第3章 C语言课程设计相关知识

在本章，我们就课程设计中涉及的一些知识点作一个概述。这些知识点主要包括图形基础与图形函数、文件操作知识、动画技术、中断知识和发声技术等，是在 Turbo C 下的操作方式，不适用于 VC++ 。

3.1 图形知识

C 语言中提供了丰富的图形处理函数，本节将对本书中涉及的图形知识点作一个简要回顾，主要包括图形模式的初始化、屏幕设置函数、颜色相关函数、画图函数、填充函数、文本输出函数，以及与这些函数相关的各项参数等。

3.1.1 图形模式的初始化

1. void far initgraph(int far * gdriver,int far * gmode,char * path)

功能：显示模式控制。gdriver 和 gmode 分别表示图形驱动器和模式，path 是指图形驱动程序所在的目录路径。有关图形驱动器、图形模式及分辨率的值如表 3.1 所示。

表 3.1 图形驱动器、图形模式及分辨率参照表

图形驱动器(gdriver)		图形模式(gmode)		色 调	分 辨 率
符号常数	数 值	符号常数	数 值		
CGA	1	CGAC0	0	C0	320×200
		CGAC1	1	C1	320×200
		CGAC2	2	C2	320×200
		CGAC3	3	C3	320×200
		CGAHI	4	2 色	640×200
MCGA	2	MCGAC0	0	C0	320×200
		MCGAC1	1	C1	320×200
		MCGAC2	2	C2	320×200
		MCGAC3	3	C3	320×200

续表

图形驱动器(gdriver)		图形模式(gmode)		色　调	分 辨 率
符号常数	数　值	符号常数	数　值		
MCGA	2	MCGAMED	4	2 色	640×200
		MCGAHI	5	2 色	640×480
EGA	3	EGALO	0	16 色	640×200
		EGAHI	1	16 色	640×350
EGA64	4	EGA64LO	0	16 色	640×200
		EGA64HI	1	4 色	640×350
EGAMON	5	EGAMONHI	0	2 色	640×350
IBM8514	6	IBM8514LO	0	256 色	640×480
		IBM8514HI	1	256 色	1024×768
HERC	7	HERCMONOHI	0	2 色	720×348
PC3270	10	PC3270HI	0	2 色	720×350
DETECT	0	用于硬件测试			
ATT400	8	ATT400C0	0	C0	320×200
		ATT400C1	1	C1	320×200
		ATT400C2	2	C2	320×200
		ATT400C3	3	C3	320×200
		ATT400MED	4	2 色	320×200
		ATT400HI	5	2 色	320×200
VGA	9	VGAL0	0	16 色	640×200
		VGAMED	1	16 色	640×350
		VGAHI	2	16 色	640×480

2. void far detectgraph(int * gdriver, * gmode)

功能：自动检测显示器硬件。gdriver 和 gmode 与 initgraph()函数中的意义一样，仍然分别表示图形驱动器和模式。

3.1.2 屏幕颜色相关函数

1. void far setbkcolor(int color)

功能：设置背景色。其中，color 表示作图的颜色，可以用颜色的符号常量表示，也可以用代表该颜色的数值来表示，颜色符号常量和对应数值的关系如表 3.2 所示。

表 3.2 颜色符号常量与其数值对应表

符号常量	数值	含义	符号常量	数值	含义
BLACK	0	黑色	DARKGRAY	8	深灰色
BLUE	1	蓝色	LIGHTBLUE	9	淡蓝色
GREEN	2	绿色	LIGHTGREEN	10	淡绿色
CYAN	3	青色	LIGHTCYAN	11	淡青色
RED	4	红色	LIGHTRED	12	淡红色
MAGENTA	5	洋红色	LIGHTMAGENTA	13	淡洋红色
BROWN	6	棕色	YELLOW	14	黄色
LIGHTGRAY	7	淡灰色	WHITE	15	白色

2. void far setcolor(int color)

功能：设置前景色，color 的取值参见表 3.2。

3. int far getbkcolor(void)

功能：返回当前背景色。

4. int far getcolor(void)

功能：返回当前背景色。

5. int far getmaxcolor(void)

功能：返回当前可用的最大颜色值。

3.1.3 图形窗口和图形屏幕函数

1. 图形窗口操作

1) void far setviewport(int x0,int y0,int x1,int y1,int clipflag)

设定一个以(x0,y0)为左上角、以(x1,y1) 为右下角的图形窗口，其中，x0、y0、x1、y1 是相对于整个屏幕的坐标。如果 clipflag 为 1，则超出窗口的输出图形自动被裁剪掉，即所有作图限制于当前图形窗口之内；如果 clipflag 为 0，则不进行裁剪，即作图将无限制地扩展于窗口边界之外，直到屏幕边界。

2) void far clearviewport(void)

清除当前图形窗口，并把光标从当前位置移到原点(0,0)。

3) void far getviewsettings(struct viewporttype far * viewport)

