第一章　关于视频图形

计算机界曾流行着这样一种说法：Pascal 不是一种面向图形的语言，也无法支持图形应用。也许，在面向图形的计算机问世之前，这种说法是正确的（除了早期的苹果机，它虽面向图形，但不面向应用）。

然而，这种说法已被今日的现实所否定。目前，以图形功能较弱的 CGA 系统为基础，视频系统的功能得到了极大的扩充。早期的计算机系统仅有 2MHz 主频，16K 内存，相对于图形应用所必备的高速度、大内存，的确显得力不从心。但是，令人欣慰的是：它已被今天主频在"20～30MHz 以上，内存 1M 以上的计算机系统所取代。

同时，图形接口及其应用也变得十分普及，如 Microsoft Windows，OS/2，以及 Ventura Publisher。

但最重要的是：Turbo Pascal 不仅是一个面向图形的编辑器，同时它还借助于 Borland 图形接口，成为一种图形处理能力极强的语言。此外，它还提供了支持图形应用的子程序和函数，并为图形应用的实现提供了尽可能高的速度。

图形已不再是为传统的程序所精心设计的装饰物，也不仅仅是为游戏提供彩色和图像的工具，相反，图形已成为一种标准，而且未来的许多应用程序都将以此作为一种评价标准，这种现象决不会是花一现，即使是那些缺乏彩色处理能力的便携式计算机，也在极力追求高分辨率的图形显示功能。例如有一种便携机具有 VGA 视频模式和一个液晶显示屏，能模拟 16 个灰阶的色彩，清晰度极高，而总重量只有 14 磅。这种便携机可以做插图和排版工作，如果没有非凡的图形处理能力，完成这些工作是难以想象的。

既然便携机都已具备了很强的图形处理能力，难道计算机图形系统还会止步不前吗？

1.1　为什么使用图形显示

因为程序的外观是个重要因素，特别是在颇具竞争性的商业环境中，图形显示的优越性远不止是单纯的装饰物。

今天，图形用户接口 GUI 已被许多系统和应用程序作为一个通用的标准。这些系统和应用程序可使用图形屏幕菜单和鼠标进行选择，不必仅依靠键盘来控制。通常，GUI 的选项不需要进一步地解释或说明就能够加以识别，而且 GUI 还提供了比通常 C）提示符更加通向用户的接口。但在某些情况下，比如在 Macintosh 中，由于过分使用了面向图符（ICON）的设计，使接口的设计走向极端，给用户带来了许多麻烦。

除去极端情况，图形和文本的均衡使用会给用户提供简单易懂的信息。例如：卷滚条既可以在下拉菜单中标明位置，并在文本中定位，又可以用鼠标来控制显示位置的改变，以及提供鼠标按键控制。因此，卷滚条是一种操作简单明了，又非常奇妙的图形元素。

同样，在事务处理中，图形就比一列列数据更简单、易于理解和比较。而在数学应用中，一个优美的三维图形，将会使抽象的研究对象变得更加直观，而枯燥的数字和符号对此则望尘莫及。对于排版软件来说，任何东西都无法替代图形，时下流行的 WordPerfect 5.0 就是最好的证明，该软件既支持标准的文本显示，又支持整页的图形（包括图像）输出。另一方面，为了完成任何一件，哪怕是很小的一点事情，WordPerfect 5.0 都需要在屏幕上拖着精致的"文件夹"和"垃

级简"移动，这不禁会惹人烦恼。显然，上述二方面又需达到个平衡。

然而，当你为使用哪种软件而犹豫不定时，头等重要的还是如何完成图形显示功能。

1.1.1 图形显示

现有的各种视频硬件(其中至少有 10 个标准的视频适配器)已经向程序员提出了挑战。每种视频硬件都各支持一个功能集，它们可能使用不同的内存地址，支持或不支持多个图形页面(且/或多个文本页面)，并且，所支持的分辨率从 320×200 到 1024×768 不等，所支持的色彩也由单色到 256 种颜色各异。

编制图形应用程序时，程序员首先要弄清的问题是主机的型号以及其上要配置哪种视频适配器。一种常用的方法是询问最终用户，允许他们选择自己想用的视频适配器。但这种方法存在着不足之处，因为即使是职业程序员也不会对他要做开发工作的所有硬件有十足的把握，面对普适的最终用户来说，他们甚至不知道要给系统配置图形适配器，更何况让他们自己确定适配器的型号了。因此，还有一种方法就是在编程时，向硬件发出询问来判定所使用的硬件配置。

假如硬件标志有标准可循，上述方法就会变得十分简单。遗憾的是，尽管视频适配器硬件的种类在迅速增长，但对硬件标志尚未提供任何的形式化描述。

值得高兴的是，Turbo Pascal 4.0V(和 Turbo C 1.5V)的发表，使对现有的视频适配卡的鉴别、支持和应用更为成熟，而且根据 Borland 公司过去的能力，我们完全可以相信，以后新推出的视频适配器也将得到充分的支持。

