

EasyGraphicsEngine

——EGE 帮助文档

版本：20110401 0.3.6 Release

目录

EasyGraphicsEngine——EGE 帮助文档.....	1
基本说明	5
安装	6
超简单的使用预览	7
库函数说明	8
绘图环境相关函数	9
initgraph	10
cleardevice	11
closegraph	12
getviewport	13
setviewport	14
clearviewport	15
setactivepage	16
setinitmode	17
setvisualpage	18
window_getviewport	19
window_setviewport	20
颜色表示及相关函数	21
getbkcolor	22
GetBValue	23
getcolor	24
GetGValue	25
GetRValue	26
HSLtoRGB	27
HSVtoRGB	28
RGB	29
RGBtoGRAY	30
RGBtoHSL	31
RGBtoHSV	32
setbkcolor	33
setbkcolor f	34
setbkmode	35
setcolor	36
setfontbkcolor	37
颜色表示	38
绘制图形相关函数	39
arc	41
bar	42
bar3d	43
circle	45
drawbezier	46
drawlines	47

<u>drawpoly</u>	48
<u>ellipse</u>	49
<u>fillellipse</u>	50
<u>fillpoly</u>	51
<u>floodfill</u>	53
<u>getheight</u>	54
<u>getlinestyle</u>	55
<u>getpixel</u>	56
<u>getwidth</u>	57
<u>getx</u>	58
<u>gety</u>	59
<u>line</u>	60
<u>linerel</u>	61
<u>lineto</u>	62
<u>moverel</u>	63
<u>moveto</u>	64
<u>pieslice</u>	65
<u>putpixel</u>	66
<u>putpixels</u>	67
<u>rectangle</u>	68
<u>sector</u>	69
<u>setfillstyle</u>	70
<u>setlinestyle</u>	71
<u>setwritemode</u>	72
<u>文字输出相关函数</u>	73
<u>getfont</u>	74
<u>LOGFONT 结构体</u>	75
<u>outtext</u>	79
<u>outtextrect</u>	80
<u>outtextxy</u>	81
<u>setfont</u>	83
<u>settextjustify</u>	86
<u>textheight</u>	87
<u>textwidth</u>	88
<u>图像处理相关函数</u>	89
<u>getimage</u>	90
<u>IMAGE 对象</u>	91
<u>imagefilter blurring</u>	92
<u>putimage</u>	93
<u>putimage alphablend</u>	96
<u>putimage transparent</u>	97
<u>putimage alphantransparent</u>	98
<u>三元光栅操作码</u>	99
<u>鼠标相关函数</u>	106

FlushMouseMsgBuffer	107
GetMouseMsg	108
GetMousePos	109
MouseHit	110
ShowMouse	111
MOUSEMSG 结构体	112
时间函数	114
API Sleep	115
delay	116
delay ms	117
delay fps	118
delay jfps	119
fclock	120
数学函数	121
其它函数	122
BeginBatchDraw	123
EndBatchDraw	124
FlushBatchDraw	125
GetFPS	126
GetHWnd	127
InputBoxGetLine	128
keystate	129
random	130
randomf	131
randomize	132
示例程序	133
基础级别示例	133
字符阵	133
鼠标操作范例	134
彩虹	135
初等级别示例	136
星空	136
滚动字幕	138
下雪	139
中等级别示例	141
变幻线	141
高等级别示例	145
俄罗斯方块	145
与 Borland BGI 绘图库的兼容情况	150
联系我们	151

基本说明

许多学编程的都是从 C 语言开始入门的，而目前的现状是：

1. 有些学校以 Turbo C 为环境讲 C 语言，只是 Turbo C 的环境实在太老了，复制粘贴都很不方便。

2. 有些学校直接拿 VC 来讲 C 语言，因为 VC 的编辑和调试环境都很优秀，并且 VC 有适合教学的免费版本。可惜在 VC 下只能做一些文字性的练习题，想画条直线画个圆都很难，还要注册窗口类、建消息循环等等，初学者会受严重打击的。初学编程想要绘图就得用 TC，很无奈。