获得关于现行窗口的信息，并将其存于 viewporttype 定义的结构变量 viewport 中。其中，viewporttype 的结构说明如下：

```
struct viewporttype
```

```
{
    int left,top,right,bottom;
    int clipflag;
};
```

2. 图形屏幕操作

1) void far setactivepage(int pagenum)

为图形输出选择激活页,即后续图形的输出被写到函数选定的 pagenum 页面,该页面并不一定可见。

2) void far setvisualpage(int pagenum)

使 pagenum 所指定的页面变成可见页,页面从 0 开始。如果先用 setactivepage()函数在不同页面上画出一幅幅图像,再用 setvisualpage()函数交替显示,就可以实现一些动画的效果。

3) unsined far imagesize(int x0,int y0,int x1,int y1)

该函数一般在保存指定范围内的像素时使用,它计算要保存从左上角为(x0,y0)到右下角为(x1,y1) 的图形屏幕区域所需的字节数。

4) void far getimage(int x0,int y0,int x1,int y1,void far * buf)

将从左上角为(x0,y0)到右下角为(x1,y1) 的图形屏幕区域的图像保存到 buf 所指向的内存空间,该内存空间的大小由 imagesize()函数计算。

5) void far putimage(int x,int y,void * mapbuf,int op)

将保存的图像输出到左上角点(x,y)的位置上,op 规定了如何释放内存中的图像,图像输出方式如表 3.3 所示。

表 3.3 图像输出方式

符号常数	数值	含 义	符号常数	数值	含 义
COPY_PUT	0	复制	AND_PUT	3	与屏幕图像与后复制
XOR_PUT	1	与屏幕图像异或的复制	NOT_PUT	4	复制反像的图形
OR_PUT	2	与屏幕图像或后复制			

6) void far cleardevice(void)

清除屏幕内容。

3.1.4 画图函数

1. 画点

1) 画点函数

(1) void far putpixel(int x,int y,int color)

在坐标(x,y)处以 color 所代表的颜色画一点。

(2) int far getpixel(int x,int y)

获得点(x,y)处的颜色值。

2) 有关坐标位置的函数

(1) int far getmaxx(void)

返回 x 轴的最大值。

(2) int far getmaxy(void)

返回 y 轴的最大值。

(3) int far getx(void)

返回光标所在位置的横坐标。

(4) void far gety(void)

返回光标所在位置的纵坐标。

(5) void far moveto(int x,int y)

移动光标到(x,y)点。

(6) void far moverel(int dx,int dy)

把光标从当前位置(x,y)移动到位置(x+dx,y+dy)。

2. 画线

1) 画线函数

(1) void far line(int x0,int y0,int x1,int y1)

从点(x0,y0)到点(x1,y1) 画直线。

(2) void far lineto(int x,int y)

画一条从当前位置到(x,y)的直线。

(3) void far linerel(int dx,int dy)

画一条从当前位置(x,y)到位置(x+dx,y+dy)之间的直线。

(4) void far circle(int x,inty,int radius)

以(x,y)为圆心、radius 为半径画一个圆。

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

以(x,y)为圆心、radius 为半径,画一段从 stangle(角度)到 endangle(角度)的圆弧线。

(6) void ellipse(int x,int y,int stangle,int endangle,int xradius,int yradius)

以(x,y)为中心,以 xradius、yradius 分别为 x 轴和 y 轴半径,画一段从角 stangle 到 endangle 的椭圆线。当 stangle=0、endangle=360 时,画出一个完整的椭圆。

(7) void far rectangle(int x1,int y1,int x2,inty2)

以(x1,y1) 为左上角、(x2,y2) 为右下角画一个矩形框。

(8) void far drawpoly(int numpoints,int far * polypoints)

画一个顶点数为 numpoints、各顶点坐标由 polypoints 给出的多边形。

2) 设定线型函数

(1) void far setlinestyle(int linestyle,unsigned upattern,int thickness)

该函数用来设置线的有关信息,其中 linestyle 表示线的形状(线形状的取值及含义见表 3.4),thickness 表示线的宽度(线宽度的取值及含义见表 3.5),对于 upattern,只有 linestyle 选 USERBIT LINE 时才有意义(选其他线型,upattern 取 0 即可)。此处 upattern 的 16 位二进制数的每一位代表一个像元,如果为 1,则该像元打开,否则该像元关闭。

表 3.4 线形状的取值及含义

符号常数	数 值	含 义	符号常数	数 值	含 义
SOLID_LINE	0	实线	DASHED_LINE	3	点画线
DOTTED_LINE	1	点线	USERBIT_LINE	4	用户定义线
CENTER_LINE	2	中心线			

表 3.5 线宽的取值及含义

符号常数	数 值	含 义
NORM_WIDTH	1	一点宽
THIC_WIDTH	3	三点宽

(2) void far setwritemode(int mode)