然而，不幸的是，对各种视频性能的支持也带来了一系列问题：如何使用这些新工具，对这些日益增多的新性能，我们做些什么，可能做些什么？

这些正是本书要重点讨论的问题。

1.2 视频适配器类型

对现有的各种主要视频卡类型，Turbo Pascal 4.0V 或更高版本，以及 Turbo C 1.5V 或更高版本，都提供了全面支持。但本书仅阐述 Turbo Pascal 的图形。如果你正在使用 Turbo C，那么它也据供了相应的函数，你可以使用与 Turbo Pascal 名字和功能基本相同的函数和子程序，来完成同样有效的操作。

目前，从最初的纯字符视频系统到超高分辨率的 IBM－8514，以及介于两者之间的具有各种高低不同分辨率的视频系统都同时存在着。较高分辨率的视频卡必须与相应分辨率的监视器相匹配，但这只是个硬件问题。对程序员而言，他所关心的只是所使用的硬件必须与调用 Turbo Pascal 的 DetectGraph 函数所返回的标识相一致。视频卡和监视器之间的不匹配应由最终用户去解决，而不应是程序员所关心的问题。

目前，绝大多数多同步高分辨率的图形板都提供了一些识别现有的监视器类型的方法。

1.3 视频模式

不论 PC 机还是 XT 或 AT 机都配有各种类型的视频卡。最低档的视频卡是单色显示适配器(MDA)，它仅支持纯字符显示方式，如果在其上想利用图形功能开发软件，就必须对系统升级。

稍高档些的是使用比较广泛的彩色图形适配器(CGA)，而大力神单色图形适配器、多色图形阵列(MCGA)，以及增强型图形适配器(EGA)将会提供较高的分辨率，并使用户有更广的选

择余地。为了适应排版或CAD等应用中对图形性能的更高要求，相继出现了具有更高分辨率的产品，如400线图形适配器、视频图形适配器VGA(又可称多同步视频适配器)、PC—3270、IBM—8514等视频适配器。

Turbo Pascal支持上述所有类型的适配器，这种支持包括六种图形接口文件.BGI(ATT、CGA、EGAVGA、HERC、IBM8514、PC3270)和四种图形字形(GOTH.CHR、LITT.CHR、SANS.CHR、TRIP.CHR)。随着新的视频图形卡的出现，新的.BGI文件也会不断涌现出来，同时用户可以自己生成或从制造商那里获得新的.CHR字形。(参见第十七章)。

然而这些图形支持文件在链接时并不存放在基本内存中，如果没有图形需求，就可以忽略这些文件，以提高编译速度。

1.4 视频图形的不足

一个源程序经过编译后，它的.EXE目标程序将链接上运行所需的外部.BGI和.CHR文件，这些文件共需大约60K磁盘空间。就算在存贮上没有过多的要求，但在运行时依赖于外部文件也会带来许多麻烦。

首先，对函数InitGraph调用时，必须包括对.BGI和.CHR文件所在驱动器和路径的说明，如果未说明路径，则会认定在当前目录下，这种路径信息一般是由程序员提供的。如果没有找到这些需要的文件(因为不论在程序设计时怎样精心，总会出这样那样问题)，程序就会终止运行。

第二，如果缺省为当前路径，并找到了上述外部文件，但系统又调用了另一个目录或路径下的程序，则系统也会崩溃。

第三，千万不能指望最终用户去重视外部文件。他们可能很喜欢你的程序，并且用起来也很谨慎，但是一旦空间不够用时，他们就会删掉那些必需的外部文件。另一种方法是，将.BGI和.CHR文件直接链接到图形库上，这会使.EXE文件增加大约30K(参见第九章，改进的图形管理函数)。这样，驱动程序和字形就不再是外部文件，而成为.EXE文件的一个组成部分。

另外，如果由于链接这些文件而造成了程序过于庞大，你可以给这些外部文件注上"只读"、"系统"、"隐含"等标记。

1.5 外部文件的使用方法

Turbo Pascal为支持特殊应用而提供了辅助函数库，该库由以下文件组成：DOS，GRAPH，GRAPH3，OVERLAY，PRINTER，SYSTEM和TURBO3。用户自定义的文件也可以提供新函数和子程序，它们可以像配置文件那样方便地使用。

我们以GRAPH为例，来介绍外部文件的用法。GRAPH是本书中所有程序的基础，GRAPH的使用非常简单，只需在程序的首部做如下说明：

```
USES    GRAPH;
```

当需要调用几个文件中的函数时，这多个文件可以在一行中说明如下：

```
USES  CRT,  DOS,  GRAPH,  TURTLE;
```

这些文件以什么次序调用是无关紧要的，而且程序中可以包含任何多个这种文件。上述例子中还定义了一个用户自定义模块TURTLE，本书将给出它的定义。

1.5.1 包含目录的选项

如果Turbo Pascal的根目录下未包括辅助函数库中的文件，则Turbo的集成开发环境中提

供了为 Turbo.EXE 和 TPU,Include.Unit,和 Object 文件设置路径的功能(参见《Turbo Pascal 用户指南》中的"目录选项")。为了确定路径,可先从菜单中选择 Option,Directory 和 Unit 的入口,再输入那些. TPU 文件所在路径的完整说明,可以用分号把每个路径隔开,来指定多个路径,参见如下例子:

