius of circle
        // Clip the region that must be saved below the circle
        // to the boundaries of the drawing window
        if (centerx-absradiusx < wl) cleft = wl;
        else cleft = centerx - absradiusx;
        if (centerx+absradiusx > wr) cright = wr;
        else cright = centerx + absradiusx;
        if (centery-absradiusx < wt) ctop = wt;
        else ctop = centery - absradiusx;
        if (centery+absradiusx > wb) cbottom = wb;
        else cbottom = centery + absradiusx;
        if (x != oldx) { // If the size of the circle has changed, redraw it
            hide();
            show();
            oldleft = cleft; oldtop = ctop; oldx = x; oldright = cright;
        }
    }
}

// Remove the current circle by overwriting it with the stored screen
// image that was saved before the circle was drawn
void circletool::hide(void)
{
    mouse.hide();
    putimage(oldleft-wl,oldtop-wt,covered1,COPY_PUT);
    putimage((oldleft-oldright)/2-wl,oldtop-wt,covered2,COPY_PUT);
    mouse.show();
}

// Draw the circle, but first save the screen region where the circle
// will be drawn. This screen image is used in hide() to erase the circle.
void circletool::show(void)
{

```

```

mouse.hide();
getImage(cleft-wl,ctop-wt,(cleft+cright)/2-wl,cbottom-wt,ccovered1);
getImage((cleft+cright)/2-wl,ctop-wt,cright-wl,cbottom-wt,ccovered2);
if (absradiusx != 0) // Draw circle if its radius is not zero
    circle(centerx-wl,centery-wt,absradiusx); // Draw circle
mouse.show();
}

// Drawing an ellipse is similar to drawing a circle except that
// when the mouse is moved up and down it also changes the vertical
// height of the figure
void ellipsetool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) {
        mouse.getcoords(x,y);
        absradiusx = abs(centerx - x); absradiusy = abs(centery - y);
        // Clip the region that must be saved below the ellipse
        // to the boundaries of the drawing window
        if (centerx-absradiusx < wl) cleft = wl;
        else cleft = centerx - absradiusx;
        if (centerx+absradiusx > wr) cright = wr;
        else cright = centerx + absradiusx;
        if (centery-absradiusy < wt) ctop = wt;
        else ctop = centery - absradiusy;
        if (centery+absradiusy > wb) cbottom = wb;
        else cbottom = centery + absradiusy;
        // If the size of the ellipse has changed, redraw it
        if (x != oldx || y != oldy) {
            hide();
            show();
            oldleft = cleft; oldtop = ctop; oldright = cright;
            oldx = x; oldy = y;
        }
    }
}

// Display the ellipse, but first save the screen region where the
// ellipse is to be drawn so that the ellipse can later be erased
// by overwriting the screen with these saved images.
void ellipsetool::show(void)
{
    mouse.hide();
    getImage(cleft-wl,ctop-wt,(cleft+cright)/2-wl,cbottom-wt,ccovered1);
    getImage((cleft+cright)/2-wl,ctop-wt,cright-wl,cbottom-wt,ccovered2);
    if (absradiusx != 0 && absradiusy != 0) // Draw ellipse
        ellipse(centerx-wl,centery-wt,0,360,absradiusx,absradiusy);
    mouse.show();
}

// The radius is changed by moving the mouse left and right. The
// sweep angle of the arc changes as the mouse is moved up and
// down.
void arctool::updatedrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) {
        mouse.getcoords(x,y);
        radius = abs(centerx - x);
        angle = abs(centery - y);
        if (centerx-radius < wl) cleft = wl;
        else cleft = centerx - radius;
        if (centerx+radius > wr) cright = wr;
        else cright = centerx + radius;
    }
}

```



```

    if (centery-radius < wt) ctop = wt;
    else ctop = centery - radius;
    if (centery+radius > wb) cbottom = wb;
    else cbottom = centery + radius;
    // If the size of the ellipse has changed, redraw it
    if (x != oldx || y != oldy) {
        hide();
        show();
        oldleft = cleft; oldtop = ctop; oldright = cright;
        oldx = x; oldy = y;
    }
}

// Draw the arc using the current radius and angle. See the
// discussions for the similar functions in circletool and ellipsetool/
// for how this function works.
void arctool::show(void)
{
    mouse.hide();
    getimage(cleft-wl,ctop-wt,(cleft+cright)/2-wl,cbottom-wt,covered1);
    getimage((cleft+cright)/2-wl,ctop-wt,cright-wl,cbottom-wt,covered2);
    if (radius > 0 && angle > 0)
        arc(centerx-wl,centery-wt,0,angle,radius); // Draw the arc
    mouse.show();
}

// Writes text to the drawing window. A vertical line is used as a cursor.
// The text is saved in the field "string." Use the global drawing
// color as the color of the text.
void texttool::draw(void)
{
    setcolor(globaldrawcolor);
    settextjustify(LEFT_TEXT,TOP_TEXT);
    textstyle = globaltextstyle; // Remember which font is being used
    drawingtool::draw();
}

void texttool::startdrawing(void)
{
    while (!mouse.buttonreleased(LEFT_BUTTON)) :
        drawingtool::startdrawing();
    setwritemode(XOR_PUT);
    mouse.hide(); // Exclusive-OR in a cursor where mouse was pressed
    line(x-wl,y-wt,x-wl,y+textheight("S")-wt);
    mouse.show();
    setwritemode(COPY_PUT); // Make sure to use the right mode when
    moveto(x-wl,y-wt); // displaying text
    leftx = x; lefty = y; // Save starting location of text
    len = 0;
}

// Main function to enter text. It displays characters
// entered until the left mouse button is pressed.
void texttool::updatedrawing(void)
{
    int c;
    char buff[2];

    while (1) {
        c = mouse.waitforinput(LEFT_BUTTON);

```

```

    if (c < 0) break;
    mouse.hide();
    setwritemode(XOR_PUT); // Erase the cursor
    line(x-wl,y-wt,x-wl,y+textheight("S")-wt);
    setwritemode(COPY_PUT);
    mouse.show();
    if (c == CR) { // If input is a carriage return
        y += textheight("S") + 2; // then move to another line and
        x = leftx; // insert a newline character into
        moveto(x-wl,y-wt); // the string array.
        if (len < MAXSTRING-1) string[len++] = '\n';
    }
    else {
        sprintf(buff,"%c",c);
        mouse.hide();
        outtext(buff); // Write the new character
        mouse.show();
        x += textwidth(buff); // Keep track of the cursor location
        if (len < MAXSTRING-1) string[len++] = c; // Add character to string
    }
    setwritemode(XOR_PUT); // Write a new cursor
    mouse.hide();
    line(x-wl,y-wt,x-wl,y+textheight("S")-wt); // Display new cursor
    mouse.show();
    setwritemode(COPY_PUT);
}

// Finish the text input routine by adding a NULL character to the
// end of the string entered and erase the cursor.
void texttool::finishdrawing(void)
{
    setwritemode(XOR_PUT); // Erase the cursor
    mouse.hide();
    line(x-wl,y-wt,x-wl,y+textheight("S")-wt);
    mouse.show();
    setwritemode(COPY_PUT);
    string[len] = '\0'; // NULL-terminate string
}

// Erase the drawing window
void clearwindowtool::draw(void)
{
    viewporttype v;

    getviewsettings(&v);
    setviewport(wl,wt,wr,wb,1); // Set viewport to drawing window
    mouse.hide();
    clearviewport(); // Erase the drawing window
    mouse.show();
    // Restore viewport settings
    setviewport(v.left,v.top,v.right,v.bottom,1);
}

// If an error occurs in memory allocation call this routine and
// an error message will be displayed and the program will quit
void mallocerror(void)
{
    closegraph();
    printf("Not enough memory to run program\n\n");
    exit(1);
}

```

第十二章 画画程序

本章有两个互联的目标。第一，它说明怎样建立在前几章已为之开发了一些类程的画画程序。第二，它作为画画程序的用户指南。此时，你应当备有图形编程工具的功能强的程序包，因为建立有效的画画程序不是一般的任务，我们很大程度上依靠这些工具以简化处理。在完成本章之前，我们还讨论你可能试图实现的一些增强能力，使得你可以建立自己的画画程序的定制版本。

12.1 画画程序综述

虽然我们的画画程序使用了在整个这本书中开发的很多工具，但在本章中我们仍需要开发更多的代码以建立此程序。由一系列特定文件处理遗漏的组成成分。第一组，`usertool.h`和`usertool.cpp`（列表12.1和列表12.2）提供我们在画画程序中需要的少量附加的交互工具，包括为显示下拉菜单所用的类程，一组填充图案和一个调色板。第二组是我们构造的两个源文件`interact.h`和`interact.cpp`列表（11.3和列表11.4），把在`draw.cpp`和`usertool.cpp`中的工具与屏幕上的图符和菜单任选项连接在一起。我们建立的最后一个文件是`paint.cpp`。它负责通过准备图符，为绘图窗口把屏幕分段，和建立绘图对象的内部表，绘制画画程序的环境。它也初始化响应用户输入的循环。在本章结尾的列表12.5中可找到`paint.cpp`。

12.1.1 使用屏幕对象

图12.1示出画画程序环境的屏幕镜头。如所指出的，为绘图窗口保留大部分的屏幕。

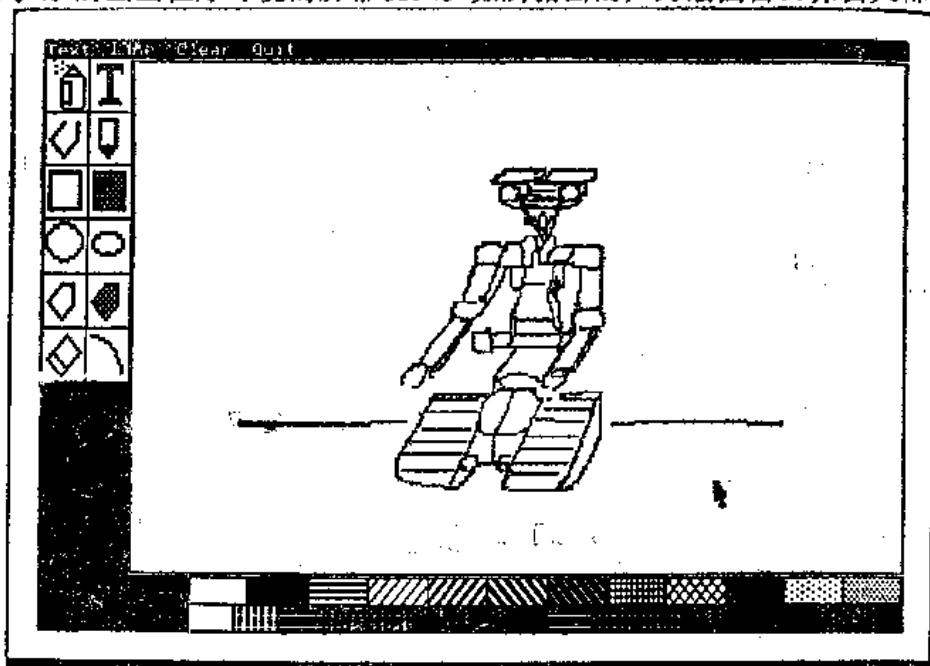


图 12.1 `paint.cpp`程序

但更使人感兴趣的是绘图环境的其它组成成分。为表示程序支持的命令的图符保留在屏幕的左边。此外,屏幕的顶部包含用于下拉各种菜单或选择附加函数的一些关键字。在屏幕的底部是在当前视频模式下可存取的填充图案和颜色。

实现的重要事情是,至今所描述的每一项都设计成作为interact.cpp文件管理的各自独立的图形对象来处理。因此,这些屏幕对象的每一个(图符、字等等)有预先定义的功能,此功能可以通过掀按鼠标来访问。例如,当你掀按喷涂壶图符时,spraycantool可用于喷涂绘图窗口以内的图画。于是,在我们继续以前,必须建立interact.cpp程序包,把这些屏幕对象和它们的操作连接在一起。

interact.cpp源文件包含在表12.1中列出的三个新类程。第一个类程interact把屏幕对象与特殊的绘图函数连接在一起,它的类程定义是:

```
class interact {
public:
    int cmd;           // Letter command that selects the object
    int left, right;    // Screen region where command is displayed
    int top, bottom;
    drawingtool *itool;
    interact(int c, int l, int t, int r, int b, drawingtool *tl);
    virtual void select(void);
    virtual void highlight(void);
};
```

interact的第一个实例变量指定可以使用的代码或字母,除了鼠标以外,可用于选择绘图对象。以下四个变量, left、right、top和bottom定义屏幕对象驻留的屏幕区域。此区域包含用户可以掀按以选择绘图工具的图符或菜单任选项。下一个存放指针itool。它置为当选择interact对象时执行的drawingtool对象。请注意,这里使用一个指针,使得我们可以充分利用多态性,以便调用从第十一章建立的绘图工具继承下来的任一成员函数。

interact类程还包含三个成员函数。第一个是用于初始化interact对象的构造器。它传送定义屏幕对象的屏幕区域的界,以及指向当选择屏幕对象时使用的绘图工具的指针。我们必须为在画画和CAD程序中使用的每一绘图工具建立interact对象。

其后的select()成员函数包含在interact中。它调用它含有的drawingtool对象:

```
void interact::select(void)
{
    highlight();           // Show that the option has been selected
    itool->draw();          // Execute the function selected
    highlight();           // Restore the option to its normal state
}
```

表 12.1 在interact.cpp中的类程

类 程	描 述
interact	把屏幕对象和动作组合在一起
userinteract	派生自interact,除了在选取屏幕对象时修改屏幕这点之外,做interact所做的一切事情
tootset	交互工具的数组以及选择和处理它们的代码

它是当我们想要选择特殊的绘图函数时必须调用的例程。请回忆一下, draw()是

用于初始化drawingtool绘图的高级函数。提供对包括在select() 中的highlight() 的调用以突出用户选择的屏幕对象。

当前, highlight() 仅仅反转屏幕对象的映像。因此, 第一次调用反转图符或菜单任选项中的颜色, 而第二次调用把屏幕对象恢复为它原来的映像。

interact类程提供我们为使用绘图工具封装屏幕对象所需的每样东西。但是, 有这样一些情况, 像对屏幕调色板, 我们在选择它时, 不想反转屏幕对象。为此, interact.cpp还包括派生的类程userinteract。它继承interact中的每样东西, 除了它重设select(), 使得只调用嵌套的drawingtool的draw() 成员函数, 而不出现强调动作:

```
virtual void select(void) { itool->draw(); }
```

你会找到列表12.3中示出的在interact.h标题文件的userinteract类程说明中的这个语句。

现在我们有类程的层次, 该层次允许我们从屏幕选择绘图工具, 我们需要建立一个把这一切连接到一个程序包的类程。这是在interact.cpp中toolset类程的作业。如下所示, toolset包含一组指向称为tool的interact对象的指针, 和管理此数组的一组三个的成员函数:

```
const int NUMTOOLS = 21;
class toolset {
public:
    interact *tool[NUMTOOLS];    // list of tools
    int numtools;                // Number of tools in list
    toolset(void) { numtools=0; }
    void add_tool(interact *t);
    void anytoprocess(int c);
}
```

请注意, 我们通过使用整型常数NUMTOOLS仅对21个交互对象分配空间, 这刚好大到足以对付我们开发的画画程序和CAD程序。

在toolset中的numtools变量记录tool数组中的对象数目。在toolset构造器中它初始化为0。如何把对象放在数组中?这是add_tool() 成员函数的任务。说明如下:

```
void toolset::add_tool(interact *t)
{
    if (numtools >= NUMTOOLS) return; // No more room in list

    tool[numtools++] = t;
}
```

我们必须为想要包含在应用程序中的每一交互工具调用add_tool()。例如, 为添加一个喷涂对象和它的图符(从(4, 10)到(36, 42))到绘图工具表, 首先需要说明一个toolset对象和一个喷涂壶对象:

```
toolset tools;
drawingtool *spraycan = new spraycantool;
```

然后调用add_tool() 把喷涂壶工具添加到toolset维持的表中:

```
tools.add_tool(new interact('s',4,10,36,42,spraycan));
```

在画画程序和CAD程序中使用的toolset的最后一个成员函数anytoprocess确定选择了哪一个绘图工具，若有，则执行它。函数分成两种情况，并且是在if-else语句周围建立的。执行哪一部分if语句取决于传送到anytoprocess()的c的值。此变量或是键盘或是鼠标事件代码，可预料从第八章鼠标工具中我们开发的waitforinput()返回它。如果c大于0，则出现键盘事件；如果它小于0，则按了鼠标按钮。最后，如果它等于0，则意味着没出现事件，不做什么事情。

对于anytoprocess()的顶部处理，有人键入字符代码以选择一个对象的情况。其次，十分类似，当出现鼠标动作时，使用它检索鼠标当前的坐标，然后搜索工具数组，以便发现在按鼠标的地方是否找到任一屏幕对象。如果是，则执行工具的对应select()成员函数：

```
else if (c < 0) {           // Mouse button pressed
    mouse.getcoords(x,y);    // Find which region was selected
    for (i=0; i<numtools; i++) {
        if (mouse.inbox(tool[i]->left,tool[i]->top,
                        tool[i]->right,tool[i]->bottom,x,y)) {
            tool[i]->select(); // Execute the function
            return;
        }
    }
}
```

12.1.2 建立环境

上节讨论的所有组成成分由在paint.cpp中可找到的setup_screen()函数产生。虽然这是一个长函数，但如果分成若干可管理的片段，那是很容易理解的：

setup_screen()的第一部分负责在屏幕的顶部建立和绘出条形主菜单，它使用以下语句

```
h = textheight("H") + 2;
setfillstyle(SOLID_FILL,EGA_BLUE);
bar3d(0,0,maxx,h,0,0);
outtextxy(2,2,"Text Line Clear Quit");
```

在跟随这些语句的那些行上建立条形菜单的功能。随后的一些语句把每一条形菜单的命令(Text, Line, Clear和Quit)添加到屏幕上的对象表中。这通过使用在上一节讨论的面向对象的实用程序interact.cpp完成。这里我们使用称为tools的toolset对象。它在interact.cpp中说明。添加一个对象到tools表是简单调用add_tool()成员函数。

当选择这些正文图符之一时，我们将编写函数以下拉适当的菜单。请注意，这些菜单项目之下的位置保存在setup_screen()中的一对全程变量中，以便我们后来知道下拉菜单放在何处。例如，在text任选项的情况下，下拉菜单出现在textwindowx和textwindowy。这些变量在usertool.cpp中定义。

下面的语句序列显示在屏幕左边的一系列图符，并把对应的绘图对象增补到tool对象中。这些图符的每一个放在各自独立的文件中，在我们开发图符编辑程序时，遵循第九章

介绍的命名约定。

显示图符是个两步过程。首先，从它的文件读图符并显示（这由在`usertool.cpp`中的函数`read_icon()`完成）。其次，图符通过调用`add_tool()`添加到屏幕上的对象表中。这些图符用于调用我们在第十一章建立的绘图函数。

在`setup_screen()`中的下一块代码建立填充图案和在屏幕底部的彩色调色板。为了使我们的环境尽可能地一致，通过激活从`drawingtool`派生的两个新类程之一使用填充图案和调色板。这些类程（`change_fill_pattern_tool`和`change_fill_color_tool`）设置在`usertool.cpp`源文件里，而它们的定义包含在标题文件`usertool.h`中。

像早先对图符和菜单对象所作的那样，我们把这些对象添加到我们的工具表中，并且将通过调用它们的`draw()`函数激活它们。如果考查它们的每一类程的定义，事实上，你会发现我们所做的一切是重设`drawingtool`的`draw()`成员函数，并用代码替换它以便选择填充图案和调色板中的颜色。

由于屏幕的大小可以根据使用的图形模式改变，所以在这些类程中的`draw()`函数要比在其它情况下的稍复杂些。要注意的主要事情是，每一例程首先计算每一单元（例如，单个填充）需要多宽，以便使所有单元适合整个屏幕。然后用`for`循环顺序通过每一图案或每种颜色，直到它们全部都显示为止。还请注意，我们没有单独地把每一单元添加到我们的对象表中，而是作为两个独立的对象添加完整的填充图案块和颜色块。

在画画程序环境中的这些填充类型的左边是显示当前填充图案和颜色的矩形区域。此区域也作为一个对象`change_draw_color_tool`添加，它用作我们的工作台以改变绘图颜色。例如，每当你掀按此区域时，当前填充颜色将变成当前绘图颜色。如同我们对填充图案和调色板工具所做的那样，此动作由在`change_draw_color_tool`中的`draw()`成员函数完成。

`setup_screen()`完成的最后一个操作是计算和画出绘图窗口的边界。按照`draw.cpp`的要求，这些边界保存在全程变量`wl`、`wt`、`wr`和`wb`中。

12.2 画画函数

虽然我们有在控制下的画画程序的基本形式，但仍需要萌发少量的函数。这些函数全部包装在`usertool.h`和`usertool.cpp`文件中。我们已经简单地论及它们中的一些，例如`change_fill_pattern_tool`和`change_fill_color_tool`，但是仍需要涉及在`usertool.cpp`中的这些和其它成员函数的更多细节。

表 12.2 `usertool.cpp`中的画画程序支撑类程

类	程	描 述
	<code>quittool</code>	终止画画程序
	<code>change_fonttool</code>	选择新的字形类型
	<code>change_line_styletool</code>	选择新的线型
	<code>change_fill_patterntool</code>	选择新的画画图案
	<code>change_fill_colortool</code>	选择新的画画颜色
	<code>change_draw_colortool</code>	置绘图颜色为填充颜色

表12.2列出usertool.cpp中的支撑类程表。我们将从考查与条形主菜单中的Quit任选项有关的函数开始，然后继续进行其它工作。请记住，通过使用第十一章的draw.cpp程序包中的类程，已经涉及到与图符对应的绘图工具、正文键入任选项和清除窗口函数。

quittool类程提供唯一的路径退出画画程序。当选择它时，通过调用exit()简单地退出图形模式和终止程序。它作为派生的drawingtool在usertool.cpp中实现，使得我们可以把它添加到绘图操作的表中，并以和所有其它工具相同的方法选择它。当然，quittool不做任何绘图工作，而是重设draw()函数以执行终止程序的语句。

12.3 下拉菜单

正如早先指出的，画画程序使用下拉菜单改变正文和线型。我们可以开发一个通用的菜单程序包以支持下拉菜单。但这可能导致转到更为基础的成分，并因而延误我们建立可用的画画程序。取而代之，为简化把下拉菜单添加到我们程序的任务，我们使用比较死板的编码方法。

用于正文和线型的下拉菜单通过揸按字Text或Line存取。当选择这些任选项之一时，通过我们的面向对象实用程序interact.cpp调用类程changefonttool或changelinestyletool。

重设这两个类程的draw()函数以提供在下拉菜单中的用户交互作用。每一个这些函数的开头是建立下拉菜单的几个语句。进程从使用gpopup()上弹菜单窗口开始。请注意，usertool.cpp说明全局窗口对象w以供使用。

其次，通过for循环用适当的选择填充窗口。例如，如果选择Text任选项，则显示不同的字形类型，以便你可以选择一个。类似地，如果选择了线的下拉菜单，则示出一组可用的线型。图12.2示出带有可视的Line下拉菜单的画画环境。

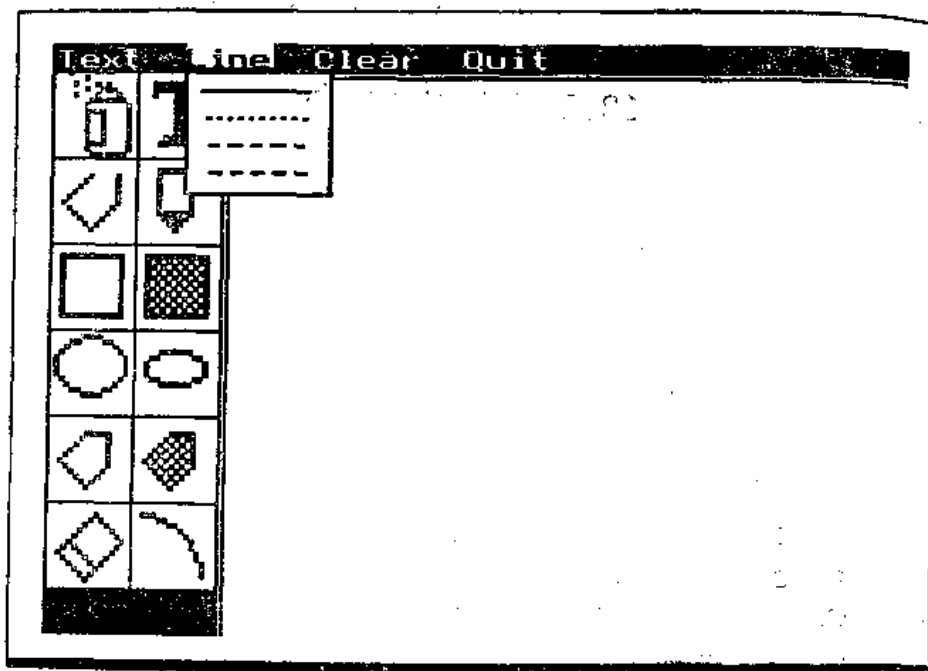


图 12.2 线的下拉菜单

下拉菜单任选项的位置被固定地编码,并且定义在函数 `changefonttool.draw()` 和 `changelinestyletool.draw()` 中。因此,当你掀按字形类型之一(在正文菜单情况下)或线段之一(在线的下拉菜单情况下)时,程序可以确定已选择了哪一个任选项并采取适当的动作。

用在 `changefonttool.draw()` 和 `changelinestyletool.draw()` 中的第二个 `for` 循环制定已选取哪一个菜单选择。但是,首先这两个函数都使用下面的语句等待你掀按左鼠标按钮:

```
while (!mouse.buttonpressed(LEFT_BUTTON));
```

一旦按下按钮,则检索鼠标坐标并与菜单中的可能选择进行比较,以便知道是否已选了一个。如果是,则改变适当的正文类型或线型并从函数返回。如果鼠标坐标不与菜单选择相对应,则 `draw()` 返回而不执行任何动作。

12.4 改变填充类型

屏幕的底部是可以在画画程序中使用的一系列填充图案和颜色。为易于把这些实用程序综合到我们已有的程序包,填充图案和彩色调色板作为两个对象而不是一系列较小的对象添加到对象表中。因此,如果在填充图案里掀按鼠标时,无论你选择了哪一个,都使用 `changefillpattern` 类。另一方面,如果你选择一个颜色框,则使用 `changefillcolor` 类。

这些类更像 `changefonttool` 和 `changelinestyletool` 那样工作,它们把按下按钮时的鼠标坐标与每一区域的位置作比较。如果鼠标选择一个框,则用该框的填充图案或颜色置 `globalfillstyle` 或 `globalfillcolor`, 它们是 `draw.cpp` 中定义的两个全局绘图参数。

12.5 用户交互作用

用于处理和控制用户的画画程序输入的代码放在 `paint.cpp` 文件中 `main()` 函数底部的 `do` 循环中。此无限循环调用鼠标成员函数 `waitforinput()` 以得到某些用户输入,然后接着调用 `anytoprocess()` 以执行适当的动作(如果有的话)。