3. 还有计算机图形学，这门课程的重点是绘图算法，而不是 Windows 编程。所以，许多老师不得不用 TC 教学，因为 Windows 绘图太复杂了，会偏离教学的重点。新的图形学的书有不少是用的 OpenGL，可是门槛依然很高。

所以，我想给大家一个更好的学习平台，就是 VC 方便的开发平台和 TC 简单的绘图功能，于是就有了这个 EGE 库。如果您刚开始学 C 语言，或者您是一位教 C 语言的老师，再或者您在教计算机图形学，那么这个库一定会让您兴奋的。

以下是该帮助所涉及的内容：

- [安装](#)
- [超简单的使用预览](#)
- [库函数说明](#)
- [示例程序](#)
- [与 Borland BGI 绘图库的兼容情况](#)
- [联系我们](#)

Version: 20110401 V0.3.6 Release

安装

系统支持

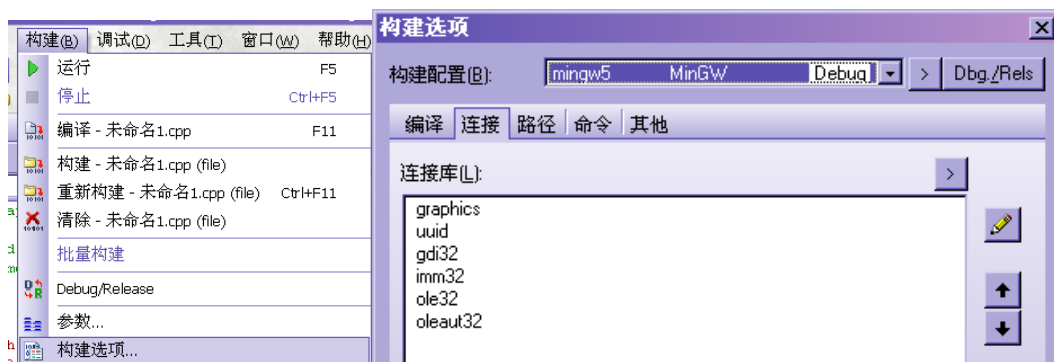
操作系统版本: Windows 98 及以上系统。

编译环境版本: Visual C++ 6.0 / 2008 / 2010 , mingw3.4.5 (CFree4.0/CFree5.0)。

安装

方法一: 将 include 和 lib 文件夹下的文件分别拷贝到 VC (或者 CFree) 的安装目录下对应的 include 和 lib 文件夹内即可。

如果你用的是 CFree5.0 (对应 libgraphics.a, lib 目录在安装目录下的 mingw 文件夹里), 还要设置编译环境, 如图加入 graphics, uuid, gdi32, imm32, ole32, oleaut32 库:



最后, 点一下“连接库”正右边的“>”按钮, 选择保存为默认, 然后关闭文件, 重新打开, 这时就能自动应用到新的文件了, 不用以后再慢慢设置了。

方法二: 把 h 和相应的 lib 文件, 直接复制到你的工程所在目录下, 与你的 CPP 同目录即可。本方法仅限于使用 VC 的。

卸载

由于安装程序并不改写注册表, 因此您在“添加删除程序”中不会看到 EGE 的卸载项。如需卸载, 请直接删除你复制进去的头文件和 lib 文件即可。

文件列表说明

下载的压缩包文件列表及对应说明如下:

include<文件夹>

graphics.h 程序需要引用的头文件

lib<文件夹>

graphics.lib VC6 版本库文件

graphics05.lib VC2005(不带 SP1)版本库文件

graphics08.lib VC2008 / VC2010 版本库文件

libgraphics.a mingw3.4.5 静态库文件, 用于 CFree

ege.pdf 本库的说明文档

项目依赖

该绘图库采用静态链接方式, 不会为您的程序增加任何额外的 DLL 依赖项。

超简单的使用预览

使用上，基本和 Turbo C 没太大区别。启动 Visual C++，创建一个控制台项目（Win32 Console Application），然后引用 graphics.h 头文件就可以了。看一个画圆的例子吧：

```
#include "graphics.h"    // 就是需要引用这个图形库
int main()
{
    initgraph(640, 480); // 初始化为640*480大小的窗口，这里和TC 略有区别
    circle(200, 200, 100); // 画圆，圆心(200, 200)，半径100
    getch();              // 等待用户按键，按任意键继续
    closegraph();         // 关闭图形界面
    return 0;
}
```