\TP\UNITS;\TP\EXE;\TP\OBJECT

这个例子说明了文件可能存在的三个子目录:\TP\UNITS 目录,\TP\EXE 目录(这里将产生一个新的. TPU 文件,因为这个子目录是在 EXE 选项下说明的)和\TP\OBJECT 目录,默认值是 Turbo Pascal 的根目录\TP。

1.6 图形设备的初始化

编制图形程序时,第一步要初始化相应的图形驱动器。表 1-1 给出了 Turbo Pascal 所支持的图形视频卡、驱动器和图形模式。

表 1-1 Turbo Pascal 所支持的视频模式

图形驱动器①	值	图形模式	关键值②	列数×行数	调色板或颜色③	视频页面
DETECT	0	requests InitGraph to execute autodetection				
CGA	1	CGAC0	0	320×200	C0	1
		CGAC1	1	320×200	C1	1
		CGAC2	2	320×200	C2	1
		CGAC3	3	320×200	C3	1
		CGACI	4	640×200	2 colors	1
MCGA	2	MCGAC0	0	320×200	C0	1
		MCGAC1	1	320×200	C1	1
		MCGAC2	2	320×200	C2	1
		MCGAC3	3	320×200	C3	1
		MCGAMED	4	640×200	2 colors	1
		MCGAHI	5	640×480	2 colors	1
EGA	3	EGALO	0	640×200	16 colors	4
		EGAHI	1	640×350	16 colors	2
EGA64	4	EGA64LO	0	640×200	16 colors	1
		EGA64HI	1	640×350	4 colors	1
EGAMONO	5	EGAMONOHI	3	640×350	2 colors	1-2④
IBM8514⑤	6	IBM8514LO	0	640×480	256 colors	1
		IBM8514HI	1	1024×768	256 colors	1
HERC	7	HERCMONOHI	0	720×348	2 colors	4
ATT400	8	ATT400C0	0	320×200	C0	1
		ATT400C1	1	320×200	C1	1
		ATT400C2	2	320×200	C2	1
		ATT400C3	3	320×200	C3	1

表 1-1 Turbo Pascal 所支持的视频模式

		ATT400MED	4	640×200	2 colors	1
		ATT400HI	5	640×200	2 colors	1
VGA	9	VGALO	0	640×200	16 colors	4
		VGAMED	1	640×350	16 colors	2
		VGAHI	2	640×480	16 colors	1
PC3270	10	PC3270HI	0	720×350	2 colors	1

注：

① 图形驱动器(Graphic Diver)和图形模式(Graphic Mode)在文件 GRAPHICS. H 中定义为常量，与之相对应的值也是在该文件中定义的。

② 函数 InitGraph，DetectGraph 或 GetGraphMode 的返回值可用来设置模式。

③ C0…C3 是指预定义的 4 个调色板(参见 SetPalette)。

④ 64K 的 EGAMONO 卡仅支持一个视频页面，而 256K 的卡则支持 2 个视频页面。

⑤ 自动检测无法识别 IBM-8514 图形卡，而 InitGraph 或 DetectGraph 把 IBM-8514 认作一个 VGA 的图形卡。因为 IBM-8514 可以仿真 VGA 卡的功能(IBM8514LO 等同于 VGAHI)，要想使用较高的分辨率模式(IBM8514HI，1024×768)，在调用 InitGraph 前，必须把 IBM-8514 的值(在 GRAPHICS. H 中定义为 6)赋给 graphdriver 这个变量，这时不能调用 DetectGraph 或者在 InitGraph 中避免使用 DETECT。参见 IBM－8514 和 SetRGBPalette 的文字注解。

1.6.1 DetectGraph

一般情况，函数 DetectGraph 是被 InitGraph 调用的，但它还可以单独调用来确定当前的图形驱动器和图形模式。

【例】
```
uses GRAPH;
var
    GraphDriver,GraphMode:integer;
begin
    GraphDriver:=DETECT;
    DetectGraph(GraphDriver,GraphMode);
    ...
end;
```

如果 GraphDriver 出错，它将返回一个错误码，否则 GraphDriver 将识别正确驱动器类型，并由 GraphMode 为此驱动器返回最有效的视频模式。

函数 DetectGraph 不对任何的图形设置进行初始化，直接调用它的基本原因在于，紧接着可使用 InitGraph 来调用一个特定的图形驱动器或选择一个图形模式，该图形模式在 InitGraph 调用中不可缺省。另外，使用 SetGraphMode 函数进行初始化后，才能调用各种不同模式。

返回值：GraphDriver 返回值是一个驱动器类型或错误码；GraphMode 返回值是最有效的视频模式。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中也有相应的函数。

1.6.2 InitGraph

在 Turbo Pascal 中，函数 InitGraph 用来为图形参数置初值，加载有效的图形驱动器，并将系统设置为所期望的图形模式。

【例】
```
uses GRAPH;
```

```
const
    DriverPath='';
var
    GraphDriver,GraphMode:integer;
begin
    GraphDriver:=DETECT;
    InitGraph(GraphDriver,GraphMode,DriverPath);
...
end
```

设置 GraphDriver 为 0(DETECT),将指示 InitGraph 去调用 DetectGraph(GraphDriver, GraphMode),来确定视频图形适配器的类型和设置.如果出错,则变量 GraphDriver 将返回一个错误码来指明错误类形.表 1-2 列出了图形错误码.