该函数规定画线的方式。mode 的取值为 1 或 0。当取 0 时,表示画线时将所画位置的原来信息覆盖了;当取 1 时,则表示画线时用现在特性的线与所画之处原有的线进行异或(XOR)操作,实际上画出的线是原有线与现在规定的线进行异或后的结果。因此,当线的特性不变,进行两次画线操作相当于没有画线。

3.1.5 封闭图形的填充

1. 先画轮廓再填充

1) void far bar(int x1,int y1,int x2,int y2)

先画一个以(y1,y1) 为左上角、(x2,y2) 为右下角的矩形窗口,再按规定模式和颜色填充。

2) void far bar3d(int x0,int y0,int x1,int y1,int depth,int topflag)

当 topflag 为非 0 时,画出一个三维的长方体;当 topflag 为 0 时,三维图形不封顶。

3) void far pieslice(int x,int y,int stangle,int endangle,int radius)

先画一个以(x,y)为圆心、radius 为半径、从角度 stangle 到角度 endangle 的扇形,再按规定方式填充。

4) void far sector(int x,int y,int stangle,intendangle,int xradius,int yradius)

先画一个以(x,y)为圆心,以 xradius、yradius 分别为 x 轴和 y 轴半径,从角度 stangle 到角度 endangle 的椭圆扇形,再按规定方式填充。

2. 任意封闭图形的填充

void far floodfill(int x,int y,int border):该函数可对任意封闭图形进行填充。其中(x,y)为封闭图形内的任意一点,border 为边界的颜色,border 指定的颜色值必须与图形轮廓的颜色值相同。

提示:(x,y)必须在所要填充的封闭图形内部,否则不能进行填充。如果不是封闭图形,则填充会从没有封闭的地方溢出去,填满其他地方。

3. 设定填充方式

1) void far setfillstyle(int pattern,int color)

以 pattern 为填充模式和以 color 为填充颜色对指定图形进行填充。填充模式 pattern 的取值和含义如表 3.6 所示。

表 3.6　填充模式 pattern 的取值和含义

符号常数	数值	含　义	符号常数	数值	含　义
EMPTY_FILL	0	以背景色填充	HATCH_FILL	7	直线网格填充
SOLID_FILL	1	实填充	XHATCH_FILL	8	斜网格填充
LINE_FILL	2	直线填充	INTTERLEAVE_FILL	9	间隔点填充
LTSLASH_FILL	3	斜线填充	WIDE_DOT_FILL	10	稀疏点填充
SLASH_FILL	4	粗斜线填充	CLOSE_DOS_FILL	11	密集点填充
BKSLASH_FILL	5	粗反斜线填充	USER_FILL	12	用户自定义填充
LTBKSKASH_FILL	6	反斜线填充			

2) void far setfillpattern(char * upattern,int color)

该函数用于设置用户定义的填充图形的颜色,以供对封闭图形进行填充。其中,upattern 是一个指向 8 字节的指针。这 8 字节定义了 8×8 点阵的图形。每字节的 8 位二进制数表示水平 8 点,8 字节表示 8 行,然后以此为模型向各封闭区域填充。

3) void far getfillpattern(char * upattern)

该函数将用户定义的填充图模存入 upattern 指针指向的内存区域。

4) void far getfillsetings(struct fillsettingstype far * fillinfo)

获得当前图形模式的颜色并将其存入结构指针变量 fillinfo 中。其中,fillsettingstype 结构定义如下:

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

其中,pattern 表示当前的填充模式,color 表示填充的颜色。

3.1.6　图形模式下的文本输出

1. 文本输出函数

1) void far outtext(char far * text)

该函数输出字符串指针 text 所指的文本所在的位置。

2) void far outtextxy(int x,int y,char far * text)

该函数在指定位置(x,y)输出字符串指针 text 所指的文本。

2.文本参数设置函数

1) void far setcolor(int color)

该函数用于设置输出文本的颜色,color 表示要设置的颜色。

2) void far settextjustify(int horiz,int vert)

该函数用于设置显示的方位。对使用 outtextxy()函数所输出的字符串,其中,哪个点对应于坐标(x,y)在 Turbo C 2.0 中是有规定的。如果把一个字符串看成一个长方形的图形,在水平方向显示时,字符串长方形在垂直方向就有顶部、中部和底部三个位置,水平方向就有左、中、右三个位置,两者结合所确定的位置就对准函数中的(x,y)位置。

settextjustify()函数中的参数 horiz 指出水平方向的位置(即左、中、右中的一个),参数 vert 指出垂直方向的位置(即顶部、中部、底部中的一个)。有关参数 horiz 和 vert 的取值及含义见表 3.7。

表 3.7 参数 horiz 和 vert 的取值及含义

符号常数	数值	含义	符号常数	数值	含义
LEFT_TEXT	0	水平	TOP_TEXT	2	垂直
RIGHT_TEXT	2	水平	CENTER_TEXT	1	水平或垂直
BOTTOM_TEXT	0	垂直			

3) void far settextstyle(int font,int direction,int charsize)