呵呵，很简单吧。

不过还是有不少区别的，比如颜色上，TC 只有 16 色，而这个库支持了真彩色。还有，这个库增加了鼠标、键盘扩展、双缓冲、批量绘图、读取图片（点阵或矢量）等功能。

另外，**如果你希望运行时完全不带控制台窗口**，如果在VC下，**默认就是去掉控制台窗口**。但如果你需要显示出来的话，你可以把#define SHOW_CONSOLE写在#include "graphics.h"的前面，例如：

```
#define SHOW_CONSOLE
#include "graphics.h"
int main()
{
    initgraph(640, 480);
    getch();
    closegraph();
    return 0;
}
```

但，如果你用的是 CFree，那把 main 改成 WinMain 即可实现控制台窗口的隐藏，如下替换一下即可：

```
#include "graphics.h"
int WinMain() // 这里在graphics.h里已经定义了宏自动把它展开成合法的声明，以减免声明的麻烦
{
    initgraph(640, 480);
    getch();
    closegraph();
    return 0;
}
```

当然，你在 VC 上也直接这样写也可以，因为也会自动帮你替换回 main，所以直接写 WinMain 可以两边通用。另外，默认会显示一个 EGE 的 logo，如果你不喜欢，**可以在 initgraph 前面调用 setinitmode**，只要调用一次，logo 就不会显示。

库函数说明

该函数库共分以下几大类函数：

- [绘图环境相关函数](#)
- [颜色表示及相关函数](#)
- [绘制图形相关函数](#)
- [文字输出相关函数](#)
- [图像处理相关函数](#)
- [鼠标相关函数](#)
- [时间函数](#)
- [其它函数](#)

绘图环境相关函数

相关函数和数据如下：

函数或数据	描述
<u>initgraph</u>	初始化绘图环境。
<u>cleardevice</u>	清除屏幕。
<u>closegraph</u>	关闭图形环境。
<u>getviewport</u>	获取当前视图信息。
<u>setviewport</u>	设置当前视图。
<u>clearviewport</u>	清空视图。
<u>setactivepage</u>	设置当前绘图页。
<u>setvisualpage</u>	设置显示页，把页面内容输出到窗口的页。
<u>window_getviewport</u>	获取当前窗口可见部分。
<u>window_setviewport</u>	设置窗口可见部分。

initgraph

这个函数用于初始化绘图环境。

```
void initgraph(  
    int Width,  
    int Height,  
    int Flag = NULL  
);
```

```
void initgraph(  
    int* gdriver,  
    int* gmode,  
    char* path
```

```
); // 兼容 Borland C++ 3.1 的重载，默认 640x480@24bit。gdriver使用TRUECOLORSIZE也可指定大小，  
这时gmode的低16位表示width，高16位表示height。
```

参数：

Width

绘图环境的宽度。如果为-1，则使用屏幕的宽度

Height

绘图环境的高度。如果为-1，则使用屏幕的高度

Style

绘图环境的样式，默认为 NULL。可为以下值：

值	含义
SHOWCONSOLE	表示可以保留原控制台窗口。

此参数在本库无效果

返回值：

（无）

示例：

参见 cleardevice

cleardevice

这个函数用于清除画面内容。具体的，是用当前背景色清空画面。

```
void cleardevice(  
    IMAGE* pimg = NULL  
);
```

参数：

pimg

指定要清除的 IMAGE，可选参数。如果不填本参数，则清除屏幕。

返回值：

（无）

示例：

```
#include "graphics.h"  
int main()  
{  
    initgraph(640, 480);  
    circle(200, 200, 100);  
    getch();  
    cleardevice();  
    getch();  
    closegraph();  
    return 0;  
}
```

closegraph

这个函数用于关闭图形环境。

```
void closegraph();
```

参数:

(无)

返回值:

(无)

示例:

参见 cleardevice

getviewport

这个函数用于获取当前视图信息。

```
void getviewport(  
    int *pleft,  
    int *ptop,  
    int *pright,  
    int *pbottom,  
    int *pclip = NULL,  
    IMAGE* pimg = NULL  
);
```

参数:

pleft

返回当前视图的左部 x 坐标。

ptop

返回当前视图的上部 y 坐标。

pright

返回当前视图的右部 x 坐标。

pbottom

返回当前视图的下部 y 坐标。

pclip

返回当前视图的裁剪标志。

pimg

详见 setviewport 的说明

返回值:

(无)

示例:

参见 setviewport

setviewport

这个函数用于设置当前视图。并且，将坐标原点移动到新的视图的 (0, 0) 位置。

```
void setviewport(  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int clip = 1,  
    IMAGE* pimg = NULL  
);
```

参数：

left

视图的左部 x 坐标。

top

视图的上部 y 坐标。(left, top) 将成为新的原点。

right

视图的右部 x 坐标。

bottom

视图的下部 y 坐标。(right-1, bottom-1) 是视图的右下角坐标。

clip

视图的裁剪标志。如果为真，所有超出视图区域的绘图都会被裁剪掉。

返回值：

(无)

示例：

```
#include "graphics.h"  
int main()  
{  
    initgraph(640, 480);  
    setviewport(100, 100, 200, 200);  
    rectangle(0, 0, 200, 200);  
    getch();  
    closegraph();  
    return 0;  
}
```

注意：

右端点和下端点取不到，上端点和左端点能取到。另外，这函数的最后一个参数为一个 IMAGE* 的指针，是一个可选参数，如果不填，则设置到当前页。如果填上，则设置到指定的 IMAGE。

clearviewport

这个函数用于清空视图。相当于对视图区进行 cleardevice。

```
void clearviewport(  
    IMAGE* pimg = NULL  
);
```

参数:

pimg

见 setviewport

返回值:

(无)

示例:

(无)

setactivepage

这个函数用于设置当前绘图页。

```
void setactivepage(int page);
```

参数:

page

绘图页，范围从，范围从0-3，越界会导致程序错误。默认值为0

返回值:

(无)

示例:

(无)

setinitmode

这个函数用于设置初始化图形的选项和模式。

```
void setinitmode(int mode, int x = CW_USEDEFAULT, int y = CW_USEDEFAULT);
```

参数:

mode

初始化模式，是二进制组合的值。如果为 INIT_DEFAULT 表示使用默认值。

可以使用的值的组合:

INIT_NOBORDER 为无边框窗口

INIT_CHILD 为子窗口（需要使用 attachHWND 指定要依附的父窗口，此函数不另说明）

INIT_TOPMOST 使窗口总在最前

INIT_WITHLOGO 使 initgraph 的时候显示开场动画 logo

x, y

初始化时窗口左上角在屏幕的坐标，默认为系统分配。

返回值:

（无）

说明:

本函数只能在 initgraph 前调用。

setvisualpage

这个函数用于设置当前显示页，显示页是输出到窗口的页。

```
void setvisualpage(int page);
```

参数：

page

绘图页，范围从，范围从0-3，越界会导致程序错误。默认值为0

返回值：

（无）

示例：

（无）

说明：

这个函数与 setactivepage 配合使用，可以模拟实现双缓冲绘图，不过库本身也提供了批量绘图机制，实现时已经使用双缓冲，必要性似乎不大，主要除了兼容老程序以外，也可以当缓冲区使用。

window_getviewport

这个函数用于获取当前窗口可见区域。

```
void window_getviewport(  
    int* left,  
    int* top,  
    int* right,  
    int* bottom  
);
```

参数:

left

返回可见区域的左部 x 坐标。

top

返回可见区域的上部 y 坐标。

right

返回可见区域的右部 x 坐标。

bottom

返回可见区域的下部 y 坐标。(right, bottom) 是可见区域的右下角坐标。

返回值:

(无)

示例:

(无)

window_setviewport

这个函数用于设置当前窗口可见区域。

```
void window_setviewport(  
    int left,  
    int top,  
    int right,  
    int bottom  
);
```

参数:

left

可见区域的左部 x 坐标。

top

可见区域的上部 y 坐标。(left, top) 将成为新的原点。

right

可见区域的右部 x 坐标。

bottom

可见区域的下部 y 坐标。(right-1, bottom-1) 是视图的右下角坐标。

返回值:

(无)

示例:

```
#include "graphics.h"  
int main()  
{  
    initgraph(640, 480);  
    window_setviewport(100, 100, 400, 400);  
    rectangle(0, 0, 200, 200);  
    getch();  
    closegraph();  
    return 0;  
}
```

颜色表示及相关函数

相关函数和数据如下：

函数或数据	描述
颜色表示	介绍颜色表示方法。
getbkcolor	获取当前绘图背景色。
getcolor	获取当前绘图前景色
GetBValue	返回指定颜色中的蓝色值。
GetGValue	返回指定颜色中的绿色值。
GetRValue	返回指定颜色中的红色值。
HSLtoRGB	转换 HSL 颜色为 RGB 颜色。
HSVtoRGB	转换 HSV 颜色为 RGB 颜色。
RGB	通过红、绿、蓝颜色分量合成颜色。
RGBtoGRAY	转换 RGB 颜色为 灰度颜色。
RGBtoHSL	转换 RGB 颜色为 HSL 颜色。
RGBtoHSV	转换 RGB 颜色为 HSV 颜色。
setbkcolor	设置当前绘图背景色。
setbkcolor_f	设置清屏时所用的背景色。
setbkmode	设置输出文字时的背景模式。
setcolor	设置当前绘图前景色。
setfontbkcolor	设置当前文字背景色。

特殊说明：

这里部分函数的最后一个参数为一个 IMAGE*的指针，是一个可选参数，如果不填，则绘画到当前页。如果填上，则设置或者绘画到指定的 IMAGE。

getbkcolor

这个函数用于获取当前绘图背景色。

```
COLORREF getbkcolor(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回当前绘图背景色。

示例:

(无)

GetBValue

GetBValue 宏用于返回指定颜色中的蓝色值。

```
BYTE GetBValue(COLORREF rgb);
```

参数:

rgb

指定的颜色。

返回值:

指定颜色中的蓝色值，值的范围 0~255。

示例:

(无)

注:

GetBValue 宏在 Windows SDK 中定义。

getcolor

这个函数用于获取当前绘图前景色

```
COLORREF getcolor(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回当前的前景颜色。

示例:

(无)

GetGValue

GetGValue 宏用于返回指定颜色中的绿色值。

```
BYTE GetGValue(COLORREF rgb);
```

参数:

rgb

指定的颜色。

返回值:

指定颜色中的绿色值，值的范围 0~255。

示例:

(无)

注:

GetGValue 宏在 Windows SDK 中定义。

GetRValue

GetRValue 宏用于返回指定颜色中的红色值。

```
BYTE GetRValue(COLORREF rgb);
```

参数:

rgb

指定的颜色。

返回值:

指定颜色中的红色值，值的范围 0~255。

示例:

(无)

注:

GetRValue 宏在 Windows SDK 中定义。

HSLtoRGB

该函数用于转换 HSL 颜色为 RGB 颜色。

```
COLORREF HSLtoRGB(  
    float H,  
    float S,  
    float L  
);
```