```
do {  
    c = mouse.waitforinput(LEFT_BUTTON);  
    tools.anytoprocess(c);  
} while(1);
```

在第八章开发的文件 `mouse.cpp` 中可找到 `waitforinput()` 函数。如所书写的那样,此函数在返回之前等待要按的左鼠标按钮或要敲打的键。动作的值返回在整型变量 `c` 中。

输入值传送到 `anytoprocess()`, 在 `interact.cpp` 中定义类 `toolset` 的成员函数。它步进对象表并检测用户是否在定界对象的区域掀了按钮,或是否按了快速存取键。如果是,则函数调用与它相关的适当例程。

请注意, `main()` 中的 `do` 循环是无限循环。无限循环在通常称为事件驱动循环中起作用(事件可以是鼠标掀按或键盘动作)。也就是说,当循环等待事件出现时,重复循环。

请记住, quittool对象为退出程序提供路径。因此,可能出现的事件之一是用户可以选择Quit选项或键入字母q。它们依次调用quittool.draw()。

12.6 编译画画程序

如前面提到的,我们的画画程序使用在本书前几章开发的一些工具。在表12.3中示出了这些文件和讨论它们的章节的列表。为了编译和连接画画程序,你需要建立包含每一这些文件的工程文件(你还需要头文件)。请注意,由于画画程序中要求的存储量和“它自身的大小”有关,你必须在大量存储模型下编译所有的文件。

表 12.3 画画程序中所用文件的描述

源 文 件	章	描 述
gtext.cpp	4	图形模式正文实用程序
mouse.cpp	8	鼠标实用程序
kbmouse.cpp	8	键盘模拟的鼠标
gpopup.cpp	10	上弹窗口程序包
interact.cpp	12	面向对象的实用程序
usertool.cpp	12	杂项环境工具
draw.cpp	11	交互式绘图实用程序
paint.cpp	12	主画画程序

12.7 使用画画程序

一旦你有了一个编译和连接的画画程序版本,就可以试验一下它。在引用该程序以后,你应该了解的第一件事是在图12.1中示出的环境。如果你有某些困难,则确认一下在适当的位置上是否已有全部图符文件和图形库。

程序是很容易运行的。你可以使用如在第八章中概述的鼠标或键盘以围绕屏幕移动光标,也可以使用为图符定义的各种快速键。绘图函数应当像第十一章描述的那样工作。我们已添加的大多数其它函数已在本章中的前面描述过。只剩下一个问题:如何改变绘图颜色。

请记住,在屏幕底部的调色板用于直接改变填充颜色。为改变绘图颜色,你必须正好在说明当前填充设置的填充图案左边的框中揿按。通过揿按此框(它也被作为setup_screen()中的一个对象添加),你将迫使绘图颜色改变成正在屏幕上显示的绘图颜色。

12.8 增强画画程序

虽然完整的画画程序很大,但是你可能仍想添加许多特色。如果你有一些想法,应该说并不太难实现。这主要是因为该程序被设计成易于添加。

首先,让我们考查一下为添加通过图符可存取的新函数你需要做些什么。例如,假定

你想要添加一个例程，它将使用喷流填充操作来填充所选择的区域。基本上，有三件事你需要做：

- (1) 使用第九章的图符编辑程序建立反映新特征功能的图符。
- (2) 从实现要求特征的drawingtool派生一个类程。
- (3) 在setup_screen() 里增加对tools.add_tool()的调用，以安装你的新函数。

12.9 进行试验的一些想法

你可能想对画画程序做许多添加。其中最有可能的是增加一些函数，它们把当前绘图窗口的映像保存到磁盘并类似地从磁盘上读出映像。用这个办法，你可以间断地对各幅图画进行工作，并可以着手把你的工作归档。为增加此特征，你需要编写一个函数，它捕获屏幕信息并把它书写到磁盘上，以及需要编写一个伴随例程，以便读保存的屏幕映像并显示它。此外，你必须提供一个函数，此函数可以从用户获取假定要写或读的磁盘的文件名。为此，你可试图扩展包含在gtext.cpp中的ggets() 函数并把它与下拉窗口函数组合在一起。用此方式你可以建立一个下拉菜单，它可支持用户交互作用并在图形模式下工作。

另一个可能性是扩充某些现有的成员函数。例如，你可以修改正文例程texttool::draw()，使得用户可以使用回退键。此外，你可以修改texttool，使得它可以键入垂直的正文或自动变比或对齐正文。只用少量的工作就可以增加其中的一个。

最后的想法是增加一个函数，该函数允许用户去掉一部分映像并把它移动到别处。此特征可以通过以下的方法实现，即让用户划分绘图窗口的矩形区域，然后使用getimage() 和putimage() 把它移到另一位置上。

• 列表 12.1 usertool.h

```
#ifndef USERTOOLH
#define USERTOOLH
#include "draw.h"
#include "gpopup.h"
const int ICONW = 16;           // Icons are defined as 16 x 16
const int ICONH = 16;           // pixel patterns
const int FILL_HEIGHT = 16;     // All 12 of the BGI fill patterns
const int NUM_FILLS_WIDE = 12;  // are 16 pixels high in size
const int BORDER = 2;           // Two-pixel border is used
extern int textwindowx;          // Global values which specify where the
extern int textwindowy;          // font pull-down menu should appear
extern int linewindowx;          // Specify where the line pull-down menu
extern int linewindowy;          // should appear
extern int maxx, maxy;          // Maximum coordinates of drawing area
extern gwindows w;               // A window object for pop-up menus
extern unsigned icon[ICONH][ICONW]; // Holds an icon pattern
void read_icon(int x, int y, char *filename);
```

```

class quittool : public drawingtool {
    virtual void draw(void);
};
// User has selected one of the fill patterns, find which one it is
// and set globalfillstyle to this type.
class changefillpattern : public drawingtool {
    virtual void draw(void);
};
// User has selected one of the fill colors -- find which one it is
// and set globalfillcolor to this value.
class changefillcolortool : public drawingtool {
    virtual void draw(void);
};
// User has selected the drawcolor block. Change globaldrawcolor
// to the color used by globalfillcolor which is the current color
// in the drawcolor block.
class changedrawcolortool : public drawingtool {
    virtual void draw(void);
};
// User has selected text main menu option. Pull down the text menu
// and let user select one of the fonts. Load this new font, if
// any is selected.
const WINDOW_LINES = 5;
class changefonttool : public drawingtool {
public:
    virtual void draw(void);
};
// Pull-down line style menu. Change global linestyle to the line type
// selected from the menu, if any.
const LINE_HEIGHT = 10;
const LINE_WIDTH = 40;
const MAX_LINESTYLES = 4;
class changelinstyletool : public drawingtool {
public:
    virtual void draw(void);
};
#endif

```

• 列表 12.2 usertool.cpp

```

// usertool.cpp -- User interface tools used in the paint program.
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <graphics.h>
#include "draw.h"
#include "usertool.h"
#include "kbdmouse.h"
#include "gpopup.h"
#include "gtext.h"
int textwindowx;           // Global values which specify where the
int textwindowy;           // font pull-down menu should appear
int linewindowx;           // Specifies where the line pull-down menu
int linewindowy;           // should appear

```

```

gwindows w; // Declare a window object to be used in the pop-up menus
unsigned icon[ICONH][ICONW]; // Holds an icon pattern read from a file

void quittool::draw(void)
{
    closegraph();
    exit(0);
}

void changefillpatterntool::draw(void)
{
    int x, i, fill_width, filltype, mx, my;

    fill_width = (getmaxx()-1-ICONW*4+2) / (NUM_FILLS_WIDE+1);
    filltype = 0;
    for(i=0, x=ICONW*4+2+fill_width; i<NUM_FILLS_WIDE; i++, x+=fill_width) {
        mouse.getcoords(mx, my);
        if (mouse.inbox(x, getmaxy()-FILL_HEIGHT*2, x+fill_width,
            getmaxy()-FILL_HEIGHT, mx, my)) {
            globalfillstyle = filltype;
            setfillstyle(globalfillstyle, globalfillcolor);
            setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
            mouse.hide();
            // Show the new fill style
            bar3d(ICONW*4+2+2, getmaxy()-FILL_HEIGHT*2+2,
                ICONW*4+2+fill_width-2, getmaxy()-2, 0, 0);
            setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
            mouse.show();
            return;
        }
        filltype++;
    }
}

// User has selected one of the fill colors -- find which one it is
// and set globalfillcolor to this value.

void changefillcolortool::draw(void)
{
    int maxcolors, color_width, fillcolor;
    int fill_width, i, x, mx, my;

    fill_width = (getmaxx()-1-ICONW*4+2) / (NUM_FILLS_WIDE+1);
    maxcolors = getmaxcolor();
    color_width = (getmaxx()-1-ICONW*4+2-fill_width) / (maxcolors + 1);
    fillcolor = 0;
    for (i=0, x=ICONW*4+2+fill_width; i<=maxcolors; i++, x+=color_width) {
        mouse.getcoords(mx, my);
        if (mouse.inbox(x, getmaxy()-FILL_HEIGHT,
            x+color_width, getmaxy(), mx, my)) {
            globalfillcolor = fillcolor;
            setfillstyle(globalfillstyle, globalfillcolor);
            setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
            mouse.hide();
            // Show the new fill color
            bar3d(ICONW*4+2+2, getmaxy()-FILL_HEIGHT*2+2,
                ICONW*4+2+fill_width-2, getmaxy()-2, 0, 0);
            setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
            mouse.show();
            return;
        }
        fillcolor++;
    }
}

```

```

}
}

// User has selected the drawcolor block. Change globaldrawcolor
// to the color used by globalfillcolor which is the current color
// in the drawcolor block.
void changedrawcolortool::draw(void)
{
    int fill_width;

    fill_width = (getmaxx()-1-ICONW*4+2) / (NUM_FILLS_WIDE+1);
    globaldrawcolor = globalfillcolor;
    setfillstyle(globalfillstyle,globalfillcolor);
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(globaldrawcolor);
    mouse.hide();
    bar3d(ICONW*4+2+2,getmaxy()-FILL_HEIGHT*2+2,          // Show new color as
        ICONW*4+2+fill_width-2,getmaxy()-2,0,0);          // border to the block;
    setlinestyle(SOLID_LINE,0,NORM_WIDTH);
    mouse.show();
}

```

```

// User has selected text main menu option. Pull down the text menu
// and let user select one of the fonts. Load this new font, if
// any is selected.
void changefonttool::draw(void)
{
    struct textsettingstype savetext;

    int windowwidth, windowheight, offset, i, mx, my;
    static char *strings[WINDOW_LINES] = {"Default", "Triplex",
        "Small", "Sans Serif", "Gothic"};

    gettextsettings(&savetext);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    windowwidth = textwidth("Sans Serif") + BORDER;
    windowheight = (textheight("S")+2) * WINDOW_LINES + 3 * BORDER;
    settextstyle(savetext.font,savetext.direction,savetext.charsize);
    mouse.hide();
    w.gpopup(textwindowx,textwindowy,textwindowx + windowwidth + BORDER,
        textwindowy+windowheight,SOLID_LINE,getmaxcolor(),
        SOLID_FILL,BLACK);
    settextjustify(LEFT_TEXT,TOP_TEXT);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
    setcolor(getmaxcolor());
    offset = BORDER;
    for (i=0; i<WINDOW_LINES; i++) {
        gprintfxy(BORDER,offset,"%s",strings[i]);
        offset += textheight(strings[i]) + 2;
    }
    mouse.show();
    while (!mouse.buttonpressed(LEFT_BUTTON)) {
        mouse.getcoords(mx,my);
        offset = BORDER;
        for (i=0; i<WINDOW_LINES; i++) {
            if (mouse.inbox(textwindowx+BORDER,offset,
                textwindowx+textwidth(strings[i]),
                textwindowy+offset+textheight(strings[i])+2,mx,my)) {
                mouse.hide();
                w.gunpop();
                mouse.show();
                settextstyle(i,HORIZ_DIR,1);
            }
        }
    }
}

```

```

        globaltextstyle = i;
        while (!mouse.buttonreleased(LEFT_BUTTON)) :
            return;
    }
    offset += textheight(strings[i]) + 2;
}
mouse.hide();
w.gunpop();
mouse.show();
}

// Pull-down line style menu. Change globaltextstyle to the line type
// selected from the menu, if any.
void changelinstyletool::draw(void)
{
    int windowwidth, windowheight, offset, i, mx, my;

    windowwidth = LINE_WIDTH + BORDER*6;
    windowheight = LINE_HEIGHT * MAX_LINESTYLES;
    mouse.hide();

    w.gpopup(linewindowx, linewindowy, linewindowx + windowwidth + BORDER,
             linewindowy+windowheight+LINE_HEIGHT/2, SOLID_LINE, getmaxcolor(),
             SOLID_FILL, BLACK);
    setcolor(getmaxcolor());
    offset = BORDER;
    for (i=0; i<MAX_LINESTYLES; i++) {
        setlinestyle(i, 0, NORM_WIDTH);
        line(BORDER*3, offset+LINE_HEIGHT/2, BORDER*3+LINE_WIDTH,
             offset+LINE_HEIGHT/2);
        offset += LINE_HEIGHT;
    }
    mouse.show();
    while (!mouse.buttonpressed(LEFT_BUTTON)) :
        mouse.getcoords(mx, my);
        offset = BORDER;
        for (i=0; i<MAX_LINESTYLES; i++) {
            if (mouse.inbox(linewindowx+BORDER*3, offset-LINE_HEIGHT/2,
                           linewindowx+BORDER*3+LINE_WIDTH,
                           linewindowy+offset+LINE_HEIGHT/2, mx, my)) {
                mouse.hide();
                w.gunpop();
                mouse.show();
                setlinestyle(i, 0, NORM_WIDTH);
                globaltextstyle = i;
                while (!mouse.buttonreleased(LEFT_BUTTON)) :
                    return;
            }
            offset += LINE_HEIGHT;
        }
    mouse.hide();
    w.gunpop();
    mouse.show();
}

// Read an icon from a file and display it. Program quits if
// the icon file cannot be found.
void read_icon(int x, int y, char *filename)
{
    FILE *iconfile;
    int i, j, width, height, iconpixel;

```

```

if ((iconfile = fopen(filename, "r")) == NULL) {
    closegraph();
    printf("Could not find icon file=%s\n", filename);
    exit(1);
}
fscanf(iconfile, "%d %d", &width, &height);
if (width != ICONW || height != ICONH) {
    closegraph();
    printf("Incompatible icon file.\n");
    exit(1);
}

// Double the size of the icon so that it is a good size
for (j=0; j<ICONH; j++) {
    for (i=0; i<ICONW; i++) {
        fscanf(iconfile, "%x", &iconpixel);
        if (iconpixel == 1) {
            putpixel(2*i+x, y+2*j, WHITE);
            putpixel(2*i+1+x, y+2*j, WHITE);
            putpixel(2*i+x, y+2*j+1, WHITE);
            putpixel(2*i+1+x, y+2*j+1, WHITE);
        }
    }
}
fclose(iconfile);
rectangle(x, y, x+ICONW*2, y+ICONH*2);
}

```

• 列表 12.3 interact.h

```

// interact.h -- Header file for interact.cpp.
// Ties together a drawing tool and the screen region that the
// user selects to activate the function.
class interact {
public:
    int cmd;           // Letter command that selects the object
    int left, right;   // Screen region where command is displayed
    int top, bottom;
    drawingtool *itool;
    interact(int c, int l, int t, int r, int b, drawingtool *tl);
    virtual void select(void);
    virtual void highlight(void);
};

// This class provides access to a user interface's action function
// without changing anything on the screen
class userinteract : public interact {
public:
    // Note call to the base class constructor
    userinteract(int c, int l, int t, int r, int b, drawingtool *tl) :
        interact(c, l, t, r, b, tl) { }
    virtual void select(void) { itool->draw(); }
};

// The primary class used to provide the user interaction tools
// for the environment. It ties together a drawing function with
// a menu options, icon, and so on.
const int NUMTOOLS = 21;
class toolset {

```



```

public:
    interact *tool[NUMTOOLS];    // List of tools
    int numtools;                // Number of tools in list

    toolset(void) { numtools=0; }
    void add_tool(interact *t);
    void anytoprocess(int c);
};

extern toolset tools;

```

• 列表 12.4 interact.cpp

```

// interact.cpp -- Defines a class that ties together interface objects
// on the screen, such as icons and menu entries, with their operations.
#include <stdio.h>
#include <alloc.h>
#include <graphics.h>
#include "kbdmouse.h"
#include "draw.h"
#include "interact.h"

// Specify a code that can be used to select the object and a
// screen region that can be selected using the mouse to activate
// the drawingtool passed in as t
interact::interact(int c, int l, int t,
                  int r, int b, drawingtool *t1)
{
    cmd = c;    itool = t1;
    left = l; top = t; right = r; bottom = b;
}

// A particular drawingtool has been selected using the mouse. Call
// its highlight function to show that it has been activated and
// then call its draw() member function.
void interact::select(void)
{
    highlight();    // Show that the option has been selected
    itool->draw();    // Do the function selected
    highlight();    // Restore the option to its normal state
}

// Highlight the option on the screen by reversing the screen image
// where the option is placed. This is action can be applied to both
// icons and menu options.
void interact::highlight(void)
{
    // Allocate memory to be used to reverse screen option
    void *region = malloc(imagesize(left,top,right,bottom));
    if (region == NULL) return;    // Not enough memory
    mouse.hide();
    getimage(left,top,right,bottom,region); // Get the option's image
    putimage(left,top,region,NOT_PUT);    // Reverse it
    mouse.show();
    free(region);    // Free the memory used
}

// This function is called to add an object to the user object list
void toolset::add_tool(interact *t)
{
    if (numtools >= NUMTOOLS) return; // No more room in list
    tool[numtools++] = t;
}

```

```

toolset tools; // Declare a list of tools

// After a button press, search the icon list to see if
// the mouse is positioned over an object. If an object
// is found, execute the function associated with the object.
// The argument c specifies what action has just taken place.
// If it is a letter, then the if-statement is executed and
// the list is checked to see if any of the objects have a
// cmd code equal to c. If so, the function associated with
// that object is executed. The else part is executed when c<0
// which occurs when a mouse button has been pressed. It is
// similar to the code described above except that it uses
// the mouse as the selector.
void toolset::anytoprocess(int c)
{
    int i, x, y;

    if (c > 0) { // A letter command is supplied
        for (i=0; i<numtools; i++) { // Find which command the letter
            if (tool[i]->cmd == c) { // corresponds to
                tool[i]->select(); // Execute the drawing function
                return;
            }
        }
    }
    else if (c < 0) { // Mouse button pressed
        mouse.getcoords(x,y); // Find which region was selected
        for (i=0; i<numtools; i++) {
            if (mouse.inbox(tool[i]->left,tool[i]->top,
                tool[i]->right,tool[i]->bottom,x,y)) {
                tool[i]->select(); // Execute the function.
                return;
            }
        }
    }
}

```

• 列表 12.5 paint.cpp

```

// paint.cpp -- A paint program. This program exploits many of
// the BGI functions to build a simple yet powerful paint program.
// Several of the tools built up earlier in the book are put to
// use in this program. The paint program will run, without modification
// in most modes, although in some low-resolution modes you may
// want to make the icons smaller. The program uses kbdmouse.cpp
// so it supports either the mouse or the keyboard automatically.
// Most of the drawing functions used in paint.cpp are discussed
// in Chapter 10. Compile with large memory model.
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include "kbdmouse.h" // Keyboard/mouse support
#include "gtext.h" // Text in graphics mode tools
#include "gpopup.h" // Graphics pop-up window package
#include "draw.h" // The interactive drawing classes
#include "interact.h" // Object-oriented management of objects
#include "usertool.h" // Miscellaneous routines needed

```

```

void setup_screen(void);      // Function prototype

int main(void)
{
    int c, errcode, gmode, gdriver = DETECT;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    if ((errcode = graphresult()) != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errcode));
        exit(1);      // Return with error code
    }
    maxx = getmaxx(); maxy = getmaxy();
    setup_screen();
    mouse.init();
    do {
        c = mouse.waitforinput(LEFT_BUTTON);
        tools.anytoprocess(c);
    } while (1);
}

// Set up the environment for the paint program
void setup_screen(void)
{
    int h, i, x, fill_width, offset, space;
    int filltype, color_width, maxcolors, fillcolor;
    // Declare a set of drawing tools. Note that late binding
    // is used to get the power of polymorphism.
    drawingtool *changeFont = new changeFonttool;
    drawingtool *changeLineStyle = new changeLineStyletool;
    drawingtool *clearWorkarea = new clearWindowtool;
    drawingtool *quit = new quittool;
    drawingtool *pencil = new penciltool;
    drawingtool *eraser = new erasertool;
    drawingtool *spraycan = new spraycantool;
    drawingtool *linetl = new linetool;
    drawingtool *rect = new rectangletool;
    drawingtool *fillrect = new fillrectangletool;
    drawingtool *poly = new polygontool;
    drawingtool *fillpoly = new fillpolygontool;

    drawingtool *circletl = new circletool;
    drawingtool *ellipsetl = new ellipsetool;
    drawingtool *arctl = new arctool;
    drawingtool *text = new texttool;
    drawingtool *changeFillPattern = new changeFillPatterntool;
    drawingtool *changeFillColor = new changeFillColorortool;
    drawingtool *changeDrawColor = new changeDrawColorortool;

    // Draw a main menu bar across the top of the screen. Each of the
    // words in the menu bar will act as a user-interface object that
    // can be selected. Some of the words, such as Text and Line, will
    // cause pull-down menus to appear if the user clicks on them.
    h = textheight("H") + 2;
    if (getmaxcolor() == 1) setfillstyle(SOLID_FILL, 0);
    else setfillstyle(SOLID_FILL, EGA_BLUE);
    bar3d(0, 0, maxx, h, 0, 0);
    outtextxy(2, 2, "Text Line Clear Quit");
    space = textwidth(" ");
    offset = 2;
}

```

```

tools.add_tool(new interact('t',offset,0,offset+textwidth("Text"),
    textheight("Text"),changeFont));
textwindowx = offset;
textwindowy = textheight("Text") + BORDER;
offset += textwidth("Text") + space;
tools.add_tool(new interact('l',offset,0,
    offset+textwidth("Line"),textheight("Line"),changelinestyle));
linewindowx = offset;
linewindowy = textheight("Line") + BORDER;
offset += textwidth("Line") + space;
tools.add_tool(new interact('c',offset,0,
    offset+textwidth("Clear"),textheight("Clear"),clearworkarea));
offset += textwidth("Clear") + space;
tools.add_tool(new interact('q',offset,0,
    offset+textwidth("Quit"),textheight("Quit"),quit));
// Now draw the icons on the left-hand side of the screen
read_icon(0,h,"spray.icn");
tools.add_tool(new interact('s',0,h,ICONW*2,h+ICONH*2,spraycan));
read_icon(ICONW*2,h,"letter.icn");
tools.add_tool(new interact('c',ICONW*2,h,ICONW*4,h+ICONH*2,text));
read_icon(0,h+ICONH*2,"line.icn");
tools.add_tool(new interact('d',0,h+ICONH*2,ICONW*2,h+2*ICONH*2,linet1));
read_icon(ICONW*2,h+ICONH*2,"pencil.icn");
tools.add_tool(new interact('p',ICONW*2,h+ICONH*2,ICONW*4,
    h+2*ICONH*2,pencil));
read_icon(0,h+2*ICONH*2,"square.icn");
tools.add_tool(new interact('s',0,h+2*ICONH*2,ICONW*2,h+3*ICONH*2,rect));
read_icon(ICONW*2,h+2*ICONH*2,"fillbox.icn");
tools.add_tool(new interact('r',ICONW*2,h+2*ICONH*2,ICONW*4,
    h+3*ICONH*2,fillrect));
read_icon(0,h+3*ICONH*2,"circle.icn");
tools.add_tool(new interact('c',0,h+3*ICONH*2,ICONW*2,
    h+4*ICONH*2,circlet1));
read_icon(ICONW*2,h+3*ICONH*2,"ellipse.icn");
tools.add_tool(new interact('g',ICONW*2,h+3*ICONH*2,ICONW*4,
    h+4*ICONH*2,ellipset1));
read_icon(0,h+4*ICONH*2,"polygon.icn");
tools.add_tool(new interact('p',0,h+4*ICONH*2,ICONW*2,h+5*ICONH*2,poly));
read_icon(ICONW*2,h+4*ICONH*2,"fillpoly.icn");
tools.add_tool(new interact('x',ICONW*2,h+4*ICONH*2,ICONW*4,
    h+5*ICONH*2,fillpoly));
read_icon(0,h+5*ICONH*2,"eraser.icn");
tools.add_tool(new interact('e',0,h+5*ICONH*2,ICONW*2,
    h+6*ICONH*2,eraser));
read_icon(ICONW*2,h+5*ICONH*2,"arc.icn");
tools.add_tool(new interact('a',ICONW*2,h+5*ICONH*2,ICONW*4,
    h+6*ICONH*2,arct1));

// Draw a backdrop below the icons
setfillstyle(SOLID_FILL,EGA_BLUE);
bar3d(0,h+6*ICONH*2,ICONW*4,maxy,0,0);

// Create the fill pattern box. This will appear on the lower portion
// of the screen.
fill_width = (maxx-1-ICONW*4+2) / (NUM_FILLS_WIDE+1);
globalfillcolor = getmaxcolor(); // Start fill color at maxcolor
globaldrawcolor = globalfillcolor;
filltype = 0;
for (i=0, x=ICONW*4+2+fill_width; i<NUM_FILLS_WIDE; i++,x+=fill_width) {
    setfillstyle(filltype,globalfillcolor);
    bar3d(x,maxy-FILL_HEIGHT*2,x+fill_width,maxy-FILL_HEIGHT,0,0);
    filltype++;
}

```