该函数用于设置输出字符的字体 font(见表 3.8)、方向 direction 和字体大小 charsize(见表 3.9)。其中方向 direction 的取值有 HORIZ_DIR(用数值 0 表示)和 VERT_DIR(用数值 1 表示)两个,分别表示从左向右和从底向顶输出字符。

表 3.8 字体 font 的取值及含义

符号常数	数值	含义
DEFAULT_FONT	0	默认字体,8×8 点阵字体
TRIPLEX_FONT	1	三倍笔画字体
SMALL_FONT	2	小号笔画字体
SANSSERIF_FONT	3	无衬线笔画字体
GOTHIC_FONT	4	黑体笔画字体

表 3.9 字体大小 charsize 的取值及含义

符号常数或数值	含义
1	8×8 点阵
2	16×16 点阵
3	24×24 点阵
4	32×32 点阵

续表

符号常数或数值	含　义
5	40×40 点阵
6	48×48 点阵
7	56×56 点阵
8	64×64 点阵
9	72×72 点阵
10	80×80 点阵
USER_CHAR_SIZE=0	用户定义的字符大小

3.2　文件操作知识

本节我们将对文件操作的基本知识作一个简要回顾，包括文件的打开与关闭、文件的读写、文件的状态判断及文件的定位。

3.2.1　文件的打开与关闭

1. 文件的打开

C 语言中打开文件的函数是 fopen()，其一般调用形式如下：

```
FILE * fp
fp=fopen(filename, method)
```

其中，fp 是一个指向 FILE 类型结构体的指针变量，filename 表示要打开的文件的名字，method 表示文件的使用方式，表 3.10 列出了文件的使用方式及其含义。

表 3.10　文件的使用方式及其含义

文件的使用方式	含　义	文件的使用方式	含　义
r，只读	为输入打开一个文本文件	r+，读写	为读/写打开一个文本文件
w，只写	为输出打开一个文本文件	w+，读写	为读/写建立一个新的文本文件
a，追加	向文本文件尾增加数据	a+，读写	为读/写打开一个文本文件
rb，只读	为输入打开一个二进制文件	rb+，读写	为读/写打开一个二进制文件
wb，只写	为输出打开一个二进制文件	wb+，读写	为读/写建立一个新的二进制文件
ab，追加	向二进制文件尾增加数据	ab+，读写	为读/写打开一个二进制文件

提示：如果不能打开一个文件，即打开文件失败，则 fopen()函数将返回一个空指针 NULL。通常可以通过判断 fopen()返回的值来决定是否打开成功。

2. 文件的关闭

在使用完一个文件后应该关闭它，以防止它被误用。关闭文件的函数是 fclose()，其一般调用形式如下：

```
fclose(fp)
```

fp 即为文件指针，是我们前面打开文件时创建的 fp。fclose()函数也有返回值，如果顺利关闭的话则返回 0；否则返回－1，可以用 ferror()函数(见 3.2.3 节)来测试。

3.2.2 文件的读写

对文件的读写，可以有多种方式，可以读写一个字符、一个字符串、一块数据或者一个整数，有时候也需要进行格式化输入和输出等。

1. 从文件读出

1) fgetc()

fgetc()函数用于从一个指定的文件中读出一个字符，该文件必须是以读或者读写方式打开的，其一般调用形式如下：

```
ch=fgetc(fp)
```

fp 为文件型指针变量，ch 为读出的字符变量。

2) fgets()

fgets()函数用于从一个指定的文件中读出一个字符串，其一般调用形式如下：

```
fgets(str, num, fp)
```

num 为需要读取的字符个数，fp 为文件型指针变量。从 fp 指向的文件中读取 num－1 个字符，然后在最后加一个'\0'字符，再把它们放入数组 str 中。

3) getw()

getw()函数表示从一个磁盘文件中读取一个整数，其一般调用形式如下：

```
getw(fp)
```

fp 为文件型指针变量。

4) fread()

fread()函数用于读取一组数据，其一般调用形式如下：

```
fread(buffer, size, count, fp)
```

buffer 是读取数据所存放的地方，size 表示要读写的字节数，count 表示要进行读写多少字节的数据项，fp 为文件型指针。

5) fscanf()

fscanf()函数用于从磁盘文件中读取 ASCII 字符，并将读取的数据存入指定变量中，其一般调用形式如下：

```
fscanf(fp, format string, in_list)
```

fp 为文件型指针，format string 是格式字符串，in_list 表示输入表列。

2. 写入文件

1) fputc()

fputc()函数表示将一个字符写入指定的磁盘文件中，其一般调用形式如下：

```
fputc(ch, fp)
```

ch 为要写入的字符，fp 为文件型指针变量。

2) fputs()

fputs()函数表示向指定文件写入一个字符串，其一般调用形式如下：

```
fputs(str, fp)
```

str 表示要写入的字符串，fp 为文件型指针变量。

3) putw()