参数：

H

原 HSL 颜色模型的 Hue(色相) 分量, $0 \leq H < 360$ 。

S

原 HSL 颜色模型的 Saturation(饱和度) 分量, $0 \leq S \leq 1$ 。

L

原 HSL 颜色模型的 Lightness(亮度) 分量, $0 \leq L \leq 1$ 。

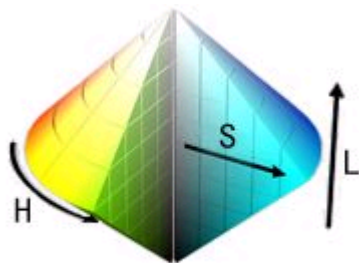
返回值：

对应的 RGB 颜色。

说明：

HSL 又称 HLS。

HSL 的颜色模型如图所示：



HSL 颜色空间模型

H 是英文 Hue 的首字母, 表示色相, 即组成可见光谱的单色。红色在 0 度, 绿色在 120 度, 蓝色在 240 度, 以此方向过渡。

S 是英文 Saturation 的首字母, 表示饱和度, 等于 0 时为灰色。在最大饱和度 1 时, 具有最纯的色光。

L 是英文 Lightness 的首字母, 表示亮度, 等于 0 时为黑色, 等于 0.5 时是色彩最鲜明的状态, 等于 1 时为白色。

示例：

请参见示例程序中的“彩虹”。

HSVtoRGB

该函数用于转换 HSV 颜色为 RGB 颜色。

```
COLORREF HSVtoRGB(  
    float H,  
    float S,  
    float V  
);
```

参数：

H

原 HSV 颜色模型的 Hue(色相) 分量, $0 \leq H < 360$ 。

S

原 HSV 颜色模型的 Saturation(饱和度) 分量, $0 \leq S \leq 1$ 。

V

原 HSV 颜色模型的 Value(明度) 分量, $0 \leq V \leq 1$ 。

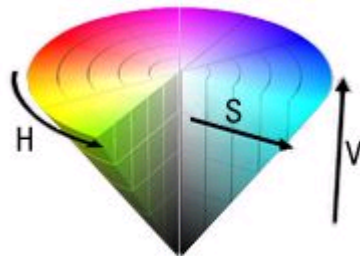
返回值：

对应的 RGB 颜色。

说明：

HSV 又称 HSB。

HSV 的颜色模型如图所示：



HSV 颜色空间模型

H 是英文 Hue 的首字母, 表示色相, 即组成可见光谱的单色。红色在 0 度, 绿色在 120 度, 蓝色在 240 度, 以此方向过渡。

S 是英文 Saturation 的首字母, 表示饱和度, 等于 0 时为灰色。在最大饱和度 1 时, 每一色相具有最纯的色光。

V 是英文 Value 的首字母, 表示明度, 等于 0 时为黑色, 在最大明度 1 时, 是色彩最鲜明的状态。

示例：

HSV 颜色模型类似于 HSL, 示例程序中的“彩虹”是 HSL 模型的操作范例, 可以参考。

RGB

RGB 宏用于通过红、绿、蓝颜色分量合成颜色。

```
COLORREF RGB(  
    BYTE byRed,      // 颜色的红色部分  
    BYTE byGreen,    // 颜色的绿色部分  
    BYTE byBlue      // 颜色的蓝色部分  
);
```

参数：

byRed

颜色的红色部分，取值范围：0~255。

byGreen

颜色的绿色部分，取值范围：0~255。

byBlue

颜色的蓝色部分，取值范围：0~255。

返回值：

返回合成的颜色。

说明：

可以通过 [GetRValue](#)、[GetGValue](#)、[GetBValue](#) 宏从颜色中分离出红、绿、蓝颜色分量。详见 [GetRValue](#)、[GetGValue](#)、[GetBValue](#)。

示例：

（无）

注：

RGB 宏在 Windows SDK 中定义。

RGBtoGRAY

该函数用于返回与指定颜色对应的灰度值颜色。

```
COLORREF RGBtoGRAY(  
    COLORREF rgb  
);
```

参数：

rgb

原 RGB 颜色。

返回值：

对应的灰度颜色。

示例：

（无）

RGBtoHSL

该函数用于转换 RGB 颜色为 HSL 颜色。

```
void RGBtoHSL(  
    COLORREF rgb,  
    float *H,  
    float *S,  
    float *L  
);
```

参数：

rgb

原 RGB 颜色。

H

用于返回 HSL 颜色模型的 Hue(色相) 分量， $0 \leq H < 360$ 。

S

用于返回 HSL 颜色模型的 Saturation(饱和度) 分量， $0 \leq S \leq 1$ 。

L

用于返回 HSL 颜色模型的 Lightness(亮度) 分量， $0 \leq L \leq 1$ 。

返回值：

(无)