```

}
rectangle(ICONW*4+2,maxy-FILL_HEIGHT*2,maxx,maxy);
tools.add_tool(new userinteract('w',ICONW*4+2+fill_width,
    maxy-FILL_HEIGHT*2,maxx,maxy-FILL_HEIGHT,changeFillpattern));
globalfillstyle = SOLID_FILL;

maxcolors = getmaxcolor();
color_width = (maxx-1-ICONW*4+2-fill_width) / (maxcolors + 1);
fillcolor = 0;
for (i=0, x=ICONW*4+2+fill_width; i<=maxcolors; i++, x+=color_width){
    setfillstyle(SOLID_FILL,fillcolor);
    bar3d(x,maxy-FILL_HEIGHT,x+color_width,maxy,0,0);
    fillcolor++;
}
tools.add_tool(new userinteract('z',ICONW*4+2+fill_width,
    maxy-FILL_HEIGHT,maxx,maxy,changeFillColor));
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
setfillstyle(globalfillstyle,globalfillcolor);
bar3d(ICONW*4+4,maxy-FILL_HEIGHT*2+2,ICONW*4+fill_width,maxy-2,0,0);
tools.add_tool(new userinteract('y',ICONW*4+2,
    maxy-FILL_HEIGHT*2,ICONW*4+fill_width+2,maxy,changeDrawcolor));
setlinestyle(SOLID_LINE,0,NORM_WIDTH);

// Draw main draw area window
wl = ICONW*4+2+1;  wt = h+2+1;

wr = maxx-1;
wb = maxy-FILL_HEIGHT*2-2-1;
rectangle(wl-1,wt-1,wr+1,wb+1);
globalfillstyle = SOLID_FILL;
globallinestyle = SOLID_LINE;
globaltextstyle = DEFAULT_FONT;
}

```

第十三章 CAD 程序

在第十二章，我们开发了一个画画程序，它提供绘制图表和模型的图形环境。在本章，我们将扩充这个画画程序，以便它变成可用的CAD程序包。从外观上看，这两个图形程序之间的差别似乎很小；但是，在内部这两者在若干重要的方法上是不同的。贯穿本章，我们在讨论怎样编写CAD程序和它如何工作时探讨这些差异。最后，通过描述若干增强措施而结束本章。这些增强可能正是你想添加到CAD程序以建立自己定制的版本。

13.1 画画与画图

图13.1示出CAD程序的环境。虽然该程序看起来像我们在前一章开发的画画程序，但这两者有重大的差异。例如，画画程序被设计为提供画图的环境——十分像美术中用水彩画画，一旦画出一个景色，就很难改变它。而另一方面，CAD程序被设计成它所显示的每一景色都是从单独的元件构造而成的。这些元件可以移动、旋转、删除等等。

CAD程序的灵活性归功于如下事实。它保持在绘图窗口中所有图形对象的一个表。这个表包含与每一图表有关的各种属性，如它们的大小和位置。此外，尺寸信息按世界坐标保存，以便它能更容易和现实世界对象的规格说明相匹配。

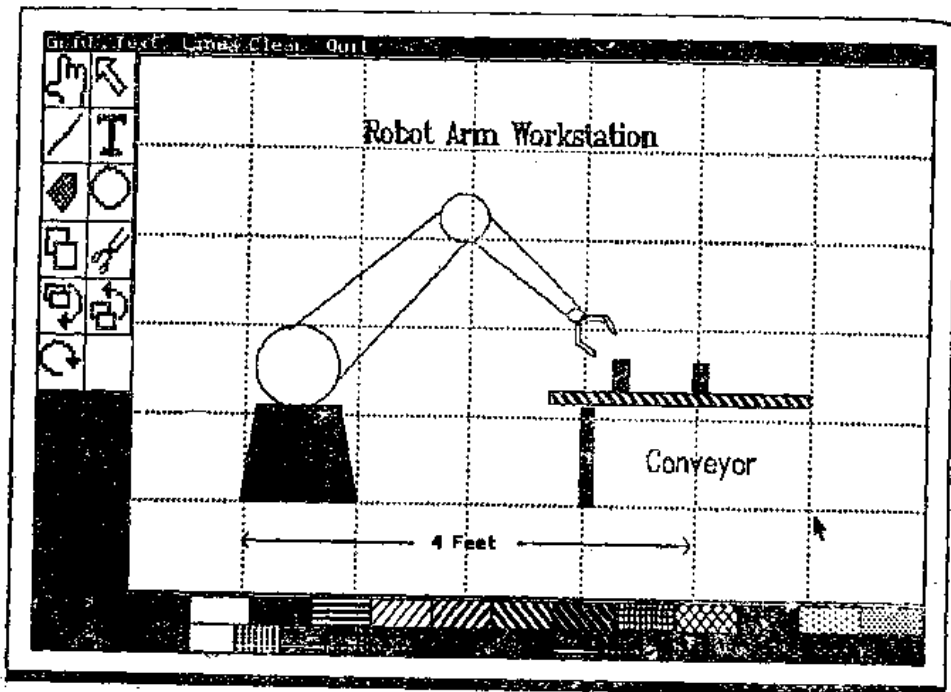


图 13.1 CAD程序的环境

绘图窗口中的每一对象都保留在一个对象表中，它类似于我们在画画程序（第十二章）中处理图符和菜单选择项所用的类程。用这个新的对象表我们可以很容易存取、处理

和移动绘图窗口中的每一个图形对象。

包含在CAD程序中的主要增强可概括如下：

- (1) 所有对象按真实世界坐标存放；
- (2) 对象可被选择和移动；
- (3) 图形保留在一个对象表中；
- (4) 支持旋转和复制功能；
- (5) 画多维化的线；
- (6) 显示调准的网格；
- (7) 可以改变对象的绘制顺序。

为了支持这些功能中的每一项。CAD程序将变得比较庞大。幸好，我们可以利用C++的继承性和建立在我们用于画画程序的代码的基础上。但是，我们将需要开发七个新的文件。它们支持针对CAD程序的专用功能。本章讨论的这些文件列在表13.1，每一个都带有简短的描述。

表 13.1 专用于CAD程序的源文件

文 件 名	列 表	描 述
cad.cpp	13.1	包含CAD程序的main()函数
gobjlist.h	13.2	包含用于gobjlist.cpp的类型定义和常数
gobjlist.cpp	13.3	保持绘图窗口中的图形表
gobject.h	13.4	包含用于gobject.cpp的类型定义和常数
gobject.cpp	13.5	提供用以表示绘图窗口中的图形的对象
caddraw.h	13.6	caddraw.cpp的类型定义和常数
caddraw.cpp	13.7	CAD程序使用的杂项函数

文件cad.cpp非常类似于在第十二章为画画程序开发的主要源文件，所以它不是本章的重点。我们将着重讨论gobjlist.cpp、gobject.cpp和caddraw.cpp。

由于CAD程序规模较大，所以仅支持画线、多边形、圆、正文的函数。这些函数提供足够的能力构成有用的CAD程序包。但是，你可以很容易修改CAD程序以包括若干出现在画画程序中的其它绘图函数。

此外，该程序当前设计成在高分辨率的EGA和VGA模式之一中工作得很好。如果你使用其它的模式，则可能需要调整图符或绘图窗口的大小。

13.1.1 设置屏幕

在图13.1中，你将会注意到CAD程序的环境稍微不同于画画程序。有一些新的图符和添加到绘图窗口的网格图案。这些变化中的每一种都可以在cad.cpp的函数setup_screen()中发现。它用于建立CAD程序的环境。

让我们从考查新的图符图案和它们相应的函数开始。如果你再观察cad.cpp，将会注意到有几个对setup_screen()中add_tool()的新的或修改了的调用。但是，我们使用的是同一技术，以保持所有绘图工具的表。因此，setup_screen函数的主要部分关系到把CAD程序的各种绘图工具添加到能用的交互工具表中。

对函数`setup_screen()`的一个不太显眼的改变是把绘图窗口设置为反映真实世界坐标7单位宽6单位高（你可以假设这些单位为任何类型——英寸、英尺、千米、等等）。这是用下面`setup_screen()`中的两个语句完成的：

```
set_window(0.0,0.0,7.0,6.0);
set_viewport(wl,wt,wr,wb);
```

两个函数都已在第六章介绍过。为帮助记忆，再重复一遍，它们用于建立真实世界坐标和屏幕坐标之间的关系。函数`set_window()`定义真实世界坐标界在 $(0.0, 0.0)$ 和 $(7.0, 6.0)$ 之间。`set_viewport()`函数（请回忆一下，这个函数不同于`setviewport()` BGI函数）定义`matrix.cpp`程序包的绘图窗口的界。

添加到CAD程序的一个新特色是在绘图窗口中显示网格。它用`caddraw.cpp`包含的`drawgrid()`函数绘出，你将会发现是在初始化窗口和观察端口之后调用这个在`setup_screen()`中的函数。设计网格是为了帮助你对齐在绘图窗口中的对象，它由一系列水平和垂直的虚线组成。以真实世界坐标的1个单位分隔开。在线型被临时改变为点线之后，用以下两个for循环在`draw_grid()`中画网格：

```
for (j=1; j<7; j++) {
    WORLDtoPC(0.0,0.0+j,x,y);
    line(wl,y,wr,y);
}
for (i=1; i<6; i++) {
    WORLDtoPC(0.0+i,0.0,x,y);
    line(x,wt,x,wb);
}
```

顶部的for循环顺序展示跨过屏幕的水平线，而第二个循环则画出垂直线。请注意，下标变量被添加到真实世界坐标。于是，用`WORLDtoPC()`把真实世界坐标转化为屏幕坐标，然后再绘出。这确保所有线条根据世界坐标的尺寸分隔。你可能需要调整传送给`set_window()`并用于for循环的值，以便你的网格图案包含正方形外表的单元。

一个称为`gridon`的全程标志（在`cad.cpp`中定义）控制是否显示网格。如果`gridon`被置为值1，则显示网格；如果它被置为0，则无需显示网格。为触发网格的显示，你可以选择程序主条形菜单中的字`Grid`。

现在，我们已考查在环境中明显的主要变化，让我们转向CAD程序的独特的内核部分。

13.1.2 对象表

第十二章的画画程序和CAD程序之间的主要差异之一是，CAD程序保持一个在其绘图窗口中的所有对象的表。这个表十分像在`interact.cpp`中出现的面向对象表，我们曾用那个面向对象表综合图符、上弹菜单，以及在画画程序中的屏幕命令和它们的操作。

如同在`interact.cpp`中的情况一样，对象表是作为类程实现的，它包含一个指向这些对象的指针数组。（我们稍后将讨论这些对象。）屏幕上的每一图形都有一个和它相联的对象。在表中管理此图形表的新类程`gobjlist`在`gobjlist.h`中定义为：

```
const int NUMOBJECTS = 20;           // Maximum size of list
class gobjlist {
```



```

public:
    drawingtool *gobjects[NUMGOBJECTS]; // List of graphics figures
    int nextobj; // Number of graphics objects
    int currentobj; // Currently selected object
    void addobj(drawingtool *obj);
    void deleteobj(void);
    void deleteall(void);
    void fliptoback(void);
    void fliptofront(void);
    void duplicate(void);
    void rotate(void);
    void select(void);
    void move(void);
    void mark(int obj);
    void displayall(void);
};

```

表13.2提供gobjlist中每一成员函数的简短描述。如这个定义所示，gobjlist对象包含一个指向数组gobjects中drawingtools的指针的表。这些是我们在第十一章开发并用在画画程序中的同样的drawingtools吗？不完全是，gobjects中的对象实际上是我们用在画画程序中的绘图工具的专用版本。即，这些对象是从用于画画程序的drawingtool对象派生出来的，在未来的一节，我们将对它们谈论更多，但首先让我们回到gobjlist的细节。你将会注意到gobjects被说明为足够大以保存NUMGOBJECTS对象，它已定义为20。如果你想一次在屏幕上有大于20的图形，你可能需要增加这个数目。caddraw.cpp中说明了一个全程gobjlist对象figlist，并贯穿CAD程序用于访问gobjlist类程的功能。

表 13.2 gobjlist中的成员函数

成员函数	描 述
addobj ()	把一个对象增加到gobjects数组中
deleteobj ()	从gobjects数组删除当前对象并从屏幕消去它的图像 从gobjects数组删除所有对象并清屏
fliptoback ()	把当前对象移到屏幕后面
fliptofront ()	把当前对象移到屏幕前面
duplicate ()	复制当前对象
rotate ()	把当前对象旋转45度
select ()	允许用户指定哪一对象为当前对象
move ()	允许用户交互地移动当前对象
mark ()	显示围绕当前对象的有界方框
displayall ()	在屏幕上显示gobjects数组中的所有对象

通过调用gobjlist成员函数addobj()把图形添加到gobjects数组。这只不过是 把传送到函数的drawingtool对象附加到gobjects数组（如果有地方的话）。此外，addobj()增加指定gobjects数组中下一空闲单元的nextobj变量，同时变量currentobj（它指向当前正被处理的数组中的对象）被置为刚才添加的对象。

gobjlist中的其余成员函数也用于处理gobjects表中的图形。这些函数管理显示哪一些对象和按什么次序显示。例如，有一些函数可用于删除、复制、移动和选择屏幕上的图形

对象。当我们讨论CAD程序支持的各种操作时，将考查这些附加的成员函数。

13.2 画各种对象

CAD程序支持有限的绘图函数集，包括画线例程，多边形函数、正文例程和画圆函数。这些例程的每一个都使用在画画程序中所用的同样代码来画图。请回想一下，这些绘图工具是作为派生自基础类程drawingtool的类程来实现的。实际上，这些工具不是精确同一的。我们已通过派生新的一组工具来扩充它们，以便把它们画的图形添加到上节讨论的gobjlist对象中。例如，我们已从第十二章使用的linetool类程派生出lineobj类程。图13.2示出我们将使用的新的层次。这些新工具的代码包含在列表13.4和列表13.5的gobject.h和gobject.cpp中。

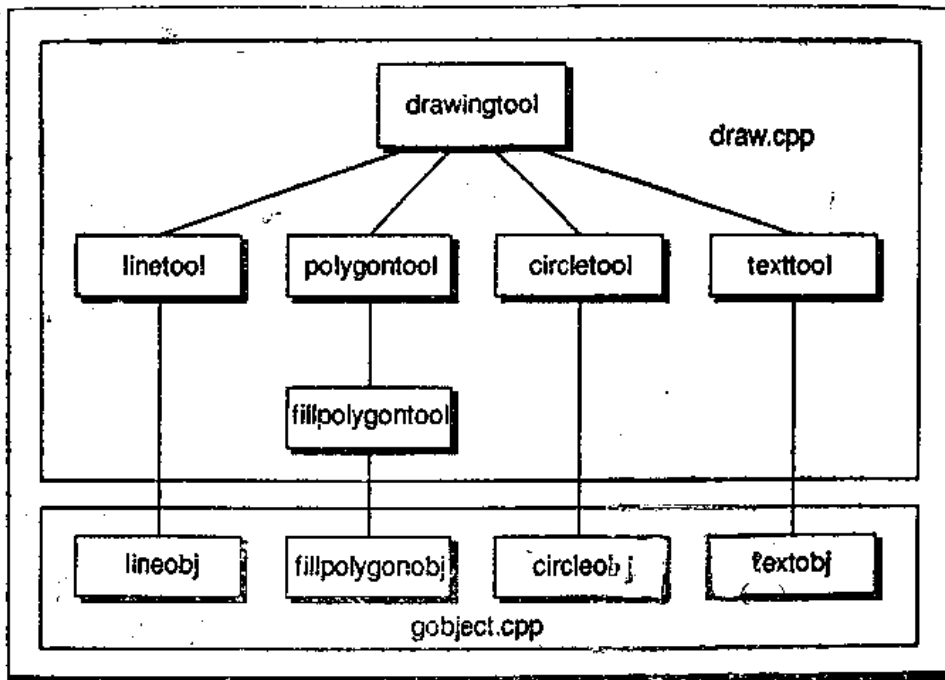


图13.2 从draw.cpp中的工具包派生的新绘图工具

那么，这些新的类程包含些什么和怎样使用呢？下面是你将从它们当中找到的新特征的简列表：

- 以世界坐标保存它们的图形的坐标；
- 提供一些函数以选择、移动、复制、转换和旋转一特定的图形；
- 重设继承的finishdrawing()函数，使得在绘制图形之后，把它们添加到对象表。

要注意一个要点是，所有的图形对象都是按世界坐标保存的，这样做是为了在程序中易于处理对象。如果你考查在派生的drawingtool类程中的各种成员函数，那么，将看到在屏幕坐标和世界坐标之间所作的转换。在下一节，我们将学习如何画这些对象和把它们添加到图形对象表中。

13.2.1 画线

新的画线类程lineobj(gobject.cpp) 派生自linetool类程(draw.cpp)。我们正重设

原来的绘图工具，以便能添加在前一节描绘的新功能。例如，重设finishdrawing() 成员函数要做的第一件事情是调用它的继承函数：

```
linetool::finishdrawing();
```

这将确保正确处理所有图形的最后润色。往下，建立lineobj、nl和用于画图形的line-
tool对象的内容，都被拷贝到lineobj对象中。最后，把lineobj对象添加到gobjects数组。
这些步骤由下面的语句完成：

```
lineobj *nl = new lineobj;           // Create a new line object
getbounds(left,top,right,bottom);
PCtoWORLD(x1,y1,xw1,yw1);           // Save endpoints in world
PCtoWORLD(x2,y2,xw2,yw2);           // coordinates
nl->copy(this);                       // Copy all settings to new object
draw_arrows();                       // Draw arrows on new line
figlist.addobj(nl);                 // Add the line to the figure list
```

值得注意的是，lineobj数据是从linetool对象拷贝的，通过PCtoWORLD() (出自matrix.cpp) 把它的端点转化为世界坐标，并按世界坐标保存。

在该语句序列中调用的getbounds() 成员函数计算对象的PC屏幕世界，并在撤压了鼠标的按钮之一时用它来确定鼠标选择哪一个对象。请注意，lineobj::getbounds() 中的边界值偏移3个像素。这样做是为了当线条为水平或垂直时，它似乎稍微大些，并因而较容易选择。

在调用addobj() 之前是下面的语句：

```
draw_arrows();
```

它也是lineobj中的一个函数，如果用户选择了这个类型，则这个成员函数在线段上附加一个箭头。三个新的线类型已添加到line菜单以支持这些箭头。图13.3示出了线菜单任选项的新的下拉菜单（它包括这些新的箭头线型）。由于设计了两种线型使得箭头在线的前导或拖尾边缘，故维持线的哪一边带箭头是重要的。依赖于所选箭头的类型，left_arrow或right_arrow或两者被置为值1。这些都用在成员函数draw_arrows() 以选择哪一种箭头被显示。箭头的大小由ARROWSIZE指定，并在gobject.cpp中置为3像素宽和高。

处理箭头的困难部分是箭头必须旋转，以便它面向所附线段的同一角度，这些调整是在draw_arrows() 中进行的。

如果left_arrow为非0，则函数draw_arrows() 在线段的前导边缘 (X1, Y1) 画箭头，如果right_arrow为非0，则在线段的拖尾边缘 (X2, Y2) 画箭头。箭头本身是由两个直接的线段组成的。它们把线的端点之一联结起来。这些点包含在数组arrowhead中。调整箭头的图形以便它的角度与线段相匹配，首先是把它的中心平移到原点，旋转它，然后再把它平移回到线段的尖端。这个过程由下列的一些行执行：

```
PCtranslatepoly(3,arrowhead,-x1+w1,-y1+wt);
PCrotatepoly(3,arrowhead,angle);
PCtranslatepoly(3,arrowhead,x1-w1,y1-wt);
```

最后，通过下面的语句画箭头：

```
drawpoly(3,arrowhead);
```

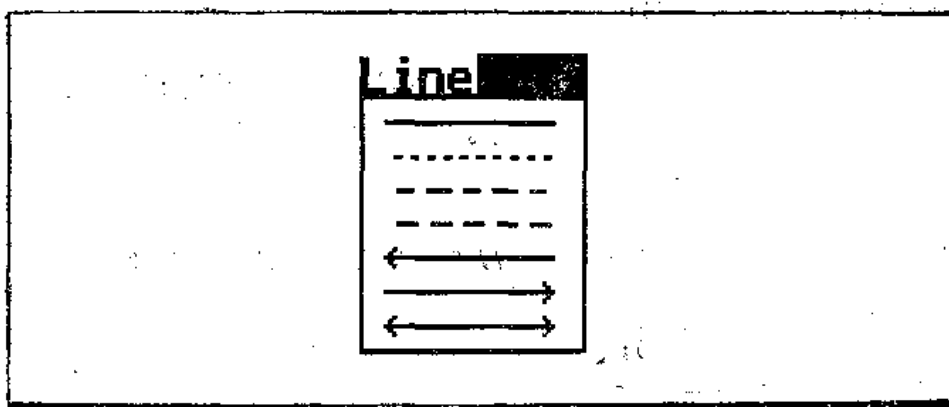


图 13.3 线型下拉菜单包含箭头符号

13.2.2 画多边形和圆

类程`fillpolygonobj`和`circleobj`显示和管理多边形和圆的图形。这两个类程是由`draw.cpp`中等价的交互式绘图工具派生而来的，你将会发现它们类似于上一节讨论的`lineobj`类程，因此我们将只关注它们的差异。

你将会注意到，这两个类程的`getbounds()`成员函数是大不相同的。原因是每一对象各不相同地存放它的图形信息。例如，我们在`lineobj`中看到端点明显给出边框的尺寸。但是，对于圆或多边形，这个任务稍微有点复杂。

在圆的情况下，通过它的中心加或减圆的半径可计算出边框。例如，语句

```
r = centerx + absradiusx
```

计算出圆的右边缘。请回想一下，当画图时`circletool`保存`absradiusx`。计算圆的顶和底稍微不同。在这些情况下，我们需要考虑屏幕的长宽比。因此，为计算圆的顶，使用下面的语句：

```
t = centery - absradiusx * aspectratio;
```

其中，`aspectratio`是在`main()`中计算的全程变量，并且基于BGI函数`getaspectratio()`返回的值。

确定多边形的边界是独特的，必须检测多边形的每一顶点，看看它是否落在边框上。但是，这个代码非常简单。

13.2.3 作为图形对象的正文

为了支持作为图形对象的正文，对用在画画程序中的`texttool`类程作了少量的修改。这些修改出现在`gobject.cpp`包含的派生类程`textobj`中。一种修改是，重设`finishdrawing()`，以便能调用`addobj()`把你键入的正文字符串添加到对象表中。

请回想一下，当调用`texttool`把正文对象添加到对象表时，ASCII正文是保存在对象的

string字段中的。正文的起点屏幕坐标在leftx和lefty中指定，它们被转化为世界坐标并保存在textobj类程的xw和yw成员变量中。

display() 成员函数显示存放在textobj中的正文。由于正文字符串可能包含新行字符，故这稍微复杂些。因此，我们必须自己检查是否出现新行字符并通过调整当前位置跳到下一行。这个过程由display() 中的for循环处理：

```
for (i=0; i<len; i++)
    if (string[i] == '\n') {        // Newline character
        lefty += textheight("S");  // Skip to next line
        moveto(leftx-wl, lefty-wt); // Adjust current position
    }
    else {
        c[0] = string[i];           // Write out the next character
        outtext(c);
    }
```

13.2.4 显示图形对象

由于CAD程序被设计或能允许你修改绘图窗口中的对象，所以必须有在修改对象后能重画屏幕的函数。这就是gobjlist类程中的display()和displayall()成员函数的目的。

例如，在上一节我们看到textobj::display() 如何被用于显示存放在string中的一行或多行的正文。有一些可与lineobj、fillpolygonobj和circleobj类程相比较的成员函数。此外，gobjlist包含例程displayall() 以画出表中所有的对象。核心动作发生在for循环，它顺序通过gobjects数组，直到抵达nextobj下标，并调用该对象的display()函数。因此，每一图形对象都自身画出：

```
for (obj=0; obj<nextobj; obj++)
    gobjects[obj]->display();
```

13.2.5 删除图形对象

现在让我们考查一下怎样从绘图窗口抹除一个对象。从屏幕删除一个对象包含两个步骤：首先，从内部的gobjects数组删除该对象，然后，清除整个屏幕并显示gobjects中所有其余的对象。

gobjlist类程中的删除例程称为deleteobj()。它除去由全程变量currentobj索引的对象，通过释放正被删除的drawingtool对象的存储，函数开始工作：

```
delete(gobjects[currentobj]);
```

往下，重排数组，以便把当前的对象移到gobjects数组的末尾，而且，在它之上的所有对象都往下移一个下标位置：

```
for (i=currentobj; i<nextobj-1; i++)
    gobjects[i] = gobjects[i+1];
```

再往下，指向gobjects下一可用对象位置的下标指针nextobj减1。此外，检测一下最后的对象是否是正被删除的那个。如果是，则currentobj下标也减小。最后，清除屏幕，通

过调用以下函数重画对象表中的所有对象:

```
cleardrawingarea();  
displayall();
```

函数cleardrawingarea()通过使用BGI clearviewport()函数清除绘图窗口。此外,如果标志gridon为1时还画出网格。第二个函数displayall()重画屏幕上的所有对象。当然,除了刚才删去的对象。请注意在删除操作以后,currentobj通常变成数组gobjects中的下一个对象。

13.3 复制函数

CAD程序还包括用于复制对象表中对象的函数duplicate()。这个函数包含在gobjlist类程中,并把由currentobj索引的对象拷贝到变量nextobj指向的gobjects中下一空闲位置。

实际上,现在每个drawingtool对象都有它自己的复制例程(dup()),可调用它来获得对象自身的复制品并返回一个指向新对象的指针。当然,单纯复制有关对象的所有信息将引起在原来的对象上面显示新的图形。为避免这点,dup()成员函数在x和y方向用0,1(世界坐标)平移它的图形坐标。在duplicate()的末尾,通过调用

```
gobjects[currentobj]->display()
```

画出当前对象的平移的副本(现在变成当前对象)。

13.4 旋转命令

CAD程序提供的另一个函数允许你围绕对象的中心旋转一个对象。这是由gobjlist成员函数rotate()和新的drawingtool对象中的rotate()成员函数完成的。再说一遍,gobjlist的rotate()成员函数实际调用drawingtool的rotate()函数执行旋转。

这些drawingtool rotate()函数使用一系列对matrix.cpp中例程的调用(参见第六章),围绕它们的中心点以45度增量顺时针旋转当前对象。

如果该对象为多边形或线(在CAD程序中不能旋转圆和正文),则把对象的中心平移到原点,旋转该对象,然后把它平移回它的原来位置,一旦旋转了该对象,就可通过调用getbounds()确定它的新屏幕边界。最后,清除屏幕并用displayall()重画所有对象。

13.5 修改绘图次序

每次调用displayall()时,它画出gobjects数组中的每一个对象。从0下标开始,往下工作直到正好在下标nextobj之前的数组最后单元。由于总是遵循这个绘图次序,所以总是先画表开头的对象。结果,如果两个对象重叠,则总是把靠近gobjects数组开头的对象画在另一个的下面。

CAD程序允许你通过交换gobjects数组中对象的位置,修改显示对象的次序。例如,

为移动在所有其它对象后面的一个对象，只要简单地把该对象移到表的头部。这就是gobjlist成员函数fliptoback()所要做的。类似地，或员函数fliptofront()把一个对象从它的当前位置移到表的末尾，以便它将最后显示，并因此在所有其它对象的顶部。由于两者类似，我们仅考查fliptofront()。

函数从检测对象表是否为空开始，这是当nextobj为0的情况，如果是这样，则该函数简单地不带任何动作返回。