表 1-2:初始化图形错误码

代码	错误信息
−2	无法检测图形卡
−3	无法定位图形驱动文件
−4	无效的驱动器(或不能识别驱动器)
−5	装载图形驱动程序时,内存溢出

函数 DetectGraph 和 GraphResults 返回的错误码相同,都如表 1-2 所示。

如果没有发生错误,则内部错误码被置成 0。InitGraph 为适当的图形驱动程序分配内存空间,加载所需要的磁盘.BGI 文件,并设置缺省的图形参数值。同样,GraphDriver 返回驱动器类型,GraphMode 返回模式的值。

GraphDriver 和 GraphMode 也可以被设定(既可用数值常量设定,又可以使用驱动器名和模式名在 GRAPH.TPU 文件中设定)。两种情况下,DriverPath 都表示.BGI 图形驱动程序所在的盘驱动器和路径。如果 DriverPath 为空,则这些文件要在缺省目录下找;如果它们放于不同的目录,则可作如下完整的路径说明:

```
DriverPath='\TP\BGI';
```

由于 DriverPath 还要在 SetTextStyle 中使用(用于搜索字形文件.CHR),所以.BGI 和.CHR 文件必须放在同一目录下。

返回值:GraphDriver 返回一个驱动器类型或错误码;GraphMode 返回最有效的视频模式。

可移值性:只适于 IBM PCS 及其兼容机,Turbo C 中也有相应的函数。

1.6.3 GetDriverName

函数 GetDriverName 返回一个表示当前图形驱动器名字的字符串。

【例】

```
uses GRAPH;
OutTextXY(50,50,'Using Driver'+GetDriverName);
```

只有在 InitGraph 设置了图形驱动器的模式之后,GetDriverName 才可以使用。参见 GetModeName 有关信息。

可移植性:只适于 IBM PCS 及其兼容机,Turbo C 中也有相应的函数。

1.7 图形错误处理函数

如上所述，在进行初始化检测时，如果图形视频卡不存在，或没有找到图形驱动程序，或者发生其它错误，则 DetectGraph 和 InitGraph 都会返回一个错误码。但是，其它情况下也会发生图形错误，函数 GraphResult 和 GraphErrorMsg 是用来检查和显示错误结果和信息的。

1.7.1 GraphResult

函数 GraphResult 返回一个数值型错误码，它是由最近的一次图形操作报告的，其值是 -15…0 之间的整数。由于调用 GraphResult 时，出错条件被复位为 0，所以返回值应存放在局部变量中，以便于将来测试用。

【例】

```
uses GRAPH;
var
ErrorNumber;
ErrorNumber:=GraphResult;
```

返回值：GraphResult 返回值是一个错误码(-15…0)，参见表 1-3 的解释。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中有相应的函数。

表 1-3 图形错误信息

错误码	图形一错误常量	错误信息串
0	grOk	No error
-1	grNoInitGraph	BGI graphics not installed(use InitGraph)
-2	grNotDetected	Graphics hardware not detected
-3	grFileNotFound	Device driver file
-4	grInvalidDriver	Invalid device driver file
-5	grNoLoadMem	Not enough memory to load driver
-6	grNoScanMem	Out of memory in scan fill
-7	grNoFloodMem	Out of memory in flood fill
-8	grFontNotFound	Font file not found(.CHR file)
-9	grNoFontMem	Not enoughmemory to load font
-10	grInvalidMode	Invalid graphics mode for selected driver
-11	grError	Graphics error(generic error)
-12	grIOerror	Graphics I/O error
-13	grInvalidFont	Invalid font file
-14	grInvalidFontNum	Invalid font number
-15	grInvalidDeviceNum	Invalid device number

常量 GraphicsErrors 和错误信息是在文件 GRAPH unit 中定义的。

1.7.2 GraphErrorMsg

函数 GraphErrorMsg 返回一个指向错误信息串的指针，这些串是在图形库(GRAPH1CS.LIB)中定义的。你可以按自己的意愿来定义一个出错信息子程序，显示更完整的出错信息。

【例】

```
uses GRAPH;
```

```
var
ErrorNumber;
ErrorNumber:=GraphResult;
writeln(GraphErrorMsg(ErrorNumber));
```

返回值:无。

可移植性:只适于 IBM PCS 及其兼容机,在 Turbo C 中也有相应的函数。

1.8 其它图形模式函数

对于 EGAMONO,HERC 和 PC3270 等视频驱动器,它们各自都支持两种或更多的视频模式,这些模式提供了各种像素分辨率,或调色板。为了处理模式查询、改变操作模式,Trubo Pascal 提供了如下几个函数。