putw()函数用于将一个整数写入指定的文件，其一般调用形式如下：

```
putw(digit, fp)
```

digit 表示要写入的整数，fp 为文件型指针变量。

4) fwrite()

fwrite()用来将一组数据写入指定的文件，其一般调用形式如下：

```
fwrite(buffer, size, count, fp)
```

其各个参数的意义和 fread()函数中的相同，只是这里的 buffer 存放的是要写入文件的数据。

5) fprintf()

fprintf()函数用于格式化输出字符串到指定的文件，其一般调用形式如下：

```
fprintf(fp, format string, out_list)
```

fp 为文件型指针，format string 是格式字符串，out_list 表示输出表列。我们举例来说明如下：

```
fprintf(fp,"%d, %2.2f", i, j)
```

本语句的作用是将整型变量 i 和实型变量 j 的值按照%d 和%2.2f 的形式输出到 fp 所指向的文件上。

3.2.3 文件的状态

1. 文件结束判断

在 ANSI 标准中，使用 feof()函数来判断文件是否结束，其一般调用形式如下：

```
feof(fp)
```

fp 为文件型指针，如果遇到文件结束符，即表示文件结束返回非 0，否则返回 0。

2. 错误检测及清除

1) ferror()

在调用各种输入输出函数时，如果出现了错误，可以用 ferror()函数来检测，其一般调用形式如下：

```
ferror(fp)
```

如果 ferror 返回 0，则表示没有出错。

2) clearer()

clearer()函数用于将文件错误标志和文件结束标志置为 0，其一般调用形式如下：

```
clearer(fp)
```

如果在调用一个输入输出函数出错后，再调用该函数，即可将 ferror()值清为 0。

3.2.4 文件的定位

文件中有一个位置指针指向当前读写的位置。有时我们需要获取位置指针的位置或者改变当前的指向位置，C 语言中提供了几个函数来实现这些功能。

1. rewind()

rewind()函数将位置指针重新返回到文件的开头，其一般调用形式如下：

```
rewind(fp)
```

2. ftell()

ftell()函数用于获取流式文件中的当前位置，用相对于文件开头的位移量来表示，其一般调用形式如下：

```
ftell(fp)
```

如果返回正整数，则表示当前的存放位置，如果返回值为－1L，则表示出错。

3. fseek()

fseek()函数用于实现改变位置指针所指向的位置，其一般调用形式如下：

```
(文件类型指针,位移量,起始点)
```

起始点有三个值，分别是文件开始(SEEK_SET，用数字 0 表示)、文件当前位置 (SEEK_CUR，用数字 1 表示)和文件末尾(SEEK_END，用数字 2 表示)；位移量是以起始点为基点而移动的字节数，如果字节数为正表示向前移动，如果为负则表示向后移动。ANSI C 标准要求位移量是 long 型数据，并规定在数字的末尾加一个字母 L，表示 long 型。

举例说明,假定有如下的调用:

```
fseek(fp, 100L, 1)
```

该语句表示将位置指针向前移动到离当前位置 100 字节处。

3.3 动画技术

我们知道电影或动画片是由一张张图像组成的,它利用人眼不能够分辨出时间间隔在25毫秒内的动态图像变化这一特性,在这些连续图像被放映时,从视觉效果上给人以动的感觉。所以在计算机屏幕上产生运动的效果需要动画技术。

3.3.1 采用延迟与清屏交错的实现方法

这种方法利用 cleardevice()和 delay()函数相互配合,先画一幅图形,让它延迟一段时间,然后清屏,再画另一幅,如此反复,就形成动态效果。本小节的例 3-1 分别通过函数 graphone()、graphtwo()和 graphthree()实现了三幅简单的动画画面,这三幅画面不停地进行切换。