```
if (nextobj == 0) return;
```

往下，使用一个临时指针temp保存currentobj指出的对象。这是将被移到gobjects数组尾部的对象。然后，通过以下循环把在currentobj之上的每一个对象下移数组中的一个位置：

```
for (i=currentobj; i<nextobj-1; i++)  
    gobjects[i] = gobjects[i+1];
```

最后，把保存在temp中的对象拷贝到对象表的尾部，并更新currentobj下标，使得它仍然指向同一对象。即使它现在是在已表的尾部。

```
gobjects[i] = temp;  
currentobj = i;
```

一旦修改了对象表，则通过清除屏幕更新绘图窗口，并通过执行以下两个语句显示gobjects中所有的对象：

```
cleardrawingarea();  
displayall();
```

这将使得当前对象在绘图窗口中所有其它图形对象的顶部显示。

现在我们已经学习了如何复制、旋转和翻转对象表中的一个对象；让我们考查怎样才能屏幕上选出一个对象并移动它。

13.6 选择和移动一个对象

CAD程序允许你通过把鼠标移向对象并撒按它来选择绘图窗口中的对象。(当这样做时，在图形附近将出现一个虚线的矩形，而currentobj将被置为相应对象的下标位置。结果，这个对象将变成任一随后使用的函数，如duplicate()和rotate()所处理的对象。如果有另一对象与当前选择的对象共享屏幕的同一区域，则可通过在同一区再撒按鼠标而交替地选择。类似地，如果在同一区域还有其它对象，你可以通过多次撒按鼠标顺序选择它们。

gobjlist成员函数select()执行早些时候描述的动作。不打算逐行分析它的代码，让我们大致观察一下它。此代码的主要目的是把currentobj下标修改到当撒按了左鼠标按钮时鼠标指向的对象。select()成员函数示出哪一个对象是当前的对象。这通过调用mark()来完成，它围绕当前对象画一个矩形框，该矩形使用了针对当前对象的drawingtool类程的left、top、right、和bottom成分来确定它的位置。请注意，setwritemode()

被置为XOR_PUT, 以便该矩形可“异或”在屏幕上, 并因此很容易移动。当select()退出时, 通过最后调用mark()擦去该矩形。

通过掀按手形图符移动鼠标到该对象, 然后掀压左鼠标按钮, 并把该对象拖曳到它的新位置, 可移动一选出的对象。当释放鼠标按钮时, 重画整个屏幕, 对象出现在它的新位置上。

这个操作是由gobjlist类程包含的move()成员函数执行的。通过在变量movex和movey中记录鼠标已经移动的量程来移动一个对象。把这个数量变换成世界坐标, 并用于平移该对象。一旦完成平移, 则擦去并完全重画屏幕以反映出这种改变。

13.7 访问gobjlist中的成员函数

迄今为止, 我们已描述若干添加到CAD程序中的新操作。它们的大多数包含在gobjlist类程中。但又怎样从环境访问它的成员函数(像deleteobj())呢? 请记住, 当选择图符或某些其它屏幕对象时, 所有其它绘图操作都被连接到活动的drawingtool中。但是, 由于我们的deleteobj()函数是gobjlist类程的成员, 故不能简单地把它添加到drawingtool对象表中——它有错误的基本类型。结果, 我们引用gobjlist中适当动作的caddraw.h和caddraw.cpp里定义了一系列的drawingtool对象。这些类程用作在gobjlist提供的环境和函数使用的drawingtools之间的桥梁。例如, deleteobj()成员函数是在派生自drawingtool的类程deleteobjtool中调用的。因此, 对deleteobjtool的说明被写成:

```
class deleteobjtool : public drawingtool {
public: virtual void draw(void);
}
```

draw()成员函数被重设, 因为这是在选择drawingtool时调用的函数。在这种情况下, 我们需要draw()简单地调用全程表figlist中的deleteobj()成员函数:

```
void deleteobjtool::draw(void) { figlist.deleteobj(); }
```

然后, 这个新的“绘图工具”被添加到cad.cpp的setup_screen()函数的工具表中, 并在掀按了它的图符时生效。类似的技术用于访问objlist类程中的其它函数, 如flip_toback()、rotate()和move()。

13.8 扩充CAD程序

有许多方法增强本章提供的CAD程序。你可能想添加一些函数, 以便把绘图窗口中的对象保存到文件中, 提供放大和缩小正被显示的对象的功能, 或增加组合对象的能力。让我们简略地考查一下前两种设想。

如果你希望能保存CAD程序中的对象, 则所需做的一切, 就是把gobjects数组保存到文件中。你可以用文件的第一行保存gobjects数组中对象的数目和当前对象的下标。文件的其余部分可用于保存gobjects数组中的每一个drawingtool结构, 其中每一对象都应有一个标志和它相联, 该标志标识对象的数据类型。从文件把这个信息读到gobjects数组同样

是很简单的事情，这将允许你初始化整个屏幕或读入选出的对象。

你可能想要添加的另一个特色是放大和缩小在绘图窗口的对象的功能。基本思想是改变屏幕上显示的世界坐标的窗口大小。例如，为放大绘图窗口中的对象，你可以置窗口为(0.0,0.0)和(3.0,3.0)之间，如下所示：

```
void zoomin(void)
{
    set_window(0.0,0.0,3.0,3.0);
    set_viewport(wl,wt,wr,wb);
    cleardrawingarea();
    displayall();
}
```

13.9 编译CAD程序

CAD程序包含许多贯穿本书的源文件。为编译和连接CAD程序，表 13.3 中所列的是你必须包含在工程文件中的源文件。请记住，像画画程序那样，CAD应用程序是比较大的，所以你必须用大存储模型来编译它。

表 13.3 为编译CAD程序必须连接的文件

文 件	标题文件	章
gtext.cpp	gtext.h	4
matrix.cpp	matrix.h	6
mouse.cpp	mouse.h	8
kbmouse.cpp	kbmouse.h	8
gpopup.cpp	gpopup.h	10
draw.spp	draw.h	11
userool.cpp	usertool.h	12
interact.cpp	interact.h	12
caddraw.cpp	caddraw.h	13
cad.cpp	cad.h	13

• 列表13.1 cad.cpp

```
// cad.cpp -- A CAD program. This program exploits many of the BGI
// functions to build a simple yet powerful CAD program.
// Several of the tools built up earlier in the book are put to
// use in this program. The program will run in many of the
// graphics adapters supported by the BGI and supports many of
// their modes. However, it is currently configured so that it
// will appear best if run on an EGA or VGA display. The program
// supports either the mouse or the keyboard automatically.
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdarg.h>
#include <alloc.h>
```

```

#include <math.h>
#include "kbdmouse.h"
#include "usertool.h"
#include "interact.h"
#include "gtext.h"           // Text in graphics mode tools
#include "gpopup.h"          // Graphics pop-up window package
#include "matrix.h"
#include "caddraw.h"
#include "draw.h"
#include "gobject.h"
#include "cad.h"

void setup_screen(void);      // Only function in cad.cpp
double aspectratio;          // Aspect ratio of the screen
int gridon;                  // Value of 1 if grid is to be displayed
extern int currentobj;        // Current graphics object in list
extern int nextobj;           // Next available slot in list of objects
int left_arrow;              // If value of 1, adds left or right arrow
int right_arrow;             // heads to the lines that are drawn

main()
{
    int c, errcode, gmode, gdriver = DETECT;
    int xasp, yasp;

    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    if ((errcode=graphresult()) != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errcode));
        exit(1);              // Return with error code
    }
    getaspectratio(&xasp, &yasp);
    aspectratio = (double)xasp / (double)yasp;
    maxx = getmaxx(); maxy = getmaxy();
    setup_screen();
    mouse_init();
    do {
        c = mouse_waitforinput(LEFT_BUTTON);
        tools.anytoprocess(c);
    } while (1);
}

// Set up the environment for the CAD program
void setup_screen(void)
{
    int h, i, x, fill_width, offset, space;
    int filltype, color_width, maxcolors, fillcolor;
    // Declare a set of drawing tools. Note that late binding
    // is used to get the power of polymorphism.
    drawingtool *togglegrid = new togglegridtool;
    drawingtool *deleteallobj = new deleteallobjtool;
    drawingtool *changeFont = new changeFonttool;
    drawingtool *changelinestyle = new cadchangelinestyletool;
    drawingtool *quit = new quittool;
    drawingtool *linetl = new lineobj;
    drawingtool *fillpoly = new fillpolygonobj;
    drawingtool *circletl = new circleobj;
    drawingtool *text = new textobj;
    drawingtool *changeFillpattern = new changeFillpatternool;
    drawingtool *changeFillColor = new changeFillColorool;
    drawingtool *changeDrawcolor = new changeDrawcolorool;
    drawingtool *moveobj = new moveobjtool;

```

```

drawingtool *selectobj = new selectobjtool;
drawingtool *duplicateobj = new duplicateobjtool;
drawingtool *deleteobj = new deleteobjtool;
drawingtool *fliptofront = new fliptofronttool;
drawingtool *fliptoback = new fliptobacktool;
drawingtool *rotateobj = new rotateobjtool;

// Draw a main menu bar across the top of the screen. Each of the
// words in the menu bar will act as a user-interface object that
// can be selected.
h = textheight("H") + 2;
if (getmaxcolor() == 1) setfillstyle(SOLID_FILL,0);
else setfillstyle(SOLID_FILL,EGA_BLUE);
bar3d(0,0,maxx,h,0,0);
outtextxy(2,2,"Grid Text Line Clear Quit");
space = textwidth(" ");
offset = 2;

tools.add_tool(new interact('g',offset,0,offset+textwidth("Grid"),
    textheight("Grid"),togglegrid));
offset += textwidth("Grid") + space;
tools.add_tool(new interact('t',offset,0,offset+textwidth("Text"),
    textheight("Text"),changeFont));
textwindowx = offset;
textwindowy = textheight("Text") + BORDER;
tools.add_tool(new interact('l',offset,0,offset+textwidth("Line"),
    textheight("Line"),changelinestyle));
linewindowx = offset;
linewindowy = textheight("Line") + BORDER;
offset += textwidth("Line") + space;
tools.add_tool(new interact('c',offset,0,offset+textwidth("Clear"),
    textheight("Clear"),deleteallobj));
offset += textwidth("Clear") + space;
tools.add_tool(new interact('q',offset,0,offset+textwidth("Quit"),
    textheight("Quit"),quit));

// Now draw the icons on the left-hand side of the screen
read_icon(0,h,"hand.icn");
tools.add_tool(new interact('s',0,h,ICONW*2,h+ICONH*2,moveobj));
read_icon(ICONW*2,h,"pointer.icn");
tools.add_tool(new interact('c',ICONW*2,h,ICONW*4,h+ICONH*2,selectobj));
read_icon(0,h+ICONH*2,"line.icn");
tools.add_tool(new interact('d',0,h+ICONH*2,ICONW*2,h+2*ICONH*2,linet));
read_icon(ICONW*2,h+ICONH*2,"letter.icn");
tools.add_tool(new interact('p',ICONW*2,h+ICONH*2,ICONW*4,
    h+2*ICONH*2,text));
read_icon(0,h+2*ICONH*2,"fillpoly.icn");
tools.add_tool(new interact('s',0,h+2*ICONH*2,ICONW*2,
    h+3*ICONH*2,fillpoly));
read_icon(ICONW*2,h+2*ICONH*2,"circle.icn");
tools.add_tool(new interact('r',ICONW*2,h+2*ICONH*2,ICONW*4,
    h+3*ICONH*2,circlet));
read_icon(0,h+3*ICONH*2,"duplicat.icn");
tools.add_tool(new interact('p',0,h+3*ICONH*2,ICONW*2,
    h+4*ICONH*2,duplicateobj));
read_icon(ICONW*2,h+3*ICONH*2,"scissor.icn");
tools.add_tool(new interact('x',ICONW*2,h+3*ICONH*2,ICONW*4,
    h+4*ICONH*2,deleteobj));
read_icon(0,h+4*ICONH*2,"flipfrnt.icn");
tools.add_tool(new interact('e',0,h+4*ICONH*2,ICONW*2,
    h+5*ICONH*2,fliptofront));

```

```

read_icon(ICONW*2,h+4*ICONH*2,"flipback.icn");
tools.add_tool(new interact('x',ICONW*2,h+4*ICONH*2,ICONW*4,
    h+5*ICONH*2,fliptoback));
read_icon(0,h+5*ICONH*2,"rotate.icn");
tools.add_tool(new interact('e',0,h+5*ICONH*2,ICONW*2,
    h+6*ICONH*2,rotateobj));
// Draw a backdrop below the icons
setfillstyle(SOLID_FILL,EGA_BLUE);
bar3d(0,h+6*ICONH*2,ICONW*4,maxy,0,0);

// Create the fill pattern box. This will appear on the lower
// portion of the screen.
fill_width = (maxx-1-ICONW*4+2) / (NUM_FILLS_WIDE+1);
globalfillcolor = getmaxcolor(); // Start fill color at maxcolor
globaldrawcolor = globalfillcolor;
filltype = 0;

for (i=0, x=ICONW*4+2+fill_width; i<NUM_FILLS_WIDE; i++,
    x += fill_width) {
    setfillstyle(filltype,globalfillcolor);
    bar3d(x,maxy-FILL_HEIGHT*2,x+fill_width,maxy-FILL_HEIGHT,0,0);
    filltype++;
}
rectangle(ICONW*4+2,maxy-FILL_HEIGHT*2,maxx,maxy);
tools.add_tool(new userinteract('w',ICONW*4+2+fill_width,
    maxy-FILL_HEIGHT*2,maxx,maxy-FILL_HEIGHT,changeFillpattern));
globalfillstyle = SOLID_FILL;
maxcolors = getmaxcolor();
color_width = (maxx-1-ICONW*4+2-fill_width) / (maxcolors + 1);
fillcolor = 0;
for (i=0, x=ICONW*4+2+fill_width; i<=maxcolors; i++,x+=color_width) {
    setfillstyle(SOLID_FILL,fillcolor);
    bar3d(x,maxy-FILL_HEIGHT,x+color_width,maxy,0,0);
    fillcolor++;
}
tools.add_tool(new userinteract('z',ICONW*4+2+fill_width,
    maxy-FILL_HEIGHT,maxx,maxy,changeFillColor));
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
setfillstyle(globalfillstyle,globalfillcolor);
bar3d(ICONW*4+2+2,maxy-FILL_HEIGHT*2+2,
    ICONW*4+2+fill_width-2,maxy-2,0,0);
tools.add_tool(new userinteract('y',ICONW*4+2+2,maxy-FILL_HEIGHT*2+2,
    ICONW*4+fill_width-2,maxy,changeDrawcolor));
setlinestyle(SOLID_LINE,0,NORM_WIDTH);

// Draw main draw-area window
wl = ICONW*4+2+1; wt = h+2+1;
wr = maxx-1; wb = maxy-FILL_HEIGHT*2-2-1;
rectangle(wl-1,wt-1,wr+1,wb+1);

// Set WORLD coordinate to screen coordinate relationship. If
// you change these values, change the corresponding ones in
// draw_grid() located in caddraw.cpp. Pick numbers that make
// the grid look square. These work well on a VGA.
set_window(0.0,7.0,0.0,6.0);
set_viewport(wl,wr,wt,wb);

globalfillstyle = SOLID_FILL;
globallinestyle = SOLID_LINE;
globaltextstyle = DEFAULT_FONT;
gridon = 1;
draw_grid();

```

```

left_arrow = 0;
right_arrow = 0;

// Set the font here to what you want to use for the dimension labels.
// On the CGA in medium-resolution mode, small font is good.
settextstyle(globaltextstyle,HORIZ_DIR,1);
}

```

• 列表13.2 gobjlist.h

```

// gobjlist.h -- Defines a class that maintains a list of all the
// graphics figure objects that have been drawn
#ifndef GOBJLISTH
#define GOBJLISTH
#include "gobject.h"

// Maximum number of figures maintained
const int NUMGOBJECTS = 20;

// This class is used to maintain the list of figures currently
// displayed in the drawing window
class gobjlist {
public:
    drawingtool *gobjects[NUMGOBJECTS]; // List of graphics figures
    int nextobj; // Number of graphics objects
    int currentobj; // Currently selected object
    void addobj(drawingtool *obj);
    void deleteobj(void);
    void deleteall(void);
    void fliptoback(void);
    void fliptofront(void);
    void duplicate(void);
    void rotate(void);
    void select(void);
    void move(void);
    void mark(int obj);
    void displayall(void);
};
#endif

```

• 列表13.3 gobjlist.cpp

```

// gobjlist.cpp -- Member functions for the gobjlist class. These
// routines handle the management of the gobject graphics objects
// maintained in the CAD program. These are the objects that
// represent individual graphics figures on the screen.
#include <graphics.h>
#include "gobjlist.h"
#include "matrix.h"
#include "kbdmouse.h"
#include "caddraw.h"

// Adds an object to the list of graphic objects in the drawing window
void gobjlist::addobj(drawingtool *obj)
{
    if (nextobj >= NUMGOBJECTS) return; // List is full. Do nothing.
    currentobj = nextobj;
    gobjects[nextobj++] = obj;
}

```

```

// Delete an object from the graphics object list. Make the next
// object in the gobjects array the currentobj. Update the screen
// after deleting object from the list by erasing it and then
// redrawing all remaining objects.
void gobjlist::deleteobj(void)
{
    int i;

    if (nextobj == 0) return;          // Empty list. No objects.
    delete(gobjects[currentobj]);
    for (i=currentobj; i<nextobj-1; i++)
        gobjects[i] = gobjects[i+1];
    nextobj--;
    if (currentobj >= nextobj && currentobj)
        currentobj--;                // Deleted last, but not first
    cleardrawingarea();
    displayall();
}

// Delete all objects in the object list.
void gobjlist::deleteall(void)
{
    int i;

    for (i=0; i<nextobj; i++)
        delete(gobjects[i]);
    nextobj = 0;
    currentobj = 0;
    cleardrawingarea();
}

// Move the current object to the back by copying it to the
// head of the object list. After the operation the same object
// is the current object.
void gobjlist::fliptoback(void)
{
    int i;
    drawingtool *temp;

    if (nextobj == 0) return;
    temp = gobjects[currentobj];
    for (i=currentobj; i>0; i--)
        gobjects[i] = gobjects[i-1];
    gobjects[0] = temp;
    currentobj = 0;
    cleardrawingarea();
    displayall();
}

// Move the object to the front by putting it at the end of the
// object list and redrawing all of the objects. After the operation
// the same object is the current object.
void gobjlist::fliptofront(void)
{
    int i;
    drawingtool *temp;

    if (nextobj == 0) return;
    temp = gobjects[currentobj];
    for (i=currentobj; i<nextobj-1; i++)
        gobjects[i] = gobjects[i+1];

```

```

    gobjects[i] = temp;
    currentobj = i;
    cleardrawingarea();
    displayall();
}

// Duplicate currentobj. Make a copy of the current object in the
// drawing window. New object appears offset from the original
// by translating it (.1, .1). Make sure the object is saved in
// WORLD coordinates.
void gobjlist::duplicate(void)
{
    if (nextobj == 0) return;          // No objects in list
    if (nextobj >= NUMGOBJECTS) return; // Object list is full
    gobjects[nextobj] = gobjects[currentobj]->dup();
    currentobj = nextobj;              // New object becomes current object
    nextobj++;
    setviewport(wl,wt,wr,wb,1);
    gobjects[currentobj]->display();
    setviewport(0,0,getmaxx(),getmaxy(),1);
}

// Rotate the current object about its center point. The object is
// rotated in 45-degree increments. Note that only polygon and line
// rotations are supported.
void gobjlist::rotate(void)
{
    if (nextobj == 0) return;          // No objects in list
    gobjects[currentobj]->rotate(45);
    cleardrawingarea();
    displayall();                      // Display updated objects
}

// Interactively move the current object on the screen
void gobjlist::move(void)
{
    int x, y, oldx, oldy, movex, movey;
    int l, t, r, b;
    double originalx, originaly, translatex, translatey;

    if (nextobj == 0) return;          // No objects in list
    setviewport(wl,wt,wr,wb,1);
    mark(currentobj);
    while (1) {
        while (!mouse.buttonpressed(LEFT_BUTTON)) ;
        mouse.getcoords(x,y);
        if (x <= wl || y <= wt || y > wb || x > wr) {
            mark(currentobj); // Unmark object
            setviewport(0,0,getmaxx(),getmaxy(),1);
            return;
        }
        l = gobjects[currentobj]->left;
        t = gobjects[currentobj]->top;
        r = gobjects[currentobj]->right;
        b = gobjects[currentobj]->bottom;
        if (mouse.inbox(l,t,r,b,x,y)) {
            oldx = x;    oldy = y;
            while (!mouse.buttonreleased(LEFT_BUTTON)) {
                mouse.getcoords(x,y);
                movex = x - oldx;    movey = y - oldy;
                if (movex != 0 || movey != 0) {
                    mark(currentobj);

```

```

        gobjects[currentobj]->left += movex;
        gobjects[currentobj]->top += movey;
        gobjects[currentobj]->right += movex;
        gobjects[currentobj]->bottom += movey;
        mark(currentobj);
        oldx = x; oldy = y;
    }
}
PctoWORLD(1,t,originalx,originaly);
PctoWORLD(gobjects[currentobj]->left,
           gobjects[currentobj]->top,translatex,translatey);
translatex -= originalx; translatey -= originaly;
gobjects[currentobj]->translate(translatex,translatey);
cleardrawingarea();
displayall();
mark(currentobj);
}
else
    while (!mouse.buttonreleased(LEFT_BUTTON)) ;
}
}

// Select an object by encompassing it with a rectangle. An object
// can be selected by pressing the left mouse button while over
// the object. To select another object in the same region, click
// the mouse button again: Upon exiting, the currently marked
// object becomes the currentobj which is used in all succeeding
// operations.
void gobjlist::select(void)
{
    int x, y, testobj, lastobj;

    if (nextobj == 0) return; // No objects in list
    setviewport(wl,wt,wr,wb,1);

    testobj = currentobj + 1;
    if (testobj >= nextobj) testobj = 0;
    mark(currentobj); // Mark object
    lastobj = currentobj;
    while (1) {
        while (!mouse.buttonpressed(LEFT_BUTTON)) ;
        mouse.getcoords(x,y);
        if (x <= wl || y <= wt || y > wb || x > wr) {
            mark(currentobj); // Unmark object
            setviewport(0,0,getmaxx(),getmaxy(),1);
            return;
        }
        while (!mouse.buttonreleased(LEFT_BUTTON)) ;
        mouse.getcoords(x,y);
        do {
            if (mouse.inbox(gobjects[testobj]->left,
                           gobjects[testobj]->top,
                           gobjects[testobj]->right,
                           gobjects[testobj]->bottom,x,y)) {
                mark(lastobj); // Unmark object
                mark(testobj); // Mark object
                lastobj = testobj;
                currentobj = testobj;
                testobj++;
                if (testobj >= nextobj) testobj = 0;
                break;
            }
        }
    }
}

```



```

        testobj++;
        if (testobj >= nextobj) testobj = 0;
    } while (testobj != currentobj);
}

// Display all of the graphics objects in the drawing window
void gobjlist::displayall(void)
{
    int obj;
    viewporttype v;

    getviewsettings(&v);           // Save viewport settings
    setviewport(wl,wt,wr,wb,1);     // Use drawing window
    setwritemode(COPY_PUT);
    for (obj=0; obj<nextobj; obj++)
        gobjects[obj]->display();
    setviewport(v.left,v.top,v.right,v.bottom,v.clip); // Restore viewport
}

// Highlight an object by drawing a dashed rectangle around its border
void gobjlist::mark(int obj)
{
    linesettingstype saveline;
    int savecolor;
    drawingtool *t = gobjects[obj];

    savecolor = getcolor();
    getlinesettings(&saveline);
    setwritemode(XOR_PUT);
    setcolor(getmaxcolor());
    setlinestyle(DASHED_LINE,0,NORM_WIDTH);
    mouse.hide();
    rectangle(t->left-wl,t->top-wt,t->right-wl,t->bottom-wt);
    mouse.show();
    setlinestyle(saveline.linestyle,saveline.upattern,saveline.thickness);
    setcolor(savecolor);
    setwritemode(COPY_PUT);
}

```

• 列表13.4 gobject.h

```

// gobject.h -- Defines graphics objects such as lines and circles
// that are drawn in the CAD program. These objects, derived
// from the drawingtool classes in draw.h, supply the functions to
// draw, move, and rotate the graphics objects.
#ifndef GOBJECTH
#define GOBJECTH
#include "draw.h"

// Text object. The location of the text in real-world coordinates,
// its style and what it displays are saved. Note that rotation of
// text is not supported.
class textobj : public texttool {
public:
    double xw, yw;           // Top left of text in real-world coordinates
    virtual void draw(void);
    virtual void copy(textobj *f);
    virtual void translate(double transx, double transy);
    virtual void getbounds(int &l, int &t, int &r, int &b);
    virtual void display(void);
}

```

```

    virtual void finishdrawing(void);
    virtual drawingtool *dup(void);
};

// Line object
class lineobj : public linetool {
public:
    double xw1, yw1;    // Top left of text in real-world coordinates
    double xw2, yw2;    // Other endpoints of line
    int left_arrow, right_arrow; // Nonzero if the line has an arrowhead
    static const int ARROWSIZE; // Size of arrow heads
    virtual void draw(void);
    virtual void translate(double transx, double transy);
    virtual void getbounds(int &l, int &t, int &r, int &b);
    virtual void copy(lineobj *f);
    virtual void display(void);
    virtual void finishdrawing(void);

    virtual void draw_arrows(void);
    virtual drawingtool *dup(void);
    virtual void rotate(double angle);
};