1.8.1 GetGraphMode

函数 GetGraphMode 返回一个整数,来表示当前(操作)的模式,该模式由 InitGraph 和 SetGraphMode 设置。

【例】
```
uses GRAPH;
var
CurrentMode:integer;
CurrentMode:=GetGraphMode;
```

返回值:当前图形模式。

可移植性:只适于 IBM PCS 及其兼容机,Turbo C 中也有相应的函数。

1.8.2 GetModeRange

函数 GetModeRange 被调用时,带一个表示图形驱动器的整型参数(它可以是一个整型变量,或是 GRAPH 文件所定义的一个常量),并返回该驱动器所支持的最大、最小模式。

【例】
```
uses GRAPH;
var
LoMode,HiMode:integer;
GetModeRange(GraphDriver,LoMode,HiMode);
```

如果 GraphDriver 进行参数传递时发生错误,则 LoMode 和 HiMode 都返回 -1。

返回值:最大、最小有效模式或错误码 -1。

可移植性:只适于 IBM PCS 及其兼容机,在 Turbo C 中也有相应的函数。

1.8.3 GetMaxMode

函数 GetMaxMode 返回当前加载驱动器的最大模式值,返回值直接来自于驱动器,它还支持函数 GetModeRange,但 GetModeRange 仅对 BGI(Borland-supplied)驱动器有效。

【例】
```
uses GRAPH;
var
MaxMode:integer;
MaxMode:=GetMaxMode;
```

所有驱动器都支持模式 0…GetMaxMode,该函数的返回值是可传给过程 SetGraphMode 的最大值。

可移植性,只适于 IBM PCS 及其兼容机,Turbo C 中也有相应的函数。

1.8.4 GetModeName

函数 GetModeName 调用时带一个模式参数，返回值是当前图形驱动器的模式名。

【例】
```
uses GRAPH;
var
i==integer;
OutTextXY(50,50,'Valid modes are:');
for i:=0 to GetMaxMode do
OutTextXY(60,65+10*i,GetModeName(i));
```

模式名被嵌在每个图形驱动程序中，并可以作为菜单使用，来显示状态或报告系统性能。

可移值性：只适于 IBM PCS 及其兼容机，Turbo C 中也有相应的函数。

1.8.5 GraphDefaults

函数 GraphDefaults 将所有的图形设置参数复位成其默认值（初值是由 InitGraph 设置或在 GRAPH 文件中定义的），这包括将视口（图形窗口）复位成全屏幕；将 CP 移到(0,0)；置调色板颜色、背景颜色和作图颜色为默认值；将填充类型和模式类型置为缺省值；复位文本字形和调整方式。

【例】
```
uses GRAPH;
GraphDefaults;
```

返回值：无。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中也有相应的函数。

1.8.6 SetGraphMode

图形的模式必须事先由 InitGraph 进行初始化。函数 SetGraphMode 被调用时，必须带一个符合当前驱动器的图形模式参数（可使用 GetGraphMode 来获得当前模式的值，或利用 GetModeRange 来检测值域）。当调用该函数时，SetGraphMode 将选择一个新的图形模式，并清屏，将所有图形变量复位为默认值（参见 GraphDefaults）。

【例】
```
uses GRAPH;
var
ModeNumber:integer;
SetGraphMode(ModeNumber);
```

函数 SetGraphMode 还可以和 RestoreCrt 一起使用，可进行文本和图形两种显示方式的切换。使用这两个函数之前，必须先调用函数 InitGraph。

【例】
```
uses GRAPH;
var
CurrentMode:integer;
CurrentMode:=GetGraphMode;
RestoreCrtMode; /* 文本模式 */
SetGraphMode(CurrentMode) /* 图形模式 */
```

如果调用 SetGraphMode 时，其形式参数值与当前设备驱动器不符合，则 GraphResult 将返回值－10(grInvalidMode)。

返回值：无。参见 GraphResult 的错误码。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中也有相应的函数。

1.8.7 RestoreCrtMode

函数 RestoreCrtMode 将系统视频模式复位成最初由 InitGraph 检测到的文本方式。它与 SetGraphMode 一起，可进行文本和图形两种显示方式的切换。

【例】

```
uses GRAPH;
RestoreCrtMode;
```

返回值：无。参见 GraphResult 的错误码。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中有相应的函数。