例 3-1

```
#include<graphics.h>
#include<stdlib.h>
#include<dos.h>
int x,y,maxcolor;
void graphone(char *str);              /*使字符串 str 左右运动,线条上下运动*/
void graphtwo(char *str);              /*使字符串 str 上下运动,线条左右运动*/
void graphthree(char *str);    /*使字符串 str 由小变大,再由大变小,直线也随之变化*/
main()
{
    int i,driver,mode;
    char *str="W E L C O M E !";
    driver=DETECT;
    mode=0;
    initgraph(&driver,&mode,"");       /*系统初始化*/
    cleardevice();                     /*清屏*/
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    x=getmaxx();                       /*返回当前图形模式下的最大有效的 x 值*/
    y=getmaxy();                       /*返回当前图形模式下的最大有效的 y 值*/
    maxcolor=getmaxcolor();            /*返回当前图形模式下最大有效的颜色值*/
    while(!kbhit())
    {
        graphone(str);                 /*第一个动画*/
        graphtwo(str);                 /*第二个动画*/
        graphthree(str);               /*第三个动画*/
    }
```

```
    getch();
    closegraph();                          /*关闭图形模式*/
}
void graphone(char *str)
{
    int i;
    for(i=0;i<40;i++)
    {
        setcolor(1);
        settextstyle(1,0,4);
        setlinestyle(0,0,3);
        cleardevice();
        line(150,y-i*15,150,y-300-i*15);
        line(170,y-i*15-50,170,y-350-i*15);
        line(130,y-i*15-50,170,y-i*15-50);
        line(150,y-300-i*15,190,y-300-i*15);
        line(x-150,i*15,x-150,300+i*15);
        line(x-170,i*15-50,x-170,250+i*15);
        line(x-150,i*15,x-190,i*15);
        line(x-130,250+i*15,x-170,250+i*15);
        outtextxy(i*25,150,str);
        outtextxy(x-i*25,y-150,str);
        delay(5000);
    }
}
void graphtwo(char *str)
{
    int i;
    for(i=0;i<30;i++)
    {
        setcolor(5);
        cleardevice();
        settextstyle(1,1,4);
        line(i*25,y-100,300+i*25,y-100);
        line(i*25,y-120,300+i*25,y-120);
        line(x-i*25,100,x-300-i*25,100);
        line(x-i*25,120,x-300-i*25,120);
        outtextxy(150,i*25,str);
        outtextxy(x-150,y-i*25,str);
        delay(5000);
    }
}
void graphthree(char *str)
{
    int i,j,color,width;
```

```
    color=random(maxcolor);          /*随机得到颜色值*/
    setcolor(color);
    settextstyle(1,0,1);             /*设置字符串的格式*/
    outtextxy(x/2,y/2-100,str);      /*显示字符串*/
    delay(8000);
    for(i=0;i<8;i++)                 /*字符串由小变大*/
    {
        cleardevice();               /*清屏*/
        settextstyle(1,0,i);
        outtextxy(x/2,y/2-i*10-100,str);
        outtextxy(x/2,y/2+i*10-100,str);
        width=textwidth(str);        /*得到当前字符串宽度*/
        setlinestyle(0,0,1);         /*设置画线格式*/
        line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*
        15-70);
        line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*
        15-60);
        line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
        line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
        line((x-width)/2+5*(8-i)-10,y/2+i*15-10,(x+width)/2-5*(8-i),y/2+
        i*15-10)
        line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
        delay(8000);
    }
    for(i=7;i>=0;i--)                /*字符串由大变小*/
    {
        cleardevice();               /*清屏*/
        settextstyle(1,0,i);
        outtextxy(x/2,y/2-i*10-100,str);
        outtextxy(x/2,y/2+i*10-100,str);
        width=textwidth(str);
        setlinestyle(0,0,1);
        line((x-width)/2+10*(8-i),y/2+i*15-70,(x+width)/2-10*(8-i),y/2+i*
        15-70);
        line((x-width)/2+5*(8-i),y/2+i*15-60,(x+width)/2-5*(8-i),y/2+i*
        15-60);
        line((x-width)/2,y/2+i*15-50,(x+width)/2,y/2+i*15-50);
        line((x-width)/2,y/2+i*15-20,(x+width)/2,y/2+i*15-20);
        line((x-width)/2+5*(8-i),y/2+i*15-10,(x+width)/2-5*(8-i),y/2+i*
        15-10);
        line((x-width)/2+10*(8-i),y/2+i*15,(x+width)/2-10*(8-i),y/2+i*15);
        delay(8000);
    }
}
```

程序中用到的库有 graphics.h、dos.h 和 stdlib.h。其中，graphics.h 中的图形函数，除

initgraph()、cleardevice()、closegraph()、settextjustify()、settextstyle()、setlinestyle()、outtextxy()、setcolor()、line()外，还包括如下：

```
void far textwidth(char far * str):
```

功能：以像素为单位，返回由 str 所指向的字符串宽度，针对当前字符的字体与大小。

该程序用到的 dos.h 中的库函数有 delay()，其原型说明如下：

```
void delay(unsigned milliseconds)
```

功能：该函数将程序的执行暂停一段时间(毫秒)。

```
void far random(int num)
```

功能：此函数返回一个 0～num 范围内的随机数。该函数在 stdlib.h 库中。

3.3.2 动态开辟视图窗口的方法

我们还可以利用视图窗口设置技术来实现视图窗口动画效果，具体方法是：在不同视图窗口中设置同样的图像，然后让视图窗口沿 x 轴方向移动设置，这次出现前要清除上次视图窗口的内容，这样就会出现图像沿 x 轴移动的效果。也就是说，在位置动态变化但大小不变的视图窗口中(用 setviewport()函数)，设置固定图形(也可是微小变化的图像)，这样虽呈现在观察者面前的是当前视图窗口位置在动态变化，但视觉上却像是看到图像在屏幕上动态变化一样。