// Circle object
class circleobj : public circletool {
public:
    double centerwx, centerwy; // Center of circle in world coordinates
    virtual void draw(void);
    virtual void getbounds(int &l, int &t, int &r, int &b);
    virtual void copy(circleobj *f);
    virtual void translate(double transx, double transy);
    virtual void display(void);
    virtual void finishdrawing(void);
    virtual drawingtool *dup(void);
};

class fillpolygonobj : public fillpolygontool {
public:
    double pointsw[MAXPOLYSIZE];
    ~fillpolygonobj(void);
    virtual void draw(void);
    void getbounds(int &l, int &t, int &r, int &b);
    virtual void copy(fillpolygonobj *f);
    virtual void display(void);
    virtual void translate(double transx, double transy);
    virtual void finishdrawing(void);
    virtual drawingtool *dup(void);
    virtual void rotate(double angle);
};
#endif

```

• 列表13.5 gobject.cpp

```

// gobject.cpp -- Defines the functions used to support graphics figures
// as objects in the CAD program. The classes here are derived from the
// tools developed in draw.cpp in Chapter 10. These objects are maintained
// in a list by the class gobjlist which is defined in gobjlist.cpp.
#include <graphics.h>
#include <alloc.h>
#include <math.h>
#include <string.h>

```

```

#include "gobject.h"
#include "gobjlist.h"
#include "kbmouse.h"
#include "matrix.h"
#include "gtext.h"
#include "caddraw.h"

const double PI = 3.1415926;

// Override draw() so it won't be called if the object list is full
void textobj::draw(void)
{
    if (figlist.nextobj < NUMGOBJECTS-1) texttool::draw();
}

// Create a new text object by making a copy of the current text
// object and then translating it in world coordinates by (0.1,0.1)
// Returns a pointer to the new object.
drawingtool *textobj::dup(void)
{
    textobj *nt = new textobj;           // Create new text object
    nt->copy(this);                       // Copy "this" to new object
    nt->xw += 0.1;  nt->yw += 0.1;         // Translate new text
    // Calculate screen position of duplicated text
    WORLDtoPC(nt->xw,nt->yw,nt->leftx,nt->lefty);
    nt->getbounds(nt->left,nt->top,nt->right,nt->bottom);
    return nt;                           // Return pointer to new
}

// Display the text in the text object. It accounts for newline
// characters included in the string.
void textobj::display(void)
{
    int i;
    char c[] = "c";                     // String to hold a single character

    setcolor(drawcolor);                 // Use the stored drawing color
    WORLDtoPC(xw,yw,leftx,lefty);        // Using the world coordinates saved
    moveto(leftx-wl,lefty-wt);           // in the object, write the text out
    settextstyle(textstyle,HORIZ_DIR,1);
    mouse.hide();
    for (i=0; i<len; i++)
    {
        if (string[i] == '\n') {         // Newline character
            lefty += textheight("S");    // Skip to next line
            moveto(leftx-wl,lefty-wt);   // Adjust current position
        }
        else {
            c[0] = string[i];             // Write out the next character
            outtext(c);
        }
    }
    mouse.show();
}

// Get bounding box of text object in screen coordinates. Since text
// can stretch across multiple lines, check for the newline character
// to get the real text block's height, and to find its true width.
void textobj::getbounds(int &l, int &t, int &r, int &b)
{
    int i=0, linew=0, w=0, h=0, startofline=0, ptr;
    char buff[80];

    l = leftx;  t = lefty;

```

```

    for (i=0; string[i]; i++) {
        linewidth++; // Increment line width
        if (string[i] == '\n') { // Newline encountered
            h++; // Increment height count
            if (linewidth > w) { // Check whether this line is
                w = linewidth; ptr = startofline; // the longest line so far
            }
            linewidth = 0; // Reset the line width counter
            startofline = i+1; // Remember where next line starts
        }
    }
    if (linewidth > w) // Check whether last line is
        w = linewidth; ptr = startofline; // the longest line

    strncpy(buff, &string[ptr], w); // Copy the longest line to a string
    buff[w] = 0; // so it can be used to get the
    r = leftx + textwidth(buff); // width of the widest line
    b = lefty + (textheight("C") + 2) * (h + 1);
}

// Move the top-left location of the text by the amount (transx, transy)
void textobj::translate(double transx, double transy)
{
    xw += transx;    yw += transy;
}

// Copy the text object f to "this"
void textobj::copy(textobj *f)
{
    left = f->left;
    top = f->top;
    right = f->right;
    bottom = f->bottom;
    leftx = f->leftx;
    lefty = f->lefty;
    xw = f->xw;
    yw = f->yw;
    drawcolor = f->drawcolor;
    textstyle = f->textstyle;
    len = f->len;
    strcpy(string, f->string);
}

// After entering a text string, add it to the figure object list
void textobj::finishdrawing(void)
{
    texttool::finishdrawing();
    if (len == 0) return; // Don't add object if string is empty
    textobj *nl = new textobj; // Create a new object to add
    getbounds(left, top, right, bottom);
    PCtoWORLD(leftx, lefty, xw, yw); // Save things in world coordinates
    nl->copy(this); // Copy settings to new object
    figlist.addobj(nl); // Add "this" to the list of objects
}

// Override draw() so it won't be called if the object list is full
void lineobj::draw(void)
{
    if (figlist.nextobj < NUMGOBJECTS-1) linetool::draw();
}

// After drawing a line, append it to the figure list

```

```

void lineobj::finishdrawing(void)
{
    linetool::finishdrawing();    // Call inherited routine
    left_arrow = ::left_arrow;    // Remember the settings of
    right_arrow = ::right_arrow;  // the arrows
    lineobj *nl = new lineobj;    // Create a new line object
    getbounds(left,top,right,bottom);
    PCtoWORLD(x1,y1,xw1,yw1);    // Save endpoints in world
    PCtoWORLD(x2,y2,xw2,yw2);    // coordinates
    nl->copy(this);               // Copy all settings to new object
    draw_arrows();               // Draw arrows on new line
    figlist.addobj(nl);          // Add the line to the figure list
}

// Create a new line object by making a copy of the current line
// object. Translate the new line in world coordinates by (0.1,0.1)
// and then set its bounding box size.
drawingtool *lineobj::dup(void)
{
    lineobj *nl = new lineobj;    // Create new line
    nl->copy(this);               // Copy current object to new object
    nl->xw1 += 0.1; nl->yw1 += 0.1; // Translate new line
    nl->xw2 += 0.1; nl->yw2 += 0.1;
    WORLDtoPC(nl->xw1,nl->yw1,nl->x1,nl->y1); // Calculate new line's
    WORLDtoPC(nl->xw2,nl->yw2,nl->x2,nl->y2); // screen position
    nl->getbounds(nl->left,nl->top,nl->right,nl->bottom);
    return nl;
}

// Copy line1 to line2
void lineobj::copy(lineobj *f)
{
    left = f->left;
    top = f->top;
    right = f->right;
    bottom = f->bottom;
    x1 = f->x1;
    y1 = f->y1;
    x2 = f->x2;
    y2 = f->y2;
    xw1 = f->xw1;
    yw1 = f->yw1;
    xw2 = f->xw2;
    yw2 = f->yw2;
    drawcolor = f->drawcolor;
    linestyle = f->linestyle;

    left_arrow = f->left_arrow;
    right_arrow = f->right_arrow;
}

// Display the line in the line object
void lineobj::display(void)
{
    int xpc1, ypc1, xpc2, ypc2;

    setcolor(drawcolor);
    setlinestyle(linestyle,0,NORM_WIDTH);
    WORLDtoPC(xw1,yw1,xpc1,ypc1);
    WORLDtoPC(xw2,yw2,xpc2,ypc2);
    mouse.hide();
    line(xpc1-w1,ypc1-w1,xpc2-w1,ypc2-w1);
}

```

```

    mouse.show();
    draw_arrows();
}

// Translate the line in world coordinates by the (transx,transy)
void lineobj::translate(double transx, double transy)
{
    xw1 += transx;
    yw1 += transy;
    xw2 += transx;
    yw2 += transy;
}

// Get the bounding box for a line object. Note that the bounding box
// is expanded by 3 pixels to make it easier to select a line.
void lineobj::getbounds(int &l, int &t, int &r, int &b)
{
    if (x1 <= x2) {
        l = x1 - 3;
        r = x2 + 3;
    }
    else {
        l = x2 - 3;
        r = x1 + 3;
    }
    if (y1 <= y2) {
        t = y1 - 3;
        b = y2 + 3;
    }
    else {
        t = y2 - 3;
        b = y1 + 3;
    }
}

// Rotate a line
void lineobj::rotate(double angle)
{
    int cx, cy, PCpoints[4], numpoints = 2;
    double points[4], cwx, cwy;

    cx = (left + right) / 2;           // Get the center of the line
    cy = (top + bottom) / 2;
    PCtoWORLD(cx,cy,cwx,cwy);           // Translate to world coordinates
    points[0] = xw1; points[1] = yw1;    // Store line endpoints in an
    points[2] = xw2; points[3] = yw2;    // array
    WORLDtranslatepoly(numpoints,points,-cwx,-cwy);
    WORLDrotatepoly(numpoints,points,angle);
    WORLDtranslatepoly(numpoints,points,cwx,cwy);
    WORLDpolytoPCpoly(numpoints,points,PCpoints);
    xw1 = points[0]; yw1 = points[1];
    xw2 = points[2]; yw2 = points[3];
    x1 = PCpoints[0]; y1 = PCpoints[1];
    x2 = PCpoints[2]; y2 = PCpoints[3];
    getbounds(left,top,right,bottom);    // Calculate new bounding box
}

// Draw arrows to the line if needed. The mouse is already hidden
// when this function is called. The arrows are displayed as an
// open-ended polygon.
const int lineobj::ARROWSIZE = 4;      // Size of an arrow

```

```

void lineobj::draw_arrows(void)
{
    int arrowhead[3*2];          // An arrow is made from two lines offset
                                // 45 degrees on each side of the line
    double angle;                // Calculated slope of line
    int x1, x2, y1, y2;

    WORLDtoPC(xw1,yw1,x1,y1);
    WORLDtoPC(xw2,yw2,x2,y2);
    if (left_arrow) {           // Draw an arrow at the start of the line
        arrowhead[0] = x1 + ARROWSIZE - w1;
        arrowhead[1] = y1 - ARROWSIZE - wt;
        arrowhead[2] = x1 - w1;
        arrowhead[3] = y1 - wt;
        arrowhead[4] = x1 + ARROWSIZE - w1;
        arrowhead[5] = y1 + ARROWSIZE - wt;
        angle = atan2((double)(y2-y1), (double)(x2-x1)) * 180.0 / PI;
        PCtranslatepoly(3,arrowhead,-x1+w1,-y1+wt);
        PCrotatepoly(3,arrowhead,angle);
        PCtranslatepoly(3,arrowhead,x1-w1,y1-wt);
        mouse.hide();
        drawpoly(3,arrowhead);    // Draw the arrow
        mouse.show();
    }
    if (right_arrow) {          // Draw an arrow at the end of the line
        arrowhead[0] = x2 - ARROWSIZE - w1;
        arrowhead[1] = y2 - ARROWSIZE - wt;
        arrowhead[2] = x2 - w1;
        arrowhead[3] = y2 - wt;
        arrowhead[4] = x2 - ARROWSIZE - w1;
        arrowhead[5] = y2 + ARROWSIZE - wt;
        angle = atan2((double)(y2-y1), (double)(x2-x1)) * 180.0 / PI;
        PCtranslatepoly(3,arrowhead,-x2+w1,-y2+wt);
        PCrotatepoly(3,arrowhead,angle);
        PCtranslatepoly(3,arrowhead,x2-w1,y2-wt);
        mouse.hide();
        drawpoly(3,arrowhead);    // Draw the arrow
        mouse.show();
    }
}

// Override draw() so it won't be called if the object list is full
void circleobj::draw(void)
{
    if (figlist.nextobj < NUMGOBJECTS-1) circletool::draw();
}

// Get the bounding box of a circle object
void circleobj::getbounds(int &l, int &t, int &r, int &b)
{
    WORLDtoPC(centerwx,centerwy,centerx,centery);
    l = centerx - absradiusx;      // centerx - radius
    t = centery - absradiusx * aspectratio; // centery - radius
    r = centerx + absradiusx;      // centerx + radius
    b = centery + absradiusx * aspectratio; // centery + radius
}

// Make a copy of a circle object. The new circle is the same as
// "this" circle, except it is translated by (0.1,0.1). Returns
// pointer to new circle object.
drawingtool *circleobj::dup(void)
{

```

```

circleobj *nc = new circleobj; // Create a new circle object
nc->copy(this);                // Copy the circle settings
nc->centerwx += 0.1;            // Translate the center (0.1,0.1)
nc->centerwy += 0.1;
// Calculate position and bounds of new circle
WORLDtoPC(nc->centerwx,nc->centerwy,nc->centerx,nc->centery);
nc->getbounds(nc->left,nc->top,nc->right,nc->bottom);
return nc;                     // Return pointer to new circle

// Copy the settings in the f object to "this" circle
void circleobj::copy(circleobj *f)
{
    centerwx = f->centerwx;
    centerwy = f->centerwy;
    centerx = f->centerx;
    centery = f->centery;
    left = f->left;
    top = f->top;
    right = f->right;
    bottom = f->bottom;
    absradiusx = f->absradiusx;
    drawcolor = f->drawcolor;
}

// Display the circle object
void circleobj::display(void)
{
    setcolor(drawcolor);
    WORLDtoPC(centerwx,centerwy,centerx,centery);
    mouse.hide();
    circle(centerx-wl,centery-wt,absradiusx); // Draw the circle
    mouse.show();
}

// After drawing a circle, append an object for it to the list of
// graphics figures on the screen
void circleobj::finishdrawing(void)
{
    circletool::finishdrawing(); // Call the inherited routine
    circleobj *nc = new circleobj; // Create a new circle object
    PCtoWORLD(centerx,centery,centerwx,centerwy); // Use world coordinates
    getbounds(left,top,right,bottom); // Save bounding box of circle
    nc->copy(this); // Copy settings to new object
    figlist.addobj(nc); // Add object to figure list
}

// Translate a circle in world coordinates
void circleobj::translate(double transx, double transy)
{
    centerwx += transx;
    centerwy += transy;
}

// Override draw() so it won't be called if the object list is full
void fillpolygonobj::draw(void)
{
    if (figlist.nextobj < NUMGOBJECTS-1)
        fillpolygontool::draw();
}

```



```

// Duplicate the current polygon object by creating an identical
// polygon except that it is translated by (0.1,0.1). A pointer
// to the new object is returned.
drawingtool *fillpolygonobj::dup(void)
{
    fillpolygonobj *np = new fillpolygonobj; // Create new polygon object
    np->copy(this); // Copy settings to new object
    WORLDtranslatepoly(np->numpts/2,np->pointsw,0.1,0.1);
    WORLDpolytoPCpoly(np->numpts/2,np->pointsw,np->poly);
    np->getbounds(np->left,np->top,np->right,np->bottom);
    return np; // Return pointer to new object
}

// After drawing a polygon, add an object for it to the list of
// figures. Copy the polygon's settings that were used to draw
// the figure to the new object.
void fillpolygonobj::finishdrawing(void)
{
    fillpolygontool::finishdrawing();
    fillpolygonobj *np = new fillpolygonobj; // Create a polygon object
    PCpolytoWORLDpoly(numpts/2,poly,pointsw); // Use world coordinates
    getbounds(left,top,right,bottom); // Get bounds of polygon
    np->copy(this); // Copy settings
    figlist.addobj(np); // Add polygon object
}

// Copy the polygon f to "this" polygon
void fillpolygonobj::copy(fillpolygonobj *f)
{
    int i;
    left = f->left;
    top = f->top;
    right = f->right;
    bottom = f->bottom;
    numpts = f->numpts;
    for (i=0; i<numpts; i++) // Copy PC points
        poly[i] = f->poly[i];
    copyWORLDpoly(f->numpts/2,f->pointsw,pointsw);
    drawcolor = f->drawcolor;
    linestyle = f->linestyle;
    fillcolor = f->fillcolor;
    fillstyle = f->fillstyle;
}

// Display a polygon
void fillpolygonobj::display(void)
{
    setcolor(drawcolor);
    setlinestyle(linestyle,0,NORM_WIDTH);
    setfillstyle(fillstyle,fillcolor);
    WORLDpolytoPCpoly(numpts/2,pointsw,poly);
    mouse.hide();
    fillpoly(numpts/2,poly);
    mouse.show();
}

// Get bounding box of polygon object in screen coordinates
void fillpolygonobj::getbounds(int &l, int &t, int &r, int &b)
{
    int i;
    l = getmaxx(); t = getmaxy();
    r = 0; b = 0;
}

```

```

WORLDpolytoPCpoly(numpts/2,pointsw,poly);
for (i=0; i<numpts; i+=2) {
    if (poly[i] < l) l = poly[i];
    if (poly[i] > r) r = poly[i];

    if (poly[i+1] < t) t = poly[i+1];
    if (poly[i+1] > b) b = poly[i+1];
}
l += wl; r += wl; t += wt; b += wt;
}

// Translate a polygon in world coordinates
void fillpolygonobj::translate(double transx, double transy)
{
    WORLDtranslatepoly(numpts/2,pointsw,transx,transy);
}

// Rotate a polygon by the amount in "angle"
void fillpolygonobj::rotate(double angle)
{
    int i;
    int cx = (left + right) / 2 - wl;
    int cy = (top + bottom) / 2 - wt;
    WORLDpolytoPCpoly(numpts/2,pointsw,poly);
    PCtranslatepoly(numpts/2,poly,-cx,-cy);
    PCrotatepoly(numpts/2,poly,angle);
    PCtranslatepoly(numpts/2,poly,cx,cy);
    for (i=0; i<numpts; i+=2)
        PCToWORLD(poly[i],poly[i+1],pointsw[i],pointsw[i+1]);
    getbounds(left,top,right,bottom);
}

```

• 列列13.6 caddraw.h

```

// caddraw.h -- Header file for caddraw.cpp.
#ifndef CADDRAWH
#define CADDRAWH
#include "gpopup.h"
#include "gobjlist.h"

extern gobjlist figlist; // The list of figures on the screen

// Supplies routine to pop up a menu of line styles
const int LINE_HEIGHT = 8; // Vertical spacing between lines in menu
const int LINE_WIDTH = 40; // Width of a line in the pop-up menu
const int MAX_LINESTYLES = 7; // Number of line styles in the menu
class cadchangelinestyletool : public drawingtool {
public:
    virtual void draw(void);
};

// The following classes are provided so that we can provide
// access to various functions in the environment through the
// same drawing tool object list we created in Chapter 11 for
// the paint program

class togglegridtool : public drawingtool {
public:
    virtual void draw(void);
};

```

```

class deleteallobjtool : public drawingtool {
public:
    virtual void draw(void);
};

class moveobjtool : public drawingtool {
    virtual void draw(void);
};

class selectobjtool : public drawingtool {
    virtual void draw(void);
};

class duplicateobjtool : public drawingtool {
    virtual void draw(void);
};

class deleteobjtool : public drawingtool {
    virtual void draw(void);
};

class fliptofronttool : public drawingtool {
    virtual void draw(void);
};

class fliptobacktool : public drawingtool {
    virtual void draw(void);
};

class rotateobjtool : public drawingtool {
    virtual void draw(void);
};

class cadlinetool : public linetool {
    virtual void draw(void);
};

class cleardrawingareatool : public drawingtool {
public:
    virtual void draw(void);
};

void draw_grid(void);
void cleardrawingarea(void);

extern int linewindowx;
extern int linewindowy;
extern int left_arrow;

extern int right_arrow;
extern int gridon;
extern gwindows w;
extern double aspectratio;
#endif

```

• 列表13.7 caddraw.CPP

```

// caddraw.cpp -- Drawing and object support routines for the CAD program
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>

```

```

#include <stdarg.h>
#include <alloc.h>
#include <math.h>
#include "kbdmouse.h"
#include "draw.h"
#include "cad.h"
#include "gpopup.h"
#include "caddraw.h"          // Utilities specific to the CAD program
#include "matrix.h"
#include "gobject.h"
#include "gobjlist.h"

gobjlist figlist;           // Globally define a list of screen figures

// Clear the drawing window
void cleardrawingarea(void)
{
    viewporttype v;

    getviewsettings(&v);
    setviewport(wl,wt,wr,wb,1); // Set viewport to drawing window
    mouse.hide();
    clearviewport();           // Clear the drawing window
    // Set viewport back to full screen
    setviewport(v.left,v.top,v.right,v.bottom,1);
    if (gridon)                // Draw a grid in the drawing
        draw_grid();          // window if the gridon flag equals 1
    mouse.show();
}

// Used to interface the environment to the move() function
void moveobjtool::draw(void) { figlist.move(); }

// Used to interface the environment to the select() function
void selectobjtool::draw(void) { figlist.select(); }

// Used to interface the environment to the duplicate() function
void duplicateobjtool::draw(void) { figlist.duplicate(); }

// Used to interface the environment to the delete() function
void deleteobjtool::draw(void) { figlist.deleteobj(); }

// Used to interface the environment to the fliptofront() function
void fliptofronttool::draw(void) { figlist.fliptofront(); }

// Used to interface the environment to the fliptoback() function
void fliptobacktool::draw(void) { figlist.fliptoback(); }

// Used to interface the environment to the rotate() function
void rotateobjtool::draw(void) { figlist.rotate(); }

// Used to interface the environment to the deleteall() function.
// Deletes all objects in the figure list.
void deleteallobjtool::draw(void) { figlist.deleteall(); }

// Select a new line style from the line style pull-down menu.
// Note that this function overrides the same function used in the
// paint program (Chapter 11) to create a pull-down menu. Arrows
// have been added to the menu from the paint version.
void cadchangelinestyletool::draw(void)
{
    int windowwidth, windowheight;

```

```

int offset, i, mx, my;

windowwidth = LINE_WIDTH + BORDER*6;
windowheight = LINE_HEIGHT * MAX_LINESTYLES;
mouse.hide();
w.gpopup(linewindowx, linewindowy, linewindowx + windowwidth + BORDER,
          linewindowy + windowheight + LINE_HEIGHT/2,
          SOLID_LINE.getmaxcolor(), SOLID_FILL, BLACK);
setcolor(getmaxcolor());
offset = BORDER;
for (i=0; i<MAX_LINESTYLES; i++) {
    if (i >= 4) { // Line styles 4, 5, and 6 have arrows
        setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
        line(BORDER*3, offset+LINE_HEIGHT/2, BORDER*3+LINE_WIDTH,
              offset+LINE_HEIGHT/2);
        switch(i) {
            case 4 : line(BORDER*3, offset+LINE_HEIGHT/2,
                          BORDER*3+2, offset+LINE_HEIGHT/2-2);
                     line(BORDER*3, offset+LINE_HEIGHT/2,
                          BORDER*3+2, offset+LINE_HEIGHT/2+2);
                     break;
            case 5 : line(BORDER*3+LINE_WIDTH, offset+LINE_HEIGHT/2,
                          BORDER*3+LINE_WIDTH-2, offset+LINE_HEIGHT/2-2);
                     line(BORDER*3+LINE_WIDTH, offset+LINE_HEIGHT/2,
                          BORDER*3+LINE_WIDTH-2, offset+LINE_HEIGHT/2+2);
                     break;
            case 6 : line(BORDER*3, offset+LINE_HEIGHT/2,
                          BORDER*3+2, offset+LINE_HEIGHT/2-2);
                     line(BORDER*3, offset+LINE_HEIGHT/2,
                          BORDER*3+2, offset+LINE_HEIGHT/2+2);
                     line(BORDER*3+LINE_WIDTH, offset+LINE_HEIGHT/2,
                          BORDER*3+LINE_WIDTH-2, offset+LINE_HEIGHT/2-2);
                     line(BORDER*3+LINE_WIDTH, offset+LINE_HEIGHT/2,
                          BORDER*3+LINE_WIDTH-2, offset+LINE_HEIGHT/2+2);
                     break;
        }
    }
    else {
        setlinestyle(i, 0, NORM_WIDTH);
        line(BORDER*3, offset+LINE_HEIGHT/2, BORDER*3+LINE_WIDTH,
              offset+LINE_HEIGHT/2);
    }
    offset += LINE_HEIGHT;
}
mouse.show();
left_arrow = 0;
right_arrow = 0;
while (!mouse.buttonpressed(LEFT_BUTTON)) {
    mouse.getcoords(mx, my);
    offset = BORDER;
    for (i=0; i<MAX_LINESTYLES; i++) {
        if (mouse.inbox(linewindowx+BORDER*3, offset-LINE_HEIGHT/2,
                        linewindowx+BORDER*3+LINE_WIDTH,
                        linewindowy+offset+LINE_HEIGHT/2, mx, my))
            mouse.hide();
        w.gunpop();
        mouse.show();
        if (i >= 4) {
            switch(i) {
                case 4 : left_arrow = 1; break;
                case 5 : right_arrow = 1; break;
            }
        }
    }
}

```

```

        case 6 : left_arrow = 1; right_arrow = 1; break;
    }
    i = SOLID_LINE;
}
setlinestyle(i,0,NORM_WIDTH);
globallinestyle = i;
while (!mouse.buttonreleased(LEFT_BUTTON)) {
    return;
}
offset += LINE_HEIGHT;
}
mouse.hide();
w.gunpop();
mouse.show();
}

// Turn grid off if on, otherwise redraw screen with grid
void togglegridtool::draw(void)
{
    gridon = (gridon) ? 0 : 1; // Toggle grid flag
    cleardrawingarea();
    figlist.displayall();
}

// Draw a background grid in the drawing window using a six
// horizontal and vertical dotted lines. This function assumes
// that the mouse is NOT visible.
void draw_grid(void)
{
    int i, j, x, y, savecolor;
    linesettingstype saveline;
    viewporttype v;

    getviewsettings(&v);
    setviewport(0,0,maxx,maxy,1);
    savecolor = getcolor();
    getlinesettings(&saveline);
    setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
    setcolor(getmaxcolor());
    for (j=1; j<6; j++) { // Place same number here as that
        WORLDtoPC(0.0,0.0+j,x,y); // used in set_window() in
        line(wl,y,wr,y); // setup_screen()
    }
    for (i=1; i<7; i++) { // Use same number here as that
        WORLDtoPC(0.0+i,0.0,x,y); // in setup_screen()
        line(x,wt,x,wb);
    }
    setlinestyle(saveline.linestyle,saveline.upattern,saveline.thickness);
    setcolor(savecolor);
    setviewport(v.left,v.top,v.right,v.bottom,v.clip);
}

```

• 列表13.8 cad.h

```

// cad.h -- A header file used in cad.cpp and caddraw.cpp.
const int ICONW = 16; // Icons are defined as 16 by 16
const int ICONH = 16; // pixel patterns
const int FILL_HEIGHT = 16; // All 12 of the BGI fill patterns
const int NUM_FILLS_WIDE = 12; // are shown 16 pixel high in size
const int BORDER = 2; // 2 pixel border is used

```