1.8.8 CloseGraph

函数 CloseGraph 将系统恢复为最初由 InitGraph 检测到的文本方式，同时，释放图形系统的驱动程序、字形和内部缓冲区所占用的内存空间。

【例】

```
uses GRAPH;
CloseGraph;
```

如果你想对文本和图形两种方式互相切换，则可以利用函数 RestoreCrtMode 和 SetGraphMode。

返回值：无。参看 GraphResult 的错误码。

可移植性：只适于 IBM PCS 及其兼容机，Turbo C 中也有相应的函数。

```
{FIRSTGRP.PAS}
{Turbo Pascal 5.5 中初始化图形方式的演示视序}
uses GRAPH,CRT;
type
    Str10=string[10]; Str20=string[20];
const
  DriverNames : array[ 0..10 ] of Str10 =
    ( 'Detect', 'CGA', 'MCGA', 'EGA', 'EGA64',
      'EGAMono', 'IBM8514', 'HercMono', 'ATT400',
      'VGA', 'PC3270' );
  Fonts : array[ 0..4 ] of Str10 =
    ( 'Default', 'Triplex', 'Small', 'SansSerif',
      'Gothic' );
  LineStyles : array[ 0..4 ] of Str20 =
    ( 'Solid', 'Dotted', 'Center', 'Dashed',
      'User Defined' );
  FillStyles : array[ 0..11 ] of Str20 =
    ( 'Empty', 'Solid', 'Line Fill', 'Light Slash',
      'Slash', 'Back Slash', 'Light Back Slash',
      'Hatch', 'XHatch', 'Interleave', 'Wide Dot',
      'Close Dot' );
  TextDirect : array[ 0..1 ] of Str10 =
    ( 'Horizontal', 'Vertical' );
  HorizJust : array[ 0..2 ] of Str20 =
    ( 'Flush Left', 'Centered', 'Flush Right' );
  VertJust : array[ 0..2 ] of Str10 =
    ( 'Bottom', 'Centered', 'Top' );

var
```

```
    GraphDriver, {图形驱动器值}
    GraphMode, {图形模式值}
    AspectRatio, {屏幕像素的纵横比}
    MaxX,MaxY {屏幕分辨率的最大值}
    MaxColors :integer;{可用彩色的最大值}
    xasp,yasp :word; {纵横比因子}
    Palette :PaletteType;-
function N2S(Val,Digit:integer):string;
    var {将整数转换成字符串}

       Buffer : string;
    begin
       str( Val:Digit, Buffer );
       N2S := Buffer;
    end;

function R2S(Val:real;Digit,Decimal:integer):strings;
{将实数转换成字符串}
    var

       Buffer : string;
    begin
       str( Val: Digit: Decimal, Buffer );
       R2S := Buffer;
    end;


procedure TestGraphicError;
    var
        ErrorCode:integer;
    begin
        ErrorCode:GraphResult;{检查结果}
        if ErrorCode<>grOk then {如果发生错误}
        begin {复位为文本方式}
          CloseGraph; {报告错误}
          writeln(' Graphics System Error:' GraphErrorMsg(Errorlode));
          halt(1); {程序终止}
        end;
    end
procedure ChangeTextStyle(fnt,dir,chrsz:integer);
    var
        ErrorCode:integerl
    begin
        ErrorCode:GraphResult; {清除错误码}
        SetTextStyle(fnt,dir,chrsz);
```

```
        TestGraphicError;{错误检查}
    end;
procedure StatusLine(msg:string);
    var {在屏幕底部显示状态行}
      Height:integer;
    begin
      SetViewPort(0,0,MaxX,MaxY,True);
      SetColor(MaxColors-1);{首选颜色最大值}
      ChangeTextStyle( DefaultFont, HorizDir, 1 );
      SetTextJustify( CenterText, TopText );
      SetLineStyle( SolidLn, 0, NormWidth );
      SetFillStyle( EmptyFill, 0 );
      Height:=TextHeight(msg);{取得字符高度}
      Bar( 0, MaxY-(Height+4), MaxX, MaxY );
      Rectangle( 0, MaxY-(Height+4), MaxX, MaxY );
      OutTextXY( MaxX div 2, MaxY-(Height+2), msg );
      SetViewPort( 1, Height+5,
                   MaxX-1, MaxY-(Height+5), True );
    end;
procedure DrawBorder;
    var {用实线画作图边界}
        vp:ViewPortType;
    begin
        SetColor(MaxColors-1){设置作图颜色}
        SetLineStyle( SolidLn, 0, NormWidth );
        GetViewSettings( vp );
        Rectangle( 0, 0, vp.x2-vp.x1, vp.y2-vp.y1 );
    end;
procedure  ReportWindow(header:string);
    var
      Height:integer;
    begin
      clearDevice;              {清除图形屏}
      SetColor( MaxColors - 1 );
      SetViewPort( 0, 0, MaxX, MaxY, True );
      Height:=TextHeight(header);       {字符高度}
      ChangeTextStyle( DefaultFont, HorizDir, 1 );
      SetTextJustify( CenterText, TopText );
      OutTextXY( MaxX div 2, 2, header );
      SetViewPort( 0, Height+4,
                   MaxX, MaxY-(Height+4), True );
      DrawBorder;
      SetViewPort( 1, Height+5,
                   MaxX-1, MaxY-(Height+5), True );
```

```
    end;
procedure Pause;
    var
      Ch:char;
    begin
      StatusLine('Press any key…');        {向屏幕上输出提示信息}
      while KeyPressed do CH:=ReadKey;
      Ch:=ReadKey;                         {等待键入}
      ClearDevice;                         {清除屏幕}
    end;
procedure  StepDisplay(var y:integer);