例 3-2 就是这样做的，不断地沿 x 轴开辟视图窗口，就像一个大小一样的窗口沿 x 轴在移动，由于总有 clearviewport 函数清除上次窗口的相同立方体，因而视觉效果上，就像一个立方体从左向右移动一样。程序中定义的 movebar 函数作用是开辟一个视图窗口，并画一个填色的立方体，保留一阵(delay(250000))然后清除它，主程序不断调用它，因每次顶点 x 坐标在增加，因而效果是立方体沿 x 轴从左向右在运动。

例 3-2

```
#include<graphics.h>
#include<dos.h>
main()
{
    int i,driver,mode;
    graphdriver=DETECT;
    initgraph(&driver,&mode,"");
    for(i=0;i<25;i++)
    {
        setfillstyle(1, i);
        movebar(i * 20);
    }
    closegraph();
}
movebar(int xorig)          /*设窗口并画填色小立方体*/
```

```
{
    setviewport(xorig,0,639,199,1);
    setcolor(5);
    bar3d(10,120,60,150,40,1);
    floodfill(70,130,5);
    floodfill(30,110,5);
    delay(250000);
    clearviewport();
}
```

采用上面的两种方法对较复杂图形不适宜，一则画图形要占较长时间，二则视图窗口位置切换的时间变得较长，因而动画效果就会变差。

3.3.3　屏幕图像存储再放的方法

在图形方式下，与文本方式类似，除了清屏函数 cleardevice()外，还有其他的对屏幕图像操作的函数。其中一类是屏幕图像存储和显示函数，包括：存屏幕图像到内存区 getimage()函数，测定图像所占字节数的 imagesize()函数，将所存图像显示的 putimage()函数，函数详情见 3.1.3 节。

例 3-3 演示了动画的工作方式，for 循环用来在屏幕上方产生连续的五个方框，方框中套用洋红色填充的小方块，五个图像全一样。循环结束后，又在屏幕下方画出两个小框，小框用洋红色填充并在大框内。程序运行后，立即在屏幕上显示出上述图案，当按任意键后，则由函数 imagesize()得到屏幕下方大框区域内图像所占字节数，然后由 malloc()函数按字节数分配内存缓冲区 buffer，再由 getimage()函数将图像存到 buffer 中，然后复制到屏幕上方左边第一个框位置。按任意键后，又将 buffer 中图像和第二个框图像进行与操作后显示，再按任意键，buffer 中的图像又和第三个方框内图像进行或操作并显示，如此重复，则可将 5 种逻辑操作结果均显示在屏幕上。

注意：COPY 和 NOT 操作将与原来屏幕上的图像无关，buffer 中图像经过这两种操作，将覆盖掉原屏幕上图像，并将结果进行显示。

例 3-3

```
#include<graphics.h>
main()
{
    int i,j,driver,mode,size;
    void *buffer;
    driver=DETECT;
    initgraph(&driver,&mode,"");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(YELLOW);
    setlinestyle(0,0,1);                    /*用细实线*/
    setfillstyle(1,5);                      /*用洋红色实填充*/
    for(i=0;i<5;i++)                        /*产生连续的五个方框中套小框*/
```

```
    {
        j=i * 110;
        rectangle(80+j,100,130+j,150);      /* 产生小框且用洋红色填充 */
        floodfill(110+j,140,YELLOW);
        rectangle(50+j,100,130+j,180);      /* 画大框 */
    }
    rectangle(50,340,100,420);              /* 产生一个小框 */
    floodfill(80,360,YELLOW);               /* 用洋红色填充 */
    rectangle(50,340,130,420);              /* 产生一个大框 */
    getch();
    size=imagesize(40,300,132,430);
                                    /* 取得(40,300)右下角(132,430)区域图像字节数 */
    buffer=malloc(size);                    /* 分配缓冲区(按字节数) */
    getimage(40,300,132,430,buffer);        /* 存图像 */
    putimage(40,60,buffer,COPY_PUT);        /* 重新复制 */
    getch();
    j=110;
    putimage(40+j,60,buffer,AND_PUT);       /* 和屏幕上的图与操作 */
    getch();
    putimage(40+2*j,60,buffer,OR_PUT);
    getch();
    putimage(40+3*j,60,buffer,XOR_PUT);
    getch();
    putimage(40+4*j,60,buffer,NOT_PUT);
    getch();
    closegraph();
}
```

同制作幻灯片一样,将整个动画过程变成一个个片断,然后存到显示缓冲区内,当把它们按顺序重放到屏幕上时,就出现了动画效果,这可以用 getimage()和 putimage()函数来实现,这种方法较快,因它已事先将要重放的画面画好了。余下的问题,就是计算应在什么位置重放的问题了。