```
const int FALSE = 0;  
const int TRUE = 1;  
const int ESC = 27;  
const int CR = 0x0d;  
const int BS = 0x08;
```

第十四章 三维图形

本章将探讨三维图形。我们的目标是开发一个图形程序包。它从某些有利点显示三维的线帧对象。我们从定义一些用于三维图形的术语开始，往下，将讨论各种三维观察程序包的成分，并说明它怎样工作。最后，将向你提供可显示的样本三维对象文件，并建议一些你可能想添加到此程序中的扩充设想。

14.1 增加第三维

在以前的章节中，我们局限于讨论二维对象的程序，例如，在第十三章提供的 CAD 程序中，建立的对象全都按世界坐标定义，并且仅使用x维和y维，这向我们提供指定高和宽的对象。现在，将通过引入第三维z把深度添加到我们的对象中。

在下面的一些节中，将开发一个称为3d.cpp的程序。它将允许我们观察线帧对象，如图14.1所示。但是，为了开发三维的程序，我们将开发新的工具并探讨有关三维图形的各种问题。

14.1.1 使用摄影机模型

生成三维对象的景色类似于用摄影机拍摄一个对象。它们都建立三维对象的三维视图。事实上，我们将使用摄影机模型帮助说明如何显示对象。

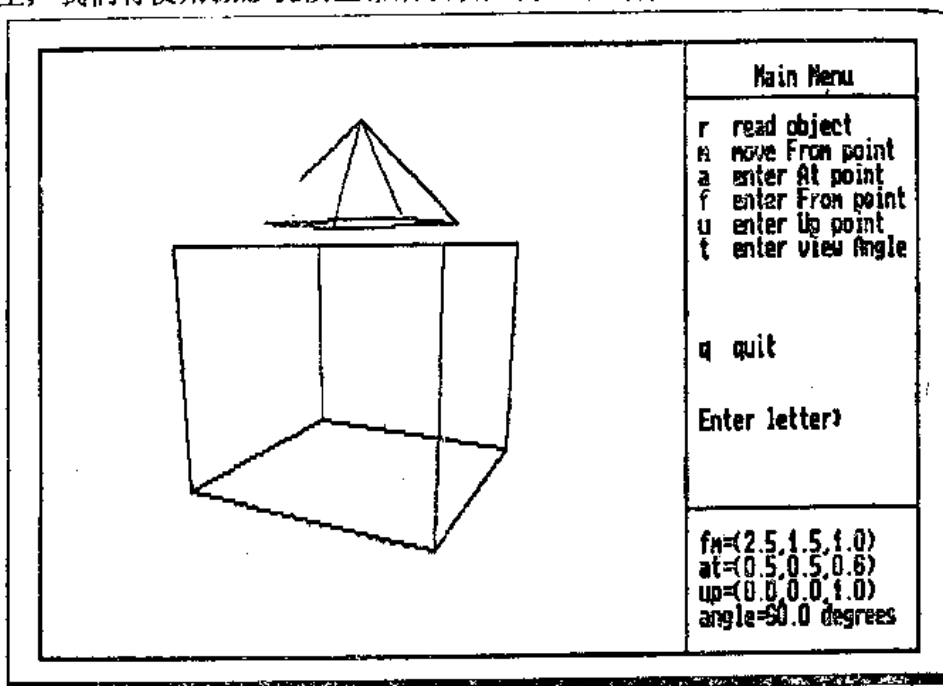


图 14.1 3d.cpp程序的环境

基本上，将假定我们正使用针孔摄影机，其中，正观察的每样东西都是清晰的。此

外，我们将假设摄影机定位在世界坐标的某个地方，即在一个称为“源”点的位置，而它正直接注视一个称为“目标”点的位置。摄影机模型还指定一观察角，它的作用十分像一个镜头，定义实际显示多少景色。最后，我们的摄影机模型包括一个称为“上行”向量的参数，它定义相对于坐标系统的观察平面的方位。图14.2解释每一个世界坐标的观察参数。

实际上，由于我们感兴趣的是对象如何相对于屏幕出现，故将使用另一种坐标系统，即“眼坐标”。它有它的原点（在源点）和目标点（在正的z轴上），如图14.3所示。术语“眼坐标”起源于这样的事实，该坐标系面向观察者——在我们的情况下是摄影机。

最后，摄影机模型使用所有对象的透视投影。换句话说，我们将把所有对象沿着从源点延伸的光线投影到观察平面。如图14.4所示。因为我们将使用透视投影，这些对象将如图14.1解释的那样变形。这给出深度的自然感觉。

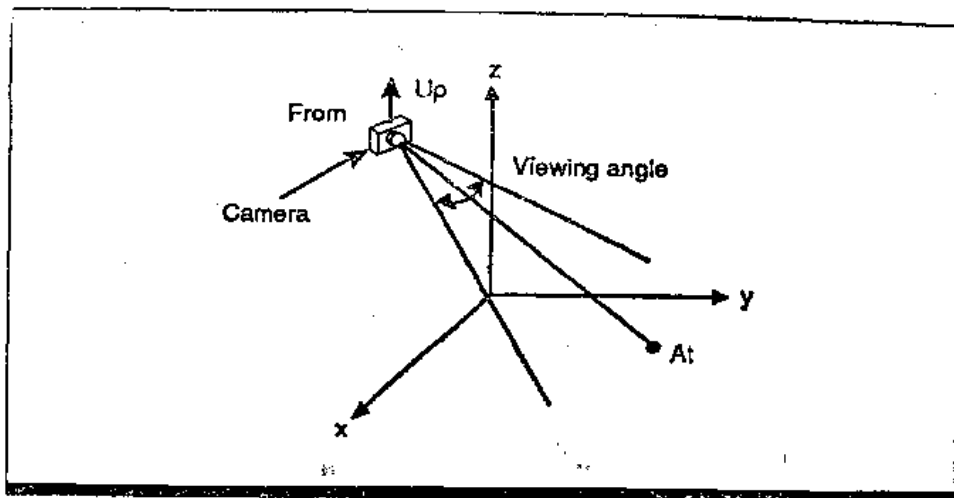


图 14.2 世界坐标的源、目标、上行和观察参数

14.1.2 一些三维的对象

为简单起见，我们将排它地处理三维线框对象，如图14.1所示。结果，每一对象由一系列的三维坐标三元组表示。它们用线段连接以建立正被显示的对象的轮廓。

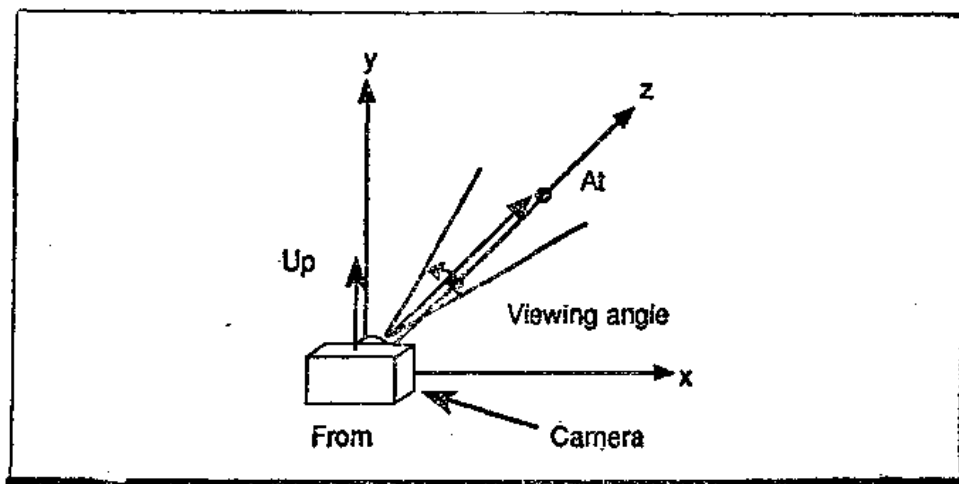


图 14.3 出现在眼坐标的观察参数

每一点（或“顶点”，通常的叫法）被说明为世界坐标的（ x 、 y 、 z ）坐标。因此，三维图形程序包的目标是把这些坐标变换为二维的屏幕坐标，然后，用线段连接它们。这个过程要求许多步骤，我们将在以下几节中讨论它们。

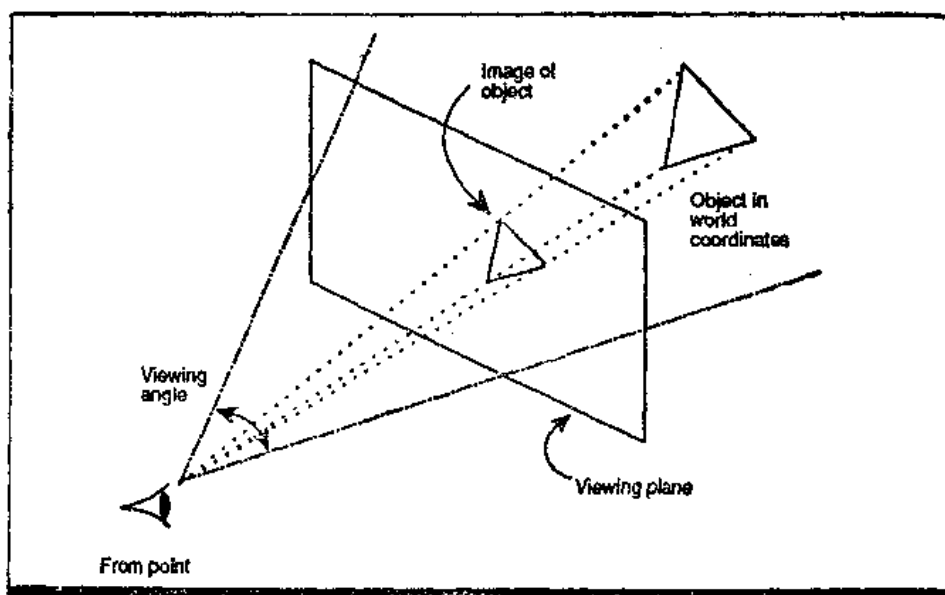


图 14.4 把三维对象投影到二维中

14.2 从世界坐标向眼坐标变换

如前所述，我们将把每一对象表示为一系列互连的顶点，其中，每一点都指定世界坐标的 x 、 y 和 z 值。但是，我们并不太感兴趣这些点定位在世界坐标的什么地方，而是它们相对于观察者（眼坐标）的位置。事实上，我们的首要步骤之一是从世界坐标向眼坐标系变换对象点。通过这样做，将很容易确定观察到的是什么对象和它们如何出现。

遗憾的是，从世界坐标向眼坐标变换三维的点不是一件寻常的事情。基本上，我们需要做的是应用一系列的变换，它们能用眼坐标系调整世界坐标系。这可以分解为以下步骤：

- (1) 平移世界坐标系以便观察者的位置（源点）是在眼坐标系的原点；
- (2) 旋转 z 轴以便目标点将落在正的 z 轴上；
- (3) 类似地旋转 y 轴；
- (4) 旋转 x 轴。

此时，这些对象可以投影到落在 z 轴的观察平面上，稍后将非常详细地描述投影的过程。

虽然刚才描述的步骤可用于在世界坐标和眼坐标之间进行变换，但它们要求大量数学运算。在 PC 上这些计算可能使系统太慢而变得不实用。

取而代之，我们将使用一向量代数技术，它可缩减必须执行的数学运算的数目，像从前一样，这个过程通过把在世界坐标上的观察者平移到眼坐标的原点开始；但是，为调整坐标轴，我们将不用一系列的旋转，而是用图 14.5 所示的步骤替换这三个步骤。我们打算推导替换先前描述的旋转的矩阵表达式 V ，但要考查一下它到底做了些什么。实质上，该

矩阵V说明单位向量 A_1 、 A_2 和 A_3 如何能调整到眼坐标系,如图14.6解释的那样。因此,

步骤 1: 用 $(-f_x, -f_y, -f_z)$ 平移这些点

步骤 2: 用 $v_{x,y,z}$ 乘步骤1所得的结果, 其中

$$V = \begin{bmatrix} a_{1x} & a_{2x} & a_{3x} & 0 \\ a_{1y} & a_{2y} & a_{3y} & 0 \\ a_{1z} & a_{2z} & a_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(A₁) (A₂) (A₃)

计算 A_1 、 A_2 和 A_3 , 假定 A' 是一个向量, 其中 $A' = A - F$, 而

$$A_3 = \frac{A'}{\|A'\|} \quad A_1 = \frac{A' \times U}{\|A' \times U\|}$$

$$A_2 = \frac{(A' \times U) \times A'}{\|(A' \times U) \times A'\|}$$

图 14.5 把世界坐标转化为眼坐标的操作

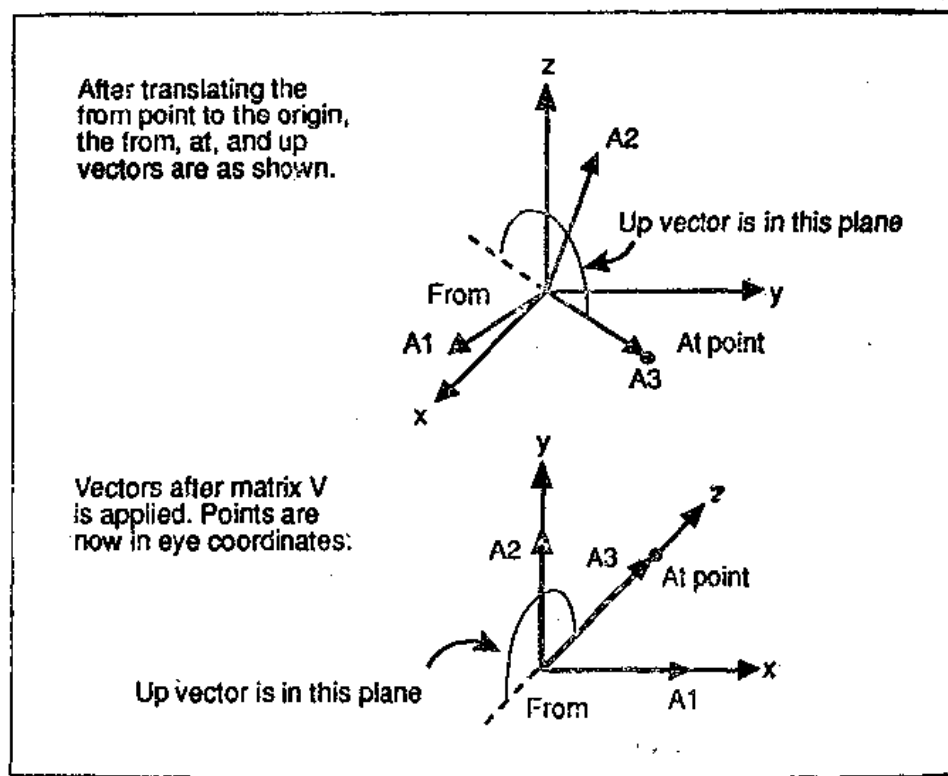


图 14.6 应用向量V前后的一些向量

通过在平移世界坐标之后,应用图14.5所示的矩阵V,我们能变换所有世界坐标为眼坐标。

此外,在把这些点转化为眼坐标之后,我们应用以下矩阵:

$$\begin{bmatrix} D & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

其中, D是

$$\frac{1}{\tan(\text{viewing_angle}/2)}$$

这个矩阵操作的目的是根据观察角度调整景色,以便该对象在线 $y = z$, $y = -z$, $x = z$ 和 $x = -z$ 之间延伸。结果,它使得裁剪容易得多。

幸好,我们能将图14.5所示的操作和上面提到的矩阵组合为少量的方程。这些方程可用于函数 `transformseg()` 在三维程序中把线从世界坐标变换为眼坐标。`transformseg()` 中的许多值都是在函数 `seteye()` 中计算的。必须在变换任一线段之前调用这个函数。你将会注意到 `seteye()` 执行图14.5描绘的大多数计算。由于许多这些操作都是向量操作,所以我们有一个独立的称为 `vector.cpp` 的实用程序执行这些操作。在列表14.1和列表14.2中提供 `vector.cpp` 的源文件和它的标题文件 `vector.h`。表14.1示出包含在 `vector.cpp` 中的函数。

表 14.1 `vector.cpp` 中的函数

函 数	描 述
<code>subtract()</code>	两个向量相减
<code>divide()</code>	用标量除向量
<code>mag()</code>	返回向量的值
<code>cross()</code>	计算两个向量的积

14.3 在三维中的裁剪

迄今为止, BGI已提供我们裁剪算法以便在显示它们时处理图形的裁剪。但是,现在我们必须开发自己的代码以裁剪三维的对象。本质上,需要做的是忽略所有未投影在投影平面的对象或裁剪掉超出投影平面上观察端口边界的对象的边缘。

为完成这点, `3d.cpp` (列表14.3) 包含函数 `clip3d()`, 和 `code()`, 它裁剪“眼坐标”的三维线段到一个“观察锥体”, 如图14.7所示。请注意, 在观察锥体中的所有对象将投影在观察平面并显示在屏幕上。这是在 `clip3d()` 的末尾完成的。我们将转到这部分, 但首先考查裁剪算法如何工作。

在把线段从世界坐标变换到眼坐标之后, 我们将把它传送给裁剪进程。这个进程从调用 `clip3d()` 开始。然后, `clip3d()` 调用 `code()`, 以确定线段的端点落在观察锥体的哪一边。如果坐标之一落在观察锥体的外边, 则置变量 `c` 中适当的位, 以指示该线可能超

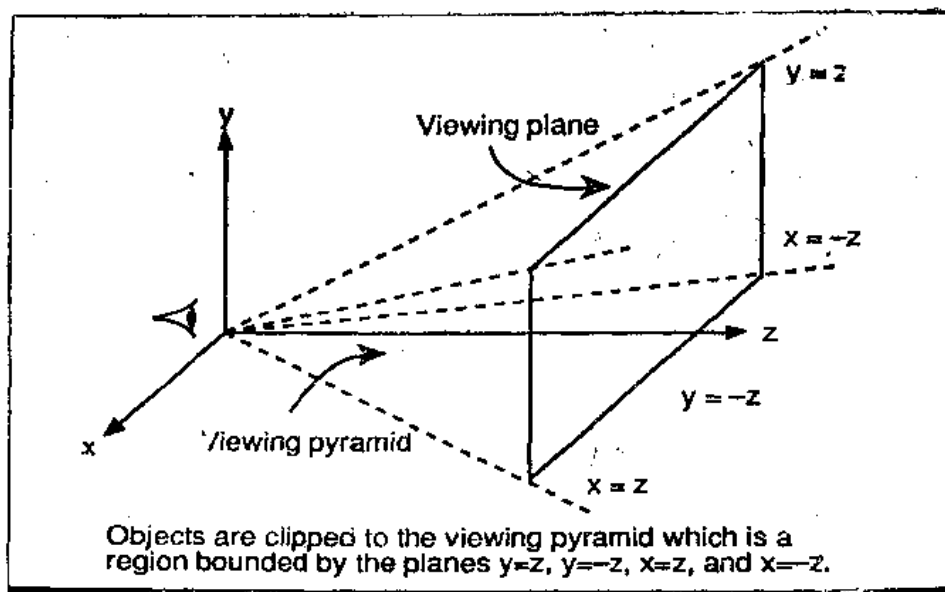


图 14.7 用观察锥体裁剪三维中的对象

越观察锥体的边缘，并且必须裁剪。通过计算线段在哪儿和观察锥体交叉，在 clip3d() 中执行裁剪。然后，这种交叉作为线的新端点，而所得的线被再传送给该进程，直到它被分解为完全在观察锥体中的段。然后显示该线。如前所述，这是在 clip3d() 的末尾完成的。让我们看看这怎样进行。

14.4 透视投影

在三维观察程序中，所有的对象都用透视投影显示。基本思想是把所有对象投影在观

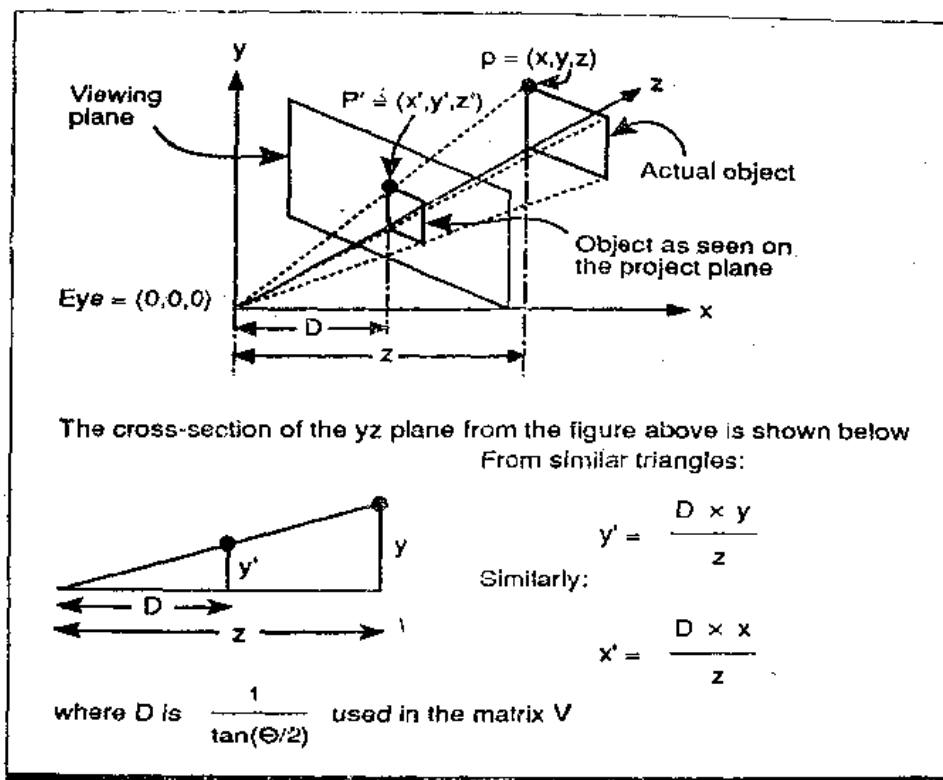


图 14.8 把对象投影到观察平面

察平面上,如图14.4所示。请注意,仅当连接对象和观察者的线通过观察平面的观察端口时,才显示这些对象。不与这个区域交叉的对象则如上节所描述的那样被裁剪。

假定我们有一段,它处在观察平面的观察端口内,我们怎么知道要画多大?幸好,我们能使用简单的类似三角的几何特性,如图14.8所示。基于这些值,把点 (x, y, z) 变换为观察平面上的 $(x/z, y/z)$ 。然后,再将这一点用函数WORLDtoPC()变化为PC坐标,就像在clip3d()中所做的那样。请注意,我们需要确保 z 为非0,以便不会用0除!一旦这些点被转化为PC屏幕坐标,则用line()来显示此线段。

14.5 对象文件

三维观察程序不允许像第十三章的CAD程序那样交互地建立对象。3d.cpp只试着显示三维对象。因此,该程序读从对象文件显示的对象。这个操作在函数read3dobject()中执行。

为了节省空间和简化修改,将以特殊的格式保存三维对象。该文件从分别说明对象的顶点数目和顶点间的连接数的标题开始。在标题之后是构成对象的顶点的一张表,其中每一顶点由三个浮点值表示,它们对应于世界坐标中点的 x 、 y 和 z 值。像早些时候提到的那样,这个表中的顶点数目由标题中的第一个值指定。往下是一系列这些顶点表的下标,这个表说明哪些顶点应和别的哪些顶点相连接。本质上,它们描述上面所读的点应如何连接以画出对象。例如,如果我们的对象是一个带有以下顶点的三角形:

(0.0, 1.0, 0.1)	顶点1
(1.0, 0.0, 0.1)	顶点2
(1.0, 0.1, 1.0)	顶点3

并且需要它们按此次序连接,则连接表应是这样的:

1 2 3 -1

这指明顶点1被连接到顶点2,它再被连接到顶点3。然后它反过来被连接到顶点1。请注意,最后的值是负的,这是在连接表中的特殊标记。它指示对象已结束或至少是对象的这面已完成。必须读入的连接数是由文件标题的第二个数目规定的。

函数read3dobject()把这些值读入表14.2列出的全程变量中,稍后根据数组connect中指定的次序存取pointarray中的值,以画出该对象。

表 14.2 存放对象的全程变量

变 量	描 述
vertices	对象中的顶点数目
length	顶点连接的数目
pointarray	按世界坐标存放的顶点数组
connect	说明顶点如何连接的索引数组

14.6 显示三维对象

现在, 我们知道怎样从文件中读三维对象, 让我们看看怎样用pointarray和connect来显示对象。这是通过在函数view() 中的嵌套while循环完成的:

```
i=1;
while (i<length) {
    startofside = i;
    i++;
    while (connect[i] > 0) {
        transformseg(pointarray[connect[i-1]],pointarray[connect[i]]);
        i++;
    }
    transformseg(pointarray[connect[i-1]],pointarray[-connect[i]]);
    transformseg(pointarray[-connect[i]],
        pointarray[connect[startofside]]);
    i++;
}
```

外层的while循环顺序通过 connect中的顶点连接表。请注意, 对C++的数组而言, 数组从1而不是传统的0下标开始。由于用负值表示一系列连接点的结束, 负0值和正0值没有区别。为此, 不使用0下标。内层的while循环顺序通过直接顶点表, 直到出现负标记为止。对于每一对连接点, 如下面那样调用transformseg() 以变换这两点, 并显示连接它们的线(如果可见的话):

```
transformseg(pointarray[connect[i-1]],pointarray[connect[i]]);
```

当connect数组达到负值时, 则显示当前的一对点, 然后用以下的行把最后的点连接回第一点:

```
transformseg(pointarray[connect[i-1]],pointarray[-connect[i]]);
transformseg(pointarray[-connect[i]],pointarray[connect[startofside]]);
```

这个过程继续到用尽connect中所有的值为止。

14.6.1 设置观察参数

为正常工作, 必须针对三维观察程序包设置若干参数。例如, 在可以观察一个对象之前, 必须把from、at、up和angle变量全部设置好。为简单起见, 给它们的每一个以缺省值, 或赋予它们可用的自动计算值。例如, 每次把一个新对象读入时通过setat()设置at位置。它调用函数minmax() 以确定该对象的边界, 然后它把at点设置在对象的中央。类似地, setfrom() 设置from点, 以便它充分远离对象, 使得整个对象将呈现在观察端口上。

此外, 还有一些函数允许你指定每一这些值, 它们可以通过选择菜单中的适当选项来改变。

14.6.2 编译3d.cpp程序

为编译3d.cpp程序, 你将需要建造一个工程文件, 它使用在表14.3中列出的文件。

表 14.3 用于建造程序3d.exe的文件

源文件	标题文件	章
gtext.cpp	gtext.h	4
gpopup.cpp	gpopup.h	10
vector.cpp	vector.h	14
3d.cpp	None	14

14.6.3 使用三维程序

在已经编译3d.cpp之后,你可准备试试显示一个对象。在下一节,示出两个三维对象的文件列表。为显示这些对象的每一个,首先把数据键入两个独立的文件中,并按指示给它们以名字,然后,通过键入字母r选择读文件命令,可以用三维程序读入这些文件。在屏幕中央出现一个上弹窗口,它提示你键入要读的文件名,你应键入这个文件名并按回车键。一旦按回车键,则读入对象文件并显示该对象。

你可以通过从屏幕选择适当的菜单选项用每一观察参数来试验。例如,为改变源点,你可以键入字母f,它将使上弹窗口提示你给出新的源点。例如,试试把源点改到世界坐标(2, 2, 2)或甚至(0, 0, 0)。后面的值会把你带进对象的里面!

另一种改变源点的方法是,使用m菜单选项,当你按上箭头时,即向对象增值步进,或者,当你按下箭头时,离开该对象。

你可能想去试验的另一方面是改变观察角。这可以通过选择菜单选项来完成,它提示你给出观察角——有效地允许你放大和缩小一个对象。

14.6.4 一些样板对象

本节列出两个对象数据文件,你可在你的三维程序中使用它们。下面是图14.9示出的对象的数据文件test1.dat:

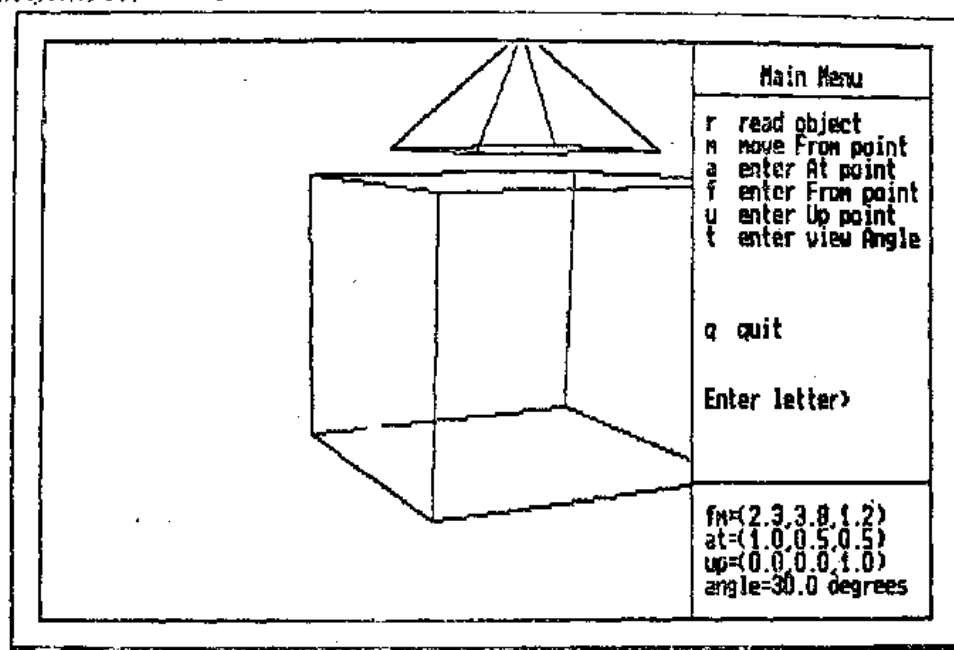


图 14.9 由文件test1.dat创建的对象


```

13 40
1.0 1.0 1.0 1.0 1.0 0.0
1.0 0.0 0.0 1.0 0.0 1.0
0.0 1.0 1.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0
0.8 0.2 1.1 0.8 0.8 1.1
0.2 0.8 1.1 0.2 0.2 1.1
0.5 0.5 1.5
1 5 8 -4 5 6 7 -8 6 2 3 -7 1 4 3 -2
8 7 3 -4 6 5 1 -2 9 10 -13 10 11 -13 11 12
-13 12 9 -13 12 11 10 -9

```

下面的数据文件称为test2.dat, 生成图14.10所示的对象:

```

38 54
1.0 1.0 1.0 1.0 1.0 0.0
1.0 0.0 0.0 1.0 0.0 1.0
0.0 1.0 1.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0
0.25 0.0 0.25 0.25 0.0 0.75
0.75 0.0 0.75 0.75 0.0 0.7
0.3 0.0 0.7 0.3 0.0 0.5
0.6 0.0 0.5 0.6 0.0 0.45
0.3 0.0 0.45 0.3 0.0 0.25
1.0 0.3 0.2 1.0 0.3 0.3
1.0 0.6 0.3 1.0 0.6 0.5
1.0 0.3 0.5 1.0 0.3 0.8
1.0 0.7 0.8 1.0 0.7 0.7
1.0 0.4 0.7 1.0 0.4 0.6
1.0 0.7 0.6 1.0 0.7 0.2
0.4 0.3 1.0 0.4 0.6 1.0
0.2 0.6 1.0 0.2 0.7 1.0
0.8 0.7 1.0 0.8 0.6 1.0
0.5 0.6 1.0 0.5 0.3 1.0
1 5 8 -4 5 6 7 -8 6 2 3 -7 1 4 3 -2
8 7 3 -4 6 5 1 -2 9 10 11 12 13 14 15 16
17 -18 19 20 21 22 23 24 25 26 27 28 29 -30 31 32 33 34
35 36 37 -38

```

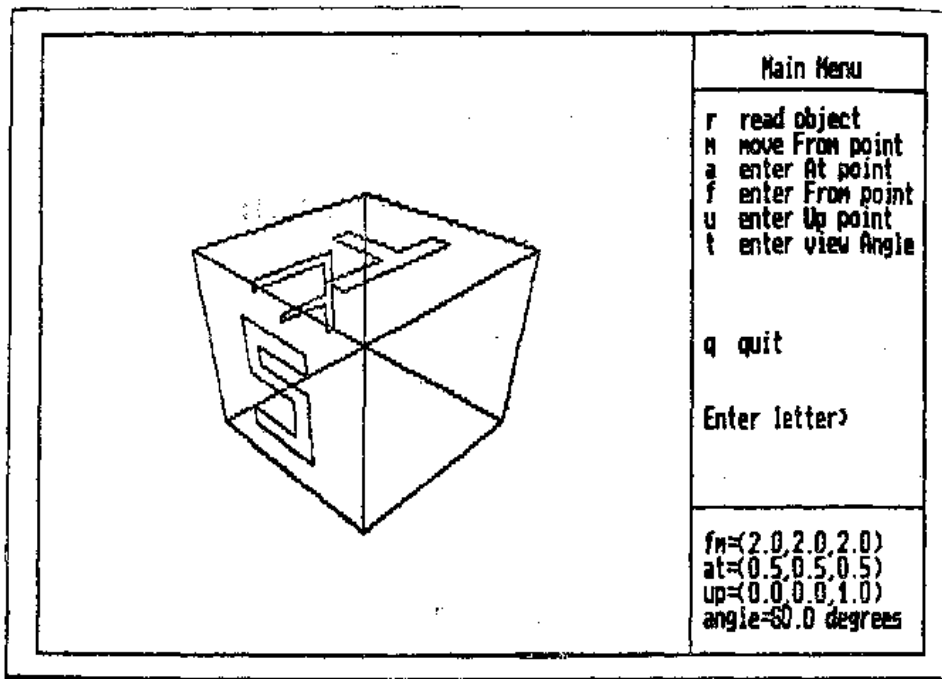


图 14.10 由文件test2.dat创建的显示

14.7 扩充三维程序

有一些你可能试图添加到三维观察程序包的增强功能，你想试的第一件事情也许是增加能交互地画三维对象的函数。这里的困难来自交互地画对象的清晰方法。你将大概需要使用几个观察端口，它们以不同的透视图示出该对象，为的是向你提供你所画的对象的完美概念。

另一种可能的程序扩充是向所画的对象添加颜色或填充图案。由于我们的文件格式把对象的边缘组合成为一些面（通过使用connect数组），你可以试试用fillpoly()一次画对象的各面而不是只画它们的一条线。用这种方法，将使对象的各面成为实心。但是，你将发现，为正确地画出对象，画哪一面的次序是重要的。

• 列表14.1 vector.h

```
// vector.h -- Header file for vector.cpp.
struct VECTOR {
    double x,y,z;
};

double mag(VECTOR *v);
VECTOR subtract(VECTOR *v1, VECTOR *v2);
VECTOR cross(VECTOR *v1, VECTOR *v2);
VECTOR divide(VECTOR *v, double num);
```

• 列表14.2 vector.cpp

```
// vector.cpp -- Vector operations for 3d.cpp.
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "vector.h"

// Calculate the magnitude of the vector
double mag(VECTOR *v)
{
    return(sqrt(v->x * v->x + v->y * v->y + v->z * v->z));
}

// Subtract the two vectors
VECTOR subtract(VECTOR *v1, VECTOR *v2)
{
    VECTOR d;

    d.x = v1->x - v2->x;
    d.y = v1->y - v2->y;
    d.z = v1->z - v2->z;
    return(d);
}

// Cross multiply the two vectors v1 and v2
VECTOR cross(VECTOR *v1, VECTOR *v2)
{
    VECTOR c;
```

```

    c.x = v1->y * v2->z - v2->y * v1->z;
    c.y = v1->z * v2->x - v2->z * v1->x;
    c.z = v1->x * v2->y - v2->x * v1->y;
    return(c);
}

// Divide the scalar number into the vector v
VECTOR divide(VECTOR *v, double num)
{
    VECTOR result;

    if (num == 0) {
        printf("Divide by 0 in Matrix Divide Operation\n");
        exit(1);
    }
    result.x = v->x / num;
    result.y = v->y / num;
    result.z = v->z / num;
    return(result);
}

```

• 列表14.3 3d.cpp

```

// 3d.cpp -- Wire-frame viewing package. Displays wire-frame three-
// dimensional views of objects using perspective projection. Objects
// are read from files and displayed according to the settings of the
// from, at, up, and viewing angle. This program is designed to
// be used with a display mode that has a resolution of 640x200.
// If you use a mode with a different resolution, you may need to
// change the constants PCL, PCR, PCT, and PCB in order to get the
// program to display figures with the correct proportions.
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <string.h>
#include <math.h>
#include <conio.h>
#include "gtext.h"
#include "vector.h"
#include "gpopup.h"

const int NUMCONNECTIONS = 100; // An object can have this many
const int NUMVERTICES = 150;   // vertices and connections.
const int PCL = 1;              // Borders of viewing region on
const int PCR = 469;            // the screen. These four variables
const int PCT = 1;              // correspond to the left, right,
const int PCB = 198;            // top, and bottom pixel boundaries
                                // of the viewing region.

// Returns the larger of two values
inline double maxof(double val1, double val2) {
    return (val1 > val2) ? val1 : val2;
}

// Converts an angle to radians
inline double torad(double degrees) {
    return degrees * 0.017453293;
}

double a, b, c, d, dval;
VECTOR from, at, up;           // Viewing parameters

```

```

double angle; // The viewing angle
VECTOR a1, a2, a3; // Used in three-dimensional transform
int connect[NUMCONNECTIONS]; // Vertex connections
VECTOR pointarray[NUMVERTICES]; // Vertices
int length; // Number of vertex connections
int vertices; // Number of vertices
double objxmin, objxmax; // Extent of three-dimensional object
double objymin, objymax;
double objzmin, objzmax;
double t_from; // Controls from-at point dist
double offsx, offsy, offsz; // Transform variables
VECTOR dist; // Distance to from-at point
int maxx, maxy; // Size of the screen
int pcb, pct, pci, pcr; // PC screen boundaries of viewing area
gwindows w; // Declare a window object

// Function prototypes
void init3dgraphics(void);
void minmax(void);
void setat(void);
void setfrom(void);
void seteye(void);
int code(double x, double y, double z);
void clip3d(double x1, double y1, double z1, double x2,
            double y2, double z2);
int readobject(char *filename);
void transformseg(VECTOR *v1, VECTOR *v2);
void view(void);
void showvalues(void);
void clear_viewport(void);
void WORLDtoPC(double xw, double yw, int *xpc, int *ypc);
void movefrom(void);
void getfrom(void);
void getat(void);
void getup(void);
void getangle(void);
void read3dobjectfile(void);
void mainmenu(void);
void popuperror(char *message);

main()
{
    char ch, *prompt = "Enter letter> ";

    init3dgraphics();
    rectangle(pci-1, pct-1, pcr+1, pcb+1);
    mainmenu();
    showvalues();
    do {
        gprintfxy(60*8, 15*8, "%s ", prompt);
        moveto(60*8+textwidth(prompt), 15*8);
        ch = ggetche();
        switch(ch) {
            case 'm' : movefrom(); break;
            case 'r' : read3dobjectfile(); break;
            case 'a' : getat(); break;
            case 'f' : getfrom(); break;
            case 'u' : getup(); break;
            case 't' : getangle(); break;
        }
        showvalues();
    } while (ch != 'q');
    closegraph();
}

```

```

    return(0);
}

// Converts clipped world coordinates to screen coordinates
void WORLDtoPC(double xw, double yw, int *xpc, int *ypc)
{
    *xpc = (int)(a * xw + b);
    *ypc = (int)(c * yw + d);
}

// Must be called before program begins main execution. It sets
// up the dimensions of the screen and various default values.
void init3dgraphics(void)
{
    int gmode, gdriver = DETECT;
    int errcode;
    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    if ((errcode = graphresult()) != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errcode));
        exit(1); // Return with an error code
    }

    // Setup the dimensions of the viewing region
    maxx = getmaxx(); maxy = getmaxy();
    pcb = maxy - 1; pct = PCT;
    pcr = PCR; pcl = PCL;
    a = (pcr - pcl) / (1 + 1); // Set viewport and window
    b = pcl - a * (-1); // mapping variables
    c = (pct - pcb) / (1 + 1);

    d = pcb - c * (-1);
    // Set default values for from, at, and up vectors
    from.x = 1.0; from.y = 0.0; from.z = 0.0;
    at.x = 0.0; at.y = 0.0; at.z = 0.0;
    up.x = 0.0; up.y = 0.0; up.z = 1.0;
    angle = torad(60.0);
    t_from = 1.0;
}

// Return the minimum and maximum values in the point array
// for the x, y, and z values
void minmax(void)
{
    int i;

    objxmin = 32000; objymin = 32000; objzmin = 32000;
    objxmax = -32000; objymax = -32000; objzmax = -32000;
    for (i = 0; i < vertices; i++) {
        if (pointarray[i].x > objxmax) objxmax = pointarray[i].x;
        else if (pointarray[i].x < objxmin) objxmin = pointarray[i].x;
        if (pointarray[i].y > objymax) objymax = pointarray[i].y;
        else if (pointarray[i].y < objymin) objymin = pointarray[i].y;
        if (pointarray[i].z > objzmax) objzmax = pointarray[i].z;
        else if (pointarray[i].z < objzmin) objzmin = pointarray[i].z;
    }
}

// Routine to provide a default value for the at point. It
// is set to the midpoint of the extents of the object.
void setat(void)
{
    minmax();
}

```

```

    at.x = (objxmin+objxmax) / 2.0;
    at.y = (objymin+objymax) / 2.0;
    at.z = (objzmin+objzmax) / 2.0;
}

// Routine that provides a default value for the from point. It
// is dependent on the At point and the view angle.
const double WIDTH = 1.8; // Ratio used to determine from point
                          // It is based on size of object
void setfrom(void)
{
    from.x = at.x + (objxmax-objxmin) / 2.0 + WIDTH *
        maxof((objzmax-objzmin)/2.0, (objymax-objymin)/2.0);
    from.y = at.y;
    from.z = at.z;
}

// There must be a valid object in pointarray before calling this
// function. It sets up the various variables used in transforming
// an object from world to eye coordinates.
void seteye(void)
{
    double amarkmag, tempmag;
    VECTOR temp;

    dval = cos(angle/2.0) / sin(angle/2.0);
    dist = subtract(&at,&from);
    amarkmag = mag(&dist);
    a3 = divide(&dist,amarkmag);
    temp = cross(&dist,&up);
    tempmag = mag(&temp);
    a1 = divide(&temp,tempmag);
    temp = cross(&a1,&a3);
    tempmag = mag(&temp);
    a2 = divide(&temp,tempmag);
    offsx = -a1.x * from.x - a1.y * from.y - a1.z * from.z;
    offsy = -a2.x * from.x - a2.y * from.y - a2.z * from.z;
    offsz = -a3.x * from.x - a3.y * from.y - a3.z * from.z;
}

const int NOEDGE = 0x00;
const int LEFTEDGE = 0x01;
const int RIGHTEDGE = 0x02;
const int BOTTOMEDGE = 0x04;
const int TOPEDGE = 0x08;

// Returns a code specifying which edge in the viewing
// pyramid was crossed. There may be more than one.
int code(double x, double y, double z)
{
    int c;

    c = NOEDGE;
    if (x < -z) c |= LEFTEDGE;
    if (x > z) c |= RIGHTEDGE;
    if (y < -z) c |= BOTTOMEDGE;
    if (y > z) c |= TOPEDGE;
    return(c);
}

// Clips the line segment in 3D coordinates to the viewing pyramid

```

```

void clip3d(double x1, double y1, double z1, double x2,
           double y2, double z2)
{
    int c, c1, c2, xpc1, ypc1, xpc2, ypc2;
    double x, y, z, t;

    c1 = code(x1, y1, z1);
    c2 = code(x2, y2, z2);
    while ((c1 != NOEDGE) || c2 != NOEDGE) {
        if ((c1 & c2) != NOEDGE) return;
        c = c1;
        if (c == NOEDGE) c = c2;
        if ((c & LEFTEDGE) == LEFTEDGE) {
            // Crosses left edge
            t = (z1 - x1) / ((x1 - x2) - (z2 - z1));

            z = t * (z2 - z1) + z1;
            x = -z;
            y = t * (y2 - y1) + y1;
        }
        else if ((c & RIGHTEDGE) == RIGHTEDGE) {
            // Crosses right edge
            t = (z1 - x1) / ((x2 - x1) - (z2 - z1));
            z = t * (z2 - z1) + z1;
            x = z;
            y = t * (y2 - y1) + y1;
        }
        else if ((c & BOTTOMEDGE) == BOTTOMEDGE) {
            // Crosses bottom edge
            t = (z1 + y1) / ((y1 - y2) - (z2 - z1));
            z = t * (z2 - z1) + z1;
            x = t * (x2 - x1) + x1;
            y = -z;
        }
        else if ((c & TOPEdge) == TOPEdge) {
            // Crosses top edge
            t = (z1 - y1) / ((y2 - y1) - (z2 - z1));
            z = t * (z2 - z1) + z1;
            x = t * (x2 - x1) + x1;
            y = z;
        }
        if (c == c1) {
            x1 = x; y1 = y; z1 = z;
            c1 = code(x, y, z);
        }
        else {
            x2 = x; y2 = y; z2 = z;
            c2 = code(x, y, z);
        }
    }
    if (z1 != 0) {
        WORLDtoPC(x1/z1, y1/z1, &xpc1, &ypc1);
        WORLDtoPC(x2/z2, y2/z2, &xpc2, &ypc2);
    }
    else {
        WORLDtoPC(x1, y1, &xpc1, &ypc1);
        WORLDtoPC(x2, y2, &xpc2, &ypc2);
    }
    line(xpc1, ypc1, xpc2, ypc2);
}

```

```

// Transform the segment connecting the two vectors into
// the viewing plane. 3dclip() clips and draws the line if visible.
void transformseg(VECTOR *v1, VECTOR *v2)
{
    double x1,y1,z1,x2,y2,z2;

    x1 = (v1->x * a1.x + a1.y * v1->y + a1.z * v1->z + offsx)*dval;
    y1 = (v1->x * a2.x + a2.y * v1->y + a2.z * v1->z + offsy)*dval;
    z1 = (v1->x * a3.x + a3.y * v1->y + a3.z * v1->z + offsz);

    x2 = (v2->x * a1.x + a1.y * v2->y + a1.z * v2->z + offsx)*dval;
    y2 = (v2->x * a2.x + a2.y * v2->y + a2.z * v2->z + offsy)*dval;
    z2 = (v2->x * a3.x + a3.y * v2->y + a3.z * v2->z + offsz);
    clip3d(x1,y1,z1,x2,y2,z2);
}

// Increment through the pointarray which contains the vertices of the
// object and display them as you go. This will draw out the object.
void view(void)
{
    int i=1, startofside;

    while (i<length) {
        startofside = i++;
        while (connect[i] > 0) {
            transformseg(&pointarray[connect[i-1]],
                        &pointarray[connect[i]]);
            i++;
        }
        transformseg(&pointarray[connect[i-1]],
                    &pointarray[-connect[i]]);
        transformseg(&pointarray[-connect[i]],
                    &pointarray[connect[startofside]]);
        i++;
    }
}

// Clear the drawing window
void clear_viewport(void)
{
    setviewport(pcl,pct,pcr,pcb,1);
    clearviewport();
    setviewport(0,0,getmaxx(),getmaxy(),1);
}

// Show the from, at, up, and angle values in a window on
// the bottom right of the screen
void showvalues(void)
{
    // Erase the area where the values are to be written
    setfillstyle(SOLID_FILL, BLACK);
    bar3d(pcr+1,19*8,maxx,maxy,0,0);
    gprintfxy(60*8,19*8+textheight("F"),
              "fm=(%3.1f,%3.1f,%3.1f)",from.x,from.y,from.z);
    gprintfxy(60*8,19*8+2*textheight("F"),
              "at=(%3.1f,%3.1f,%3.1f)",at.x,at.y,at.z);
    gprintfxy(60*8,19*8+3*textheight("F"),
              "up=(%3.1f,%3.1f,%3.1f)",up.x,up.y,up.z);
    gprintfxy(60*8,19*8+4*textheight("F"),
              "angle=%3.1f degrees",angle/0.017453293);
}

```



```

// Interactively specify the from coordinates. It cannot
// be the same as the at point.

void getfrom(void)
{
    VECTOR nu;
    static char *title = "    Change the From Point    ";

    w.gpopup(maxx/2-textwidth(title)/2-5,maxy/2-3*textheight("H"),
             maxx/2+textwidth(title)/2+5,maxy/2+4*textheight("H"),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,title);
    gprintfxy(5,5+2*textheight("H"),"Enter x y z coordinates");
    gprintfxy(5,5+3*textheight("H"),"of new from point.");
    gprintfxy(5,5+4*textheight("X"),"> ");
    if (gscanfxy(5+textwidth("> "),5+4*textheight("H"),
                "%lf %lf %lf",&nu.x,&nu.y,&nu.z) < 3) {
        w.gunpop();
        return;
    }
    w.gunpop();
    if (nu.x == at.x && nu.y == at.y && nu.z == at.z)
        popuperror(" Invalid From Point ");
    else {
        from.x = nu.x; from.y = nu.y; from.z = nu.z;
        clear_viewport();
        seteye();
        view();
    }
}

// Let the user change the at point. The at point should
// not be the same as the from point or the up vector.
void getat(void)
{
    double nux, nuy, nuz;
    static char *title = "    Change the At Point    ";

    w.gpopup(maxx/2-textwidth(title)/2-5,maxy/2-3*textheight("H"),
             maxx/2+textwidth(title)/2+5,maxy/2+4*textheight("H"),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,title);
    gprintfxy(5,5+2*textheight("H"),"Enter x y z coordinates");
    gprintfxy(5,5+3*textheight("H"),"of new At point.");
    gprintfxy(5,5+4*textheight("X"),"> ");
    if (gscanfxy(5+textwidth("> "),5+4*textheight("H"),
                "%lf %lf %lf",&nux,&nuy,&nuz) < 3) {
        w.gunpop();
        return;
    }
    w.gunpop();
    if ((nux==from.x && nuy==from.y && nuz==from.z) ||
        (nux==up.x && nuy==up.y && nuz==up.z))
        popuperror(" Invalid From Point ");
    else {
        at.x = nux; at.y = nuy; at.z = nuz;
        clear_viewport();

        seteye();
        view();
    }
}

```

```

// Change the up vector coordinate to the user coordinates
// indicated. It should not be the same as the at point.
void getup(void)
{
    VECTOR nu;
    static char *title = "    Change the Up Point    ";
    w.gpopup(maxx/2-textwidth(title)/2-5,maxy/2-3*textheight("H"),
             maxx/2+textwidth(title)/2+5,maxy/2+4*textheight("H"),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,title);
    gprintfxy(5,5+2*textheight("H"),"Enter x y z coordinates");
    gprintfxy(5,5+3*textheight("H"),"of new Up point.");
    gprintfxy(5,5+4*textheight("X"),"> ");
    if (gscanfxy(5+textwidth("> ").5+4*textheight("H"),
                "%lf %lf %lf",&nu.x,&nu.y,&nu.z) < 3) {
        w.gunpop();
        return;
    }
    w.gunpop();
    if (nu.x==at.x && nu.y==at.y && nu.z==at.z)
        popuperror(" Invalid Up vector ");
    else {
        up.x = nu.x; up.y = nu.y; up.z = nu.z;
        seteye();
        clear_viewport();
        view();
    }
}

// Ask the user for a viewing angle. It must be between 0 and 180 degree
void getangle(void)
{
    double nut;
    static char *title = "    Change the Viewing Angle    ";
    w.gpopup(maxx/2-textwidth(title)/2-5,maxy/2-3*textheight("H"),
             maxx/2+textwidth(title)/2+5,maxy/2+4*textheight("H"),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,title);
    gprintfxy(5,5+2*textheight("H"),"Enter a new angle between");
    gprintfxy(5,5+3*textheight("H"),"0 and 180 degrees.");
    gprintfxy(5,5+4*textheight("X"),"> ");
    if (gscanfxy(5+textwidth("> ").5+4*textheight("H"),
                "%lf",&nut) < 1) {
        w.gunpop();
        return;
    }
    w.gunpop();
    if (nut > 0 && nut < 180) {
        angle = torad(nut);
        seteye();
        clear_viewport();
        view();
    }
    else
        popuperror(" Angle must be between 0 and 180 ");
}

// Interactively move the from point towards or away from
// the at point depending upon the user input.
void movefrom(void)

```

```

{
    const double FROM_INC = 0.1;
    char ch;
    double nufx,nufy,nufz;
    VECTOR startv;

    startv.x = from.x; startv.y = from.y;
    startv.z = from.z;
    t_from = 1.0;
    gprintfxy(60*8,15*8,"Up arrow moves in ");
    gprintfxy(60*8,16*8,"Down to move away ");
    gprintfxy(60*8,17*8,"Any key to quit ");
    while ((ch=getch()) == 0) {
        ch = getch();
        if (ch == 0x48 || ch == 0x50) { // Move in or out using
            if (ch == 0x48) { // UP and DOWN keys
                if (t_from > FROM_INC) t_from -= FROM_INC;
            }
            else
                t_from += FROM_INC;
            nufx = at.x + t_from * (startv.x - at.x);
            nufy = at.y + t_from * (startv.y - at.y);
            nufz = at.z + t_from * (startv.z - at.z);
            if (nufx == at.x && nufy == at.y && nufz == at.z)
                popuerror(" Cannot move to here ");
            else {
                from.x = nufx; from.y = nufy; from.z = nufz;
                seteye();
                clear_viewport();
                view();
                showvalues();
            }
        }
    }
    setfillstyle(SOLID_FILL,BLACK);
    bar(60*8,15*8,maxx-2,18*8-1);
}

```

// Read in a file describing a polygon which adheres to the
// standard described above. Returns 1 if file is read
// successfully; otherwise it returns 0.