    var
      Color:integer;
    begin
      inc( y, TextHeight( 'W' ) + 2 );
      Color := GetColor - 1;
      if Color = 0 then Color := GetMaxColor;
      SetColor( Color );
    end;
procedure ReportStatus;
    var                                    {报告当前的系统配置}
      ViewInfo : ViewPortType;
      LineInfo : LineSettingsType;
      FillInfo : FillSettingsType;
      TextInfo : TextSettingsType;
      Driver, Mode, Buffer : string;
      x, y, GraphHi, GraphLo : integer;
    begin
      x := 10;
      y :=  4;
      ReportWindow( 'Graphic Status Report' );
      GetViewSettings( ViewInfo );
      GetLineSettings( LineInfo );
      GetFillSettings( FillInfo );
      GetTextSettings( TextInfo );
      GetPalette( Palette );
      SetTextJustify( LeftText, TopText );
      OutTextXY( x, y, 'Graphics driver        : ' +
                       GetDriverName + '.BGI');
      StepDisplay( y );
      OutTextXY( x, y, 'Graphics device        : ( ' +
                       N2S( GraphDriver, 1 ) + ' ) ' +
                       DriverNames[GraphDriver] );
      StepDisplay( y );
      OutTextXY( x, y, 'Graphics mode          : ( ' +
                       N2S( GraphMode, 1 ) + ' ) ' +
                       GetModeName( GraphMode ) );
```

```
StepDisplay( y );
GetModeRange( GraphDriver, GraphHi, GraphLo );
OutTextXY( x, y, 'Valid mode range      : Low = ' +
                 N2S( GraphHi, 1 ) + ', High = ' +
                 N2S( GraphLo, 1 ) );
StepDisplay( y );
OutTextXY( x, y, 'Screen resolution     : ' +
                 '(  0,  0, ' +
                 N2S( GetMaxX, 2 ) + ', ' +
                 N2S( GetMaxY, 2 ) + ' )' );
StepDisplay( y );
OutTextXY( x, y, 'Current viewport      : ( ' +
                 N2S( ViewInfo.x1, 2 ) + ', ' +
                 N2S( ViewInfo.y1, 2 ) + ', ' +
                 N2S( ViewInfo.x2, 2 ) + ', ' +
                 N2S( ViewInfo.y2, 2 ) + ' )' );
StepDisplay( y );
if ViewInfo.Clip then Buffer := 'ON'
                 else Buffer := 'OFF';
OutTextXY( x, y, 'Clipping              : ' +
                 Buffer );
StepDisplay( y );
OutTextXY( x, y, 'Current position (CP) : ( ' +
                 N2S( GetX, 2 ) + ', ' +
                 N2S( GetY, 2 ) + ' )' );
StepDisplay( y );
OutTextXY( x, y, 'Max / this color      : ' +
                 N2S( MaxColors, 2 ) + ' / ' +
                 N2S( GetColor, 2 )  );
StepDisplay( y );
OutTextXY( x, y, 'Line thick / style    : ' +
                 N2S( LineInfo.Thickness, 2 ) + ' / ' +
                 LineStyles[ LineInfo.LineStyle ] );
StepDisplay( y );
OutTextXY( x, y, 'Fill color / style    : ' +
                 N2S( FillInfo.Color, 2 ) + ' / ' +
                 FillStyles[ FillInfo.Pattern ] );
StepDisplay( y );
OutTextXY( x, y, 'Character size / font : ' +
                 N2S( TextInfo.CharSize, 2 ) + ' / ' +
                 Fonts[ TextInfo.Font ] );
StepDisplay( y );
OutTextXY( x, y, 'Text direction        : ' +
                 TextDirect[ TextInfo.Direction ] );
StepDisplay( y );
OutTextXY( x, y, 'Horizontal justify    : ' +
                 HorizJust[ TextInfo.Horiz ] );
StepDisplay( y );
OutTextXY( x, y, 'Vertical justify      : ' +
                 VertJust[ TextInfo.Vert ] );
StepDisplay( y );
OutTextXY( x, y, 'Aspect Ratio  ( x/y ) : ' +
                 N2S( xasp, 4 ) + ' / ' +
                 N2S( yasp, 4 ) + ' = ' +
                 R2S( AspectRatio, 5, 3 ) );
Pause;
```

```
    end;
procedure Initialize;
    begin          {初始化图形系统并报告错误}
      GraphDriver:=DETECT;        {请求自动检测}
      InitGraph(GraphDriver,GraphMode,);
      TestGraphicError;          {检查图形错误}
      GetPalette(Palette);        {读取调色板参数}
      MaxColors:=GetMaxColor+1;
      MaxX:=380;                  {设置视口尺寸}
      MaxY:=194;                  {设置视口尺寸}
      GetAspectRatio(xasp,Yasp);      {硬件纵横比}
      AspectRatio:=xasp div yasp;
    end;
        {计算纵横比}    begin          {主程序}
      Initialize;        {设置图形模式}
      ReportStatus;  {显示图形设置}
      CloseGraph;          {设置文本模式}
    end.
```