例 3-4 演示了利用这种方法产生的两个洋红色小球碰撞、弹回然后又碰撞的动画效果。

例 3-4

```
#include<graphics.h>
main()
{
    int i,driver,mode,size;
    void *buffer;
    driver=DETECT;
    initgraph(&driver,&mode,"");
    setbkcolor(BLUE);
    cleardevice();
    setcolor(YELLOW);
```

```
    setlinestyle(0,0,1);
    setfillstyle(1,5);
    circle(100,200,30);
    floodfill(100,200,YELLOW);                          /* 填充圆 */
    size=imagesize(69,169,131,231);                     /* 指定图像占字节数 */
    buffer=malloc(size);                                /* 分配缓冲区(按字节数) */
    getimage(69,169,131,231,buffer);                    /* 存图像 */
    putimage(500,169,buffer,COPY_PUT);                  /* 重新复制 */
    do{
        for(i=0;i<185;i++)
        {
            putimage(70+i,170,buffer,COPY_PUT);         /* 左边球向右运动 */
            putimage(500-i,170,buffer,COPY_PUT);        /* 右边球向左运动 */
        }                                               /* 两球相撞后循环停止 */
        for(i=0;i<185;i++)
        {
            putimage(255-i,170,buffer,COPY_PUT);        /* 左边球向左运动 */
            putimage(315+i,170,buffer,COPY_PUT);        /* 右边球向右运动 */
        }
    }while (!kbhit());                                  /* 当不按键时重复上述过程 */
    getch();
    closegraph();
}
```

3.3.4　利用页交替的方法

对屏幕图像操作的函数，还有一类是设置显示页函数。显示适配器的显示存储器为VRAM，图形方式下存储在VRAM中的一满屏图像信息称为一页。每页一般为64K字节，VRAM可以存储要显示的图像几页(视VRAM容量而定，最大可达8页)，Turbo C 2.0支持页的功能有限，按在图形方式下显示的模式最多支持4页(EGALO显示方式)，一般为2页(注意对CGA，仅有1页)，因存储图像的页显示时，一次只能显示1页，因此必须设定某页为当前显示的页(又称可视页)，缺省时定为0页，如图3.1所示。

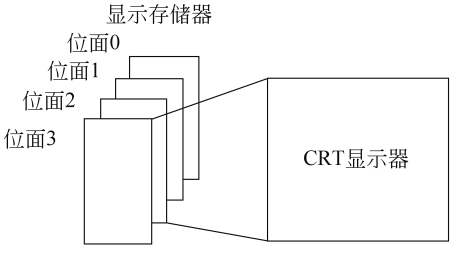

图3.1　显示页

正在由用户编辑图形的页称为当前编辑页(又称激活的页)，这个页不等于显示页，即若用户不设定该页为当前显示页时，在该页上编辑的图形将不会在屏幕上显示出来。缺省时，设定0页为当前编辑页，即若不用下述的页设置函数进行设置，就认定0页既是编辑页，又是当前显示页。

设置激活页和显示页的函数为setactivepage()和setvisualpage()，详情见3.1.3节。这两个函数只能用于EGA、VGA等显示适配器。前者设置由pagenum指出的页为激活的

页，后者设置可显示的页。当设定了激活的页，即编辑页后，则程序中其后的画图操作均在该页进行，若它不定为显示页，则其上的图像信息并不会在屏幕上显示出来。

例 3-5 的程序演示了设置显示页函数的应用。首先用 setactivepage(1) 设置 1 页为编辑页，在上面画出一个红色边框、用淡绿色填充的圆，此图并不显示出来（因缺省时，定义 0 页为可视页）。接着又定义 0 页为编辑页并清屏（即清 0 页），也定义 0 页为可视页，并在其上画出一个用洋红色填充的方块，该方块将在屏幕上显示出来。接着进入 do 循环，设置 1 页为可视页，因而其上的圆便在屏幕上显示出来，方块的图像消失，用 delay(2000)将圆图像保持 2000 毫秒即 2 秒，当不按键时，下一次循环又将 0 页设为可视页，因而方块的图像显示出来，圆图像又消失。保持 2 秒后，又重复刚开始的过程。这样我们就会看到：屏幕上同一位置洋红色方块和淡绿色圆交替出现，若将 delay 时间变少，将会出现动画的效果。

例 3-5

```
#include<graphics.h>
#include<dos.h>
main()
{
    int i,graphdriver,graphmode,size,page;
    graphdriver=DETECT;
    initgraph(&graphdriver,&graphmode,"");
    cleardevice();
    setactivepage(1);                    /*设置 1 页为编辑页*/
    setbkcolor(BLUE);
    setcolor(RED);
    setfillstyle(1,10);
    circle(130,270,30);                  /*画圆*/
    floodfill(130,270,4);                /*用淡绿色填充圆*/
    setactivepage(0);                    /*设置 0 页为编辑页*/
    cleardevice();                       /*清 0 页*/
    setfillstyle(1,5);
    bar(100,210,160,270);                /*画方块并填充洋红色*/
    setvisualpage(0);                    /*设置 0 页为可视页*/
    page=1;
    do
    {
        setvisualpage(page);             /*显示设定页的图像*/
        delay(2000);                     /*延迟 2000ms*/
        page=page-1;
        if(page<0)
            page=1;
    }while(!kbhit());
    getch();
    closegraph();
}
```