```

int read3dobject(char *filename)
{
    FILE *infile;
    int i;

    if ((infile = fopen(filename,"r")) == NULL) {
        popuerror(" Could not open file. ");
        return(0);
    }
    fscanf(infile,"%d %d",&vertices,&length);
    if (vertices >= NUMVERTICES || length >= NUMCONNECTIONS) {
        popuerror(" Object in file is too large. ");
        return(0);
    }
    for (i=1; i<=vertices; i++)
        fscanf(infile,"%lf %lf %lf",&pointarray[i].x,
            &pointarray[i].y,&pointarray[i].z);
    for (i=1; i<=length; i++)
        fscanf(infile,"%d",&connect[i]);
    fclose(infile);
}

```

```

    return(1);
}

// Prompts for the file name of the file to be read and then calls
// read3dobject() to read it. If the file is successfully read, it is
// displayed by calling view() after setting the viewing parameters.
void read3dobjectfile(void)
{
    char fn[80];
    static char *title = " Reading a 3D Object File ";
    w.gpopup(maxx/2-textwidth(title)/2-5,maxy/2-3*textheight("H"),
             maxx/2+textwidth(title)/2+5,maxy/2+4*textheight("H"),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,title);
    gprintfxy(5,5+2*textheight("H"),"Enter filename:");
    moveto(5,5+3*textheight("X"));
    gprintf("> ");
    ggets(fn);
    if (strlen(fn) > 0 && read3dobject(fn)) {
        w.gunpop();
        setat();
        setfrom();
        seteye();
        clear_viewport();
        view();
    }
    else
        w.gunpop();
}

// Listing for the main menu
void mainmenu(void)
{
    rectangle(pcr+1,0,maxx,19*8);
    gprintfxy(60*8,1*8," Main Menu");
    rectangle(pcr+1,0,maxx,textheight("M")*2+2);
    gprintfxy(60*8,3*8,"r read object");
    gprintfxy(60*8,4*8,"m move From point");
    gprintfxy(60*8,5*8,"a enter At point");
    gprintfxy(60*8,6*8,"f enter From point");
    gprintfxy(60*8,7*8,"u enter Up point");
    gprintfxy(60*8,8*8,"t enter view Angle");
    gprintfxy(60*8,12*8,"q quit");
}

// Popup an error message
void popuperror(char *message)
{
    int width;
    static char *press = " Press ENTER to Continue ";
    width = textwidth(message);
    if (width < textwidth(press))
        width = textwidth(press);
    w.gpopup(maxx/2,maxy/2+20,maxx/2+width+10,
             maxy/2+20+3*(textheight("H")+2),
             SOLID_LINE,getmaxcolor(),SOLID_FILL,BLACK);
    gprintfxy(5,5,message);
    gprintfxy(5,5+textheight("H"),press);
    getch();
    w.gunpop();
}

```

附录：BGI函数参考

本附录列出BGI包含的函数。每一函数包括函数的简短描述和它的函数原型，以及它的参数和返回值的解释（如果有的话）。为使用这些函数，必须在你的源文件中包含标题文件graphics.h，并把图形程序库graphics.lib与你的编译文件相连接。

arc() 在指定的起始点（中心）用确定的半径画圆弧。从初始角到结束角画弧。

```
void far arc(int x, int y, int stangle, int endangle, int radius);
```

x, y	用绝对坐标指定弧的中心。
stangle	按度指定初始角。
endangle	按度指定结束角。
radius	按像素指定半径。

bar() 用当前绘图颜色和填充图案绘制填充的矩形条形图。
bar() 函数不画条形的轮廓。

```
void far bar(int left, int top, int right, int bottom);
```

left, top	条形的左上角。
right, bottom	右下角。

bar3d() 用当前绘图颜色和填充图案绘制填充的三维矩形条形图。

```
void far bar3d(int left, int top, int right, int bottom, int depth,  
               int topflag);
```

left, top	条形的左上角。
right, bottom	右下角。
depth	按像素指定条形的三维深度。
topflag	指示是否在条形上带一个顶。非0值指定画一个顶。

circle() 用指定的起始点（中心）上确定的半径画圆。用当前绘图颜色画圆。圆仅可以用缺省线型（实线）画。但是在用setlinestyle()函数画圆之前可以设置线的厚度。

`void far circle(int x, int y, int radius);`

<code>x, y</code>	用绝对坐标指定圆心。
<code>radius</code>	按像素指定半径。

`cleardevice()`

清除整个图形屏幕，并把当前的位置 (cp) 移动到活动观察端口中的0, 0。即使观察端口被设置为比全屏幕较小的尺寸，每当需要清除全屏幕时，也应使用 `cleardevice()` 例程。如果仅需清除当前的观察端口，则应使用 `clearviewport()` 函数。

`void far cleardevice(void);`

`clearviewport()`

清除活动观察端口，并把当前位置 (cp) 移动到活动观察端口的0, 0。仅清除活动观察端口。如果观察端口比全屏幕小，则不改变观察端口以外的区域。

`void far clearviewport(void);`

`closegraph()`

关闭BGI图形系统和重分配为图形驱动程序、字体和缓冲区分配的存储区。恢复屏幕为它在BGI初始化以前的模式。每当你的程序不再需要BGI例程时，你应当总调用此例程，使得释放存储区。

`void far closegraph(void);`

`detectgraph()`

检验硬件以确定要使用哪一个图形驱动程序和模式。此例程总选择对所安装的适配器给出可能的最高分辨率的模式。如果由于某些原因 `detectgraph()` 找不到图形适配器，则 `graphdriver` 参数置为 -2。

`void far detectgraph(int far *graphdriver, int far *graphmode);`

<code>graphdriver</code>	返回指示可以使用哪一个图形驱动程序的整型码。
<code>graphmode</code>	返回可以使用的最高分辨率模式的整型码。

`drawpoly()`

使用当前绘图颜色和线型画多边形。

`void far drawpoly(int numpoints, int far *polypoints);`

numpoints	用于画多边形的点（坐标对）的数目。
polypoints	用于多边形的点的数组。每一点由x和y坐标对表示（首先指定x）。对于作为封闭图形画出的多边形，其第一点和最后一点必须是相同的。

ellipse() 在指定的起始点（中心）用确定的水平半径和垂直半径画椭圆形的弧。从起始角到结束角画椭圆。你可以通过设置起始角为0和结束角为360画完整的椭圆（端点相连）。按反时针方向画椭圆，其中0°在3点钟，90°在12点钟。你不能改变椭圆的线型，但是可以使用 **setlinestyle()** 置线的厚度。

```
void far ellipse(int x, int y, int stangle, int endangle, int xradius,
                 int yradius);
```

x, y	用绝对坐标指定弧的中心。
stangle, endangle	按度指定初始角和结束角。
xradius, yradius	按像素指定水平半径和垂直半径。

fillellipse() 用当前的填充图案和颜色画和填充椭圆。此函数画一个完全椭圆。如果你想要画部分椭圆，则使用 **ellispe()** 例程。

```
void far fillellipse(int x, int y, int xradius, int yradius);
```

x, y	用绝对坐标指定椭圆的中心。
xradius, yradius	按像素指定水平半径和垂直半径。

fillpoly() 用当前的线型和填充类型以及绘图和填充颜色画和填写多边形。如果你想画多边形而不填充它，则使用 **drawpoly()**。

```
void far fillpoly(int numpoints, int far *polypoints);
```

numpoints	用于画和填写多边形的点数。
polypoints	用于多边形的点的数组，每一点用x和y坐标对表示。

floodfill() 用当前颜色和填充图案喷流填充有界的区域。如果喷流填充的指定点在有界区域以外，则填充除有界区域外的所有地方。

```
void far floodfill(int x, int y, int border);
```

`z, y`

填充区域的位置。如果此位置在封闭区内，则填充区域。如果此位置在封闭区以外，则除封闭区外填充每个地方。

`border`

边界的颜色码。

`getarccoords()`

取最近的画弧命令的坐标。返回的坐标是弧的中心点和起始与结束终点坐标。

```
void far getarccoords(struct arccoordstype far *arccoords);
```

`arccoords`

用于送回弧坐标的名为`arccoordstype`的预定义数据类型。此数据类型定义为：

```
struct arccoordstype {  
    int x, y; // Center point  
    int xstart, ystart, xend, yend; // Endpoints  
};
```

`getaspectratio()`

确定现用图形驱动程序和模式的当前长宽比。

```
void far getaspectratio(int far *xasp, int far *yasp);
```

`xasp, yasp`

用于返回x和y长宽比的变量。

`getbkcolor()`

返回当前背景颜色。背景颜色作为整型码返回，范围可以从0到15。

```
int far getbkcolor(void);
```

`getcolor()`

返回当前的绘图颜色，绘图颜色作为整数码返回，范围可以从0到15。

```
int far getcolor(void);
```

`getdefaultpalette()`

返回包含在缺省彩色调色板的颜色码。CGA的缺省调色板的大小是在低分辨率模式下的4种颜色和在高分辨率模式下的2种颜色。EGA和VGA两者都在它们的缺省调色板上有16种颜色。以讨论`getpalette()`函数时提供的预定义数据结构返回调色板信息。

```
struct palettetype *far getdefaultpalette(void);
```

`getdrivername()`

返回当前安装的图形驱动程序名。

```
char *far getdrivername(void);
```


`getfillpattern()` 获得用户定义的填充图案的位设置。64像素填充图案表示为8字节序列，其中每一字节与填充图案中的8个像素相对应。如果你需要确定现用的填充颜色，则使用`getfillsettings()`。

```
void far getfillpattern(char far *upattern);
```

`upattern` 用户定义的填充图案。此参数是指向8字节序列（位模式）的指针。

`getfillsettings()` 得到当前填充图案和颜色。在第三章表3.7中列出的代码之一将作为填充图案返回。

```
void far getfillsettings(struct fillsettingstype far *fillinfo);
```

`fillinfo` 包含填充图案和填充颜色的数据结构。此数据结构定义为：

```
struct fillsettingstype {
    int pattern;    // Fill pattern id
    int color;      // Fill color
};
```

`getgraphmode()` 返回安装的图形驱动程序的当前图形模式。

```
int far getgraphmode(void);
```

`getimage()` 把矩形屏幕位映像拷贝到存储。稍后可以通过调用`putimage()`恢复此映像。

```
void far getimage(int left, int top, int right, int bottom,
                  void far *bitmap);
```

`left, top` 屏幕矩形的左上角。

`right, bottom` 屏幕矩形的右下角。

`bitmap` 参引存放位映像处的存储位置。

`getlinesettings()` 得到包括当前线型、图案和厚度的线设置信息。一个有效线型的表包含在第三章的预定义线图案的节中。请注意：仅当`linestyle`字段包含`USERBIT_LINE`时，使用`lineinfo`结构的`upattern`字段。

```
void far getlinesettings(struct linesettingstype far *lineinfo);
```

`lineinfo` 使用称为`linesettingstype`的预定义数据类型，以便返

画线设置数据。此数据类型定义为：

```
struct linesettingstype {  
    int linestyle;           // line style's code  
    unsigned upattern;       // Pattern which defines line style  
    int thickness;          // Thickness of line  
};
```

getmaxcolor() 返回现用彩色调色板中的最大颜色值。

```
int far getmaxcolor(void);
```

getmaxmode() 返回当前驱动程序最高分辨率模式的整型码。

```
int far getmaxmode(void);
```

getmaxx() 返回现用图形适配器和模式的最大水平坐标。

```
int far getmaxx(void);
```

getmaxy() 返回现在图形适配器和模式的最大垂直坐标。

```
int far getmaxy(void);
```

getmodename() 返回现用图形模式的名字。此名字作为一串字符返回。

```
char *far getmodename(int modenumber);
```

modenumber 指定图形模式的整型码。

getmoderange() 返回给定驱动程序的图形模式的范围。

```
void far getmoderange(int graphdriver, int far *lomode,  
                      int far *himode);
```

graphdriver 参引驱动程序的整型码。

lomode, himode 低模式和高模式。

getpalette() 返回有关现用彩色调色板的信息，包括它的大小和颜色。

`void far getpalette(struct palettetype far *palette):`

`palette` 使用称为`palettetype`的预定义数据类型存放调色板的信息。此数据类型定义为:

```
#define MAXCOLORS 15
struct palettetype {
    unsigned char size;           // Size of palette
    signed char colors[MAXCOLORS + 1]; // Color entries of palette
};
```

`getpixel()` 取指定位置的像素颜色。

`int far getpixel(int x, int y):`

`x, y` 像素的水平位置和垂直位置。

`gettextsettings()` 获得有关当前正文设置的信息。

`void far gettextsettings(struct textsettingstype far *textinfo):`

`textinfo` 使用预定义数据类型`textsettingstype`返回当前的正文设置。此数据类型定义为:

```
struct textsettingstype {
    int font;           // Code indicating font style
    int direction;      // Direction of text (horizontal or vertical)
    int charsize;       // Size of the active font
    int horiz;          // Horizontal text justification style
    int vert;           // Vertical text justification style
};
```

`getviewsettings()` 取当前观察端口的信息。当用`initgraph()`函数初始化图形系统时, 根据缺省, 观察窗口被置为全屏幕的尺寸。

`void far getviewsettings (struct viewporttype far *viewport) ,`

`viewport` 送回观察端口设置的预定义数据类型。此数据结构定义为:

```
struct viewporttype {
    int left, top, right, bottom; // Pixel boundaries of viewport
    int clipflag;                 // Graphics are clipped if nonzero
};
```

`getx ()` 取当前的水平坐标。

`int far getx(void):`

gety()	取当前的垂直坐标。
---------	-----------

```
int far gety(void):
```

`graphdefaults()` 重置图形属性为它们的缺省设置。表A.1中列出所有的缺省设置。

```
void far graphdefaults(void);
```

grapherrormsg() 针对graphresult() 返回的错误码送回一个错误信息字符串。

```
char *far grapherrormsg(int errorcode):
```

errorcode 由graphresult() 返回的整型错误码。

`-graphfreemem` 释放由BGI分配的存储。当关闭图形系统时,由BGI内部地调用此函数。

```
void far _graphfreemem(void far *ptr, unsigned size):
```

ptr 指向要释放的存储区的指针。

size 存储区的大小。

_graphgetmem() 为BGI分配存储。当初始化图形系统时由BGI内部地调用此函数。

```
void far * _graphgetmem(unsigned size):
```

size 要分配的存储块的大小。

graphresult() 为上一次失效的图形调用返回错误码。

```
int far graphresult(void):
```

magesize()	获得为恢复屏幕映像所需的字节数。如果该映像大小大于64K, 则返回-1。
--------------------	--------------------------------------

```
unsigned far imagesize(int left, int top, int right, int bottom):
```

left, top 左上角。

right, bottom 右下角。

表 A.1 缺省的图形设置

图形属性	缺省设置
当前位置 (cp)	左上角 (0, 0)
观察端口	全屏尺寸 (0, 0, 宽-1, 高-1)
调色板	针对当前适配器的缺省值
绘图颜色	最大的颜色
背景颜色	调色板中的颜色0
线型	实线
填充颜色	最大的颜色
填充类型	实心填充
字形尺寸	1
字形方向	水平
字形版面调整	左和上

initgraph() 通过装入指定的图形驱动程序和设置图形模式初始化图形系统。在使用任一BGI绘图函数之前,必须调用 **initgraph()** 函数。

```
void far initgraph(int far *graphdriver, int far *graphmode,
                  char far *pathtodriver);
```

graphdriver	提供一整型码以说明应使用哪一个图形驱动程序。
graphmode	提供一整型码以说明图形模式。
pathtodriver	存放图形驱动程序文件处的目录路径名。

installuserdriver() 安装新的BGI设备驱动程序。

```
int far installuserdriver(char far *drivername,
                          int huge (*autodetectptr)(void));
```

drivername	图形设备驱动程序 (BGI) 文件的DOS名。这个名字可以包含全目录路径。
autodetectptr	指向分配给新的驱动程序的任选自动检测函数的指针。

installuserfont() 装入用户提供的字形文件。

```
int far installuserfont(char far *fontname);
```

fontname	笔画字形文件的DOS名。这个名字可包含全目录路径。
-----------------	---------------------------

iline() 用绝对屏幕坐标在两点之间画线。用当前颜色和线型画线，为改变线的设置，可在画线之前调用setlinestyle() 函数。

```
void far iline(int x0, int y0, int x1, int y1);
```

x0, y0 起始端点。
x1, y1 结束端点。

linere() 使用相对坐标把一条线从当前位置 (cp) 画到新的位置上。在画线以后，通过把偏移 (dx, dy) 添加到先前的cp上以确定新的cp。

```
void far linere(int dx, int dy);
```

dx, dy 用于画线的相对水平距离和垂直距离。这些值的任一个都可以说明为正数或负数。

lineto() 把一条线从当前位置 (cp) 画到用绝对坐标说明的新位置1。

```
void far lineto(int x, int y);
```

x, y 线的端点。在画线以后该位置变成新的当前位置(cp)。

moverel() 在有效的观察端口中，把当前位置 (cp) 移离cp的一段相对距离。

```
void far moverel(int dx, int dy);
```

dx, dy 移动cp的偏移。

moveto() 把当前位置 (cp) 移到新的位置。

```
void far moveto(int x, int y);
```

x, y 新的cp。

outtext() 在有效的观察端口的当前位置显示一个正文字符串。这个正文字符串是以当前尺寸、字形、方向和版面调整设置显示的。

```
void far outtext(char far *textstring);
```

textstring 在当前观察端口显示该字符串。

outtextxy() 在指定位置显示一正文字符串。

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

x, y	显示正文处的绝对坐标位置。
textstring	显示正文字符串。

pieslice() 用当前的绘图颜色和图案填充类型画和填写一饼片。

```
void far pieslice(int x, int y, int stangle, int endangle, int radius);
```

x, y	饼片的中心点。
stangle, endangle	按度计算的起始角和终止角。
radius	按像素计算的半径。

putimage() 把位映像拷贝到屏幕上。这个函数和getimage() 对形成动画是有用的。

```
void far putimage(int left, int top, void far *bitmap, int op);
```

putpixel() 以指定颜色显示一个像素。

```
Void far putpixel(int x, int y, int pixelcolor);
```

x, y	像素的位置。
pixelcolor	说明为整型值的颜色码。

rectangle() 以当前线型和颜色画矩形。

```
void far rectangle(int left, int top, int right, int bottom);
```

left, top	左上角。
right, bottom	右下角。

registerbgidriver() 注册一个BGI图形驱动程序。驱动程序可以从磁盘装入或作为一个.OBJ文件被连接。

```
int registerbgidriver(void (*driver)(void));
```

driver	注册的驱动程序名。
--------	-----------

registerbgifont() 注册一个BGI字形驱动程序。

```
int registerbgifont(void (*font)(void));
```

<code>font</code>	注册的字形名。
<code>restorecrtmode()</code>	把屏幕恢复为选择图形模式之前的模式。
<code>void far restorecrtmode(void);</code>	
<code>sector()</code>	用当前绘图颜色和图案填充类型画和填写椭圆形的饼片。
<code>void far sector(int x, int y, int stangle, int endangle, int xradius, int yradius);</code>	
<code>x, y</code>	饼片的中心点。
<code>stangle, endangle</code>	按度计算的起始角和终止角。
<code>xradius, yradius</code>	按像素计算的水平半径和垂直半径。
<code>setactivepage()</code>	为图形显示设置活动页。
<code>void setactivepage(int pagenum);</code>	
<code>pagenum</code>	需要活动的页码。
<code>setallpalette()</code>	改变所有调色板的颜色。
<code>void far setallpalette(struct palettetype far *palette);</code>	
<code>palette</code>	包含调色板的颜色信息, 这个数据结构由 <code>getpalette()</code> 函数提供。
<code>setaspectratio()</code>	为现用的图形硬件和模式设置x和y的长宽比。
<code>void far setaspectratio(int xasp, int yasp);</code>	
<code>xasp, yasp</code>	新的水平和垂直长宽比。
<code>setbkcolor()</code>	置背景颜色。
<code>void far setbkcolor(int color);</code>	
<code>color</code>	指定新的背景颜色的整型颜色码 (0~15)。表3.4提供了背景颜色和代码。
<code>setcolor()</code>	置现用的绘图颜色。通过调用 <code>getmaxcolor()</code> 可得到给定的图形适配器和模式可用的最大颜色数目。

<code>void far setcolor(int color);</code>	
color	指定新的绘图颜色的整形颜色码。表3.4提供了背景颜色和代码。
<code>setfillpattern()</code>	
	选择用户定义的填充图案和颜色，使用位设置建立用户定义的填充图案。表3.7描述了定义图案的技术。
<code>void far setfillpattern(char far *pattern, int color);</code>	
pattern	指定用户定义的填充图案。
color	指定填充颜色。
<code>setfillstyle()</code>	
	选择当前填充图案和颜色。用这个例程设置的填充图案用于绘制图形，它在下列函数中使用： <code>bar()</code> ， <code>bar3d()</code> ， <code>fillpoly()</code> ， <code>floodfill()</code> ， <code>pieslice()</code> ， <code>sector()</code> 。
<code>void far setfillstyle(int pattern, int color);</code>	
pattern	指定12种BGI支持的填充图案之一。
color	指定填充颜色。
<code>setgraphbufsize()</code>	
	设置像 <code>floodfill()</code> 那样的BGI例程使用的内部图形缓冲区的大小，缺省的大小为4K。如果你使用这个函数，则必须在调用 <code>initgraph()</code> 之前调用它。它返回先前定义的缓冲区字节数。
<code>unsigned far setgraphbufsize(unsigned bufsize);</code>	
bufsize	图形缓冲区所用的字节数。
<code>setgraphmode()</code>	
	置系统为图形模式并清屏。为使用这函数，必须已在前面调用了 <code>initgraph()</code> 。
<code>void far setgraphmode(int mode);</code>	
mode	选择的图形模式。
<code>setlinestyle()</code>	
	置当前的线宽和类型。
<code>void far setlinestyle(int linestyle, unsigned upattern, int thickness);</code>	
linestyle	说明为整型码的线型（见表A.2）

upattern 说明为16位模式的用户线型图案。
thickness 线的厚度。支持两种型式: NORM_WIDTH = 1像素宽; THICK_WIDTH = 3像素宽。

setpalette() 改变现用彩色调色板中的颜色。

```
void far setpalette(int index, int color);
```

index 调色板索引 (0~15)。
color 替换的颜色。

表 A.2 线型码

值	名 字
0	SOLID_LINE
1	DOTTED_LINE
2	CENTER_LINE
3	DASHED_LINE
4	USERBIT_LINE

setrgbpalette() 改变现用颜色调色板的颜色。

```
void far setrgbpalette(int colornum, int red, int green, int blue);
```

colornum 装入的调色板项。
red, green, blue 颜色成分。

settextjustify() 置正文版面调度类型。表A.3列出版面调整码。

```
void far settextjustify(int horiz, int vert);
```

horiz 水平版面调整码。
vert 垂直版面调整码。

settextstyle() 定义BGI使用的正文设置。

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

font 使用的字形类型可以是表A.4列出的任一值。
direction 当显示正文时使用的字形方向, 它可以是水平正文的HORIZ_DIR (值 = 0) 或垂直正文的VERT_DIR (值 = 1)
charsize 放大所显示字符的量值, 可在0和10之间。0值仅能用



于笔画字形和指示BGI使用setusercharsize() 说明的字符尺寸或缺省的放大率 (4)。

setusercharsize() 定义所显示的笔画字形字符的大小。此尺寸用一个乘法因子说明。

void far setusercharsize(int multx, int divx, int multy, int divy);

multx, multy 指定乘法因子的高和宽。
divx, divy 指定除法因子的高和宽。

表 A.3 正文版面调整码

宏名字	值	描 述
LEFT_TEXT	0	左对齐正文
CENTER_TEXT	1	中央正文 (水平或垂直正文)
RIGHT_TEXT	2	右对齐正文
BOTTOM_TEXT	0	从底对齐正文
TOP_TEXT	2	从顶对齐正文

表 A.4 字形类型

名 字	值	描 述
DEFAULT_FONT	0	8×8位映像字形
TRIPLEX_FONT	1	三元组笔画字符
SMALL_FONT	2	小型笔画字形
SAN_SERIF_FONT	3	无衬线笔画字形
GOTHIC_FONT	4	粗黑体笔画字形

setviewport() 置当前的观察端口。

void far setviewport(int left, int top, int right, int bottom,
int clipflag);

left, top 观察端口的左上角。
right, bottom 观察端口的右上角。
clipflag 确定图形是否已被裁剪到现用的观察端口中。如果这个参数为非0, 则图形已被裁剪。

setvisualpage() 置可视的图形页号。

void far setvisualpage(int pagenum);



说明现用可视页面的整形码。

setwritemode()

为画线图形例程设置书写模式。这个例程设置的书写模式仅影响下列绘图函数：`drawpoly()`，`line()`，`linereel()`，`lineto()`，`rectangle()`。

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

writemode

指定书写模式，支持的模式是：`COPY_PUT`（值=0）和`XOR_PUT`（值=1）。使用`COPY_PUT`模式把所画的新线重写当前屏幕上的东西。使用`XOR_PUT`执行“异或”操作。

textheight()

返回按像素计算的正文字符串的高度。用当前（活动）的字形尺寸和乘法因子计算字符串的高度。

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

textstring

用以改变尺寸的字符串。

textwidth()

返回按像素计算的正文字符串的宽度。用当前（活动）的字形尺寸和乘法因子计算字符串的宽度。

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

textstring

正文字符串。