第二章　视口、屏幕和页面函数

与文本显示模式相似，图形显示模式也提供了对窗口和屏幕的管理功能。这类函数包括ClearDevice，ClearViewPort（相当于 ClrScr），SetViewPort（相当于 Window），GetViewSettings 和 SetVisualPage 以及 SetActivePage。

这些函数在文本方式下并不都存在着等价的函数，同样，有些文本方式提供的函数，图形方式下也没有等价的函数。即使二者有某些函数是等价的，但使用方法也不完全相同。

下面首先介绍屏幕管理中的 ClearDevice 函数。

2.1　屏幕管理函数

2.1.1　ClearDevice

函数 ClearDevice 不须考虑视口的设置，而清除整个的图形屏，并将 CP 移至屏幕的左上角(0,0)处。这对活动视口没有影响（包括所有处于活动态的视口），视口的设置仍保持原状，只是清除图形屏和视口。该函数无返回值，也不产生错误条件。

【例】
```
uses GRAPH;
ClearDevice;
```

ClearDevice 类似于文本方式的 ClrScr 命令，但 ClrScr 仅对窗口起作用（它只能清除当前的活动窗口），而 ClearDevice 命令则将整个的活动图形屏复位，并且不影响其它的图形屏（即那些由硬件支持的图形屏）。须注意，文本方式的函数，如 ClrScr 不能在图形方式下工作，反之亦然。参见 ClearViewPort，SetActivePage 和 SetVisualPage。

2.1.2　ClearViewPort

函数 ClearViewPort 清除当前的视口（图形窗口），将 CP 移至左上角(0,0)位置。与 ClearDevice 函数不同，ClearViewPort 对一个活动窗口的设置仅限于屏幕的一个特定区域，其操作过程与文本方式的 ClrSer 相似。

【例】
```
uses GRAPH;
ClearViewPort;
```

参见 GetViewSettings 和 SetViewPort。

2.1.3　SetViewPort

函数 SetViewPort 大致相当于文本方式的函数 Window，它被用来设置一个活动窗口。坐标 XLeft，YTop，XRight，YBottom 都是绝对屏幕坐标，它们只对活动图形页面有影响（参见 SetActivePage）。

【例】
```
uses GRAPH;
var
    XLeft,YTop,XRight,YBottom:integer;
    ClipFlag:boolean;
SetViewPort(XLeft,YTop,XRight,YBottom,ClipFlag);
```

SetViewPort 的第五个参数是 ClipFlag，如果 ClipFlag 为真，则凡是超出当前视口的图形都

将被裁掉；如果 ClipFlag 为假，则作图时超出视口的部分不被裁去。

请注意：视口的限制对 GetImage 和 PutImage 没有影响，屏幕上的图像不论 ClipFlag 怎样设置，都不受视口边界的限制。

如果 SetViewPort 得到的坐标值无效，GraphResult 将返回值－11（图形错误或通常的错误），而原来的视口设置保持不变。函数 InitGraph 和 SetGraphMode 将按照当前模式的定义，把当前视口初始化成整个图形屏。参见 ClearViewPort 和 GetSettings。

2.1.4 Get View Settings

函数 GetViewSettings 利用一个记录型变量 ViewPort，返回当前的图形窗口坐标值和 ClipFlag 值。返回的坐标值为绝对屏幕坐标。

【例】
```
uses GRAPH;
var
    ViewPort:ViewPortType;
GetViewSetting(ViewPort);
```

如果 ClipFlag 为真，作图时超出当前视口的部分就被裁掉。细节可参考 SetViewPort，还可参考 ClearViewPort，InitGraph 和 SetGraphMode。

记录类型 ViewPortType 已在 Graph 部分预定义如下：
```
Type
  ViewPortType=record
                 x1,y1,x2,y2:integer;
                 clip:boolean;
               end;
```

2.2 多个图形页面

有些图形视频卡可以支持二～四个图形显示页面（大多数视频卡对颜色和分辨率没有限制）。为了使用这些功能，Turbo Pascal 提供了两个函数：SetActivePage（它选择活动输出页）和 SetVisualPage（它选择实际出现在屏幕或监视器上的图形页面。）这些功能常用于图形动画中。上述命令仅当驱动器和模式如表 2-1 所示时，才有效。

表 2-1 支持多页面的图形模式

图形驱动器	驱动器值	图形模式	模式值	分辨率 X 方向×Y 方向	可用颜色	图形页面
EGA	3	EGALO	0	640×200	16	4
		EGAHI	1	640×350	16	2
EGAMONO	5	EGAMONOHI	3	640×350	2	4[1]
HERC	7	HERCMONOHI	0	720×348	2	2
VGA	9	VGALO	0	640×200	16	4
		VGAMED	1	640×350	16	2[2]

注：

①对于 EGAMONO 卡必须有 256K RAM 来支持多视频页面，有些 EGAMONO 卡只有 64K RAM。

②最初，VGAHI 模式（640×480）只支持一个视频页面，但是许多与 VGA 兼容的视频卡（即多同步卡）都支持 2～4 个图形页