说明：

HSL 详见 [HSLtoRGB](#)。

示例：

(无)

RGBtoHSV

该函数用于转换 RGB 颜色为 HSV 颜色。

```
void RGBtoHSV(  
    COLORREF rgb,  
    float *H,  
    float *S,  
    float *V  
);
```

参数:

rgb

原 RGB 颜色。

H

用于返回 HSV 颜色模型的 Hue(色相) 分量, $0 \leq H < 360$ 。

S

用于返回 HSV 颜色模型的 Saturation(饱和度) 分量, $0 \leq S \leq 1$ 。

V

用于返回 HSV 颜色模型的 Value(明度) 分量, $0 \leq V \leq 1$ 。

返回值:

(无)

说明:

HSV 详见 [HSVtoRGB](#)。

示例:

(无)

setbkcolor

这个函数用于设置当前背景色。并且会把当前图片上是原背景色的像素,转变为新的背景色。

```
void setbkcolor(  
    COLORREF color,  
    IMAGE* pimg = NULL  
);
```

参数:

color

指定要设置的背景颜色。注意, 该设置会同时影响文字背景色。

返回值:

(无)

示例:

(无)

setbkcolor_f

这个函数用于设置当前背景色。仅设置 cleardevice 时所使用的颜色，不立即生效，需要等 cleardevice 调用。

```
void setbkcolor_f(  
    COLORREF color,  
    IMAGE* pimg = NULL  
);
```

参数:

color

指定要设置的背景颜色。注意，该设置会同时影响文字背景色。

返回值:

(无)

示例:

(无)

setbkmode

这个函数用于设置输出文字时的背景模式。

```
void setbkmode(  
    int iBkMode,  
    IMAGE* pimg = NULL  
);
```

参数：

iBkMode

指定输出文字时的背景模式，可以是以下值：

值	描述
OPAQUE	背景用当前背景色填充（默认）。
TRANSPARENT	背景是透明的。

返回值：

（无）

示例：

（无）

setcolor

这个函数用于设置绘图前景色。

```
void setcolor(  
    COLORREF color,  
    IMAGE* pimg = NULL  
);
```

参数：

color

要设置的前景颜色。

返回值：

（无）

示例：

（无）

setfontbkcolor

这个函数用于设置绘图前景色。

```
void setfontbkcolor(  
    COLORREF color,  
    IMAGE* pimg = NULL  
);
```

参数：

color

要设置的文字背景颜色。

返回值：

（无）

示例：

（无）

颜色表示

设置绘图色有以下几种办法：

1. 用 16 进制的颜色表示，形式为：

0xbbggrr (bb=蓝，gg=绿，rr=红)

2. 用预定义颜色，如下：

常量	值	颜色	常量	值	颜色
BLACK	0	黑	DARKGRAY	0x545454	深灰
BLUE	0xA80000	蓝	LIGHTBLUE	0xFC5454	亮蓝
GREEN	0x00A800	绿	LIGHTGREEN	0x54FC54	亮绿
CYAN	0xA8A800	青	LIGHTCYAN	0xFCFC54	亮青
RED	0x0000A8	红	LIGHTRED	0x5454FC	亮红
MAGENTA	0xA800A8	紫	LIGHTMAGENTA	0xFC54FC	亮紫
BROWN	0x0054A8	棕	YELLOW	0x54FCFC	黄
LIGHTGRAY	0xA8A8A8	浅灰	WHITE	0xFCFCFC	白

3. 用 [RGB](#) 宏合成颜色。详见 [RGB](#)。

4. 用 [HSLtoRGB](#)、[HSVtoRGB](#) 转换其他色彩模型到 RGB 颜色。详见 [HSLtoRGB](#)、[HSVtoRGB](#)。

示例：

以下是部分设置前景色的方法：

```
setcolor(0xff0000);
```

```
setcolor(BLUE);
```

```
setcolor(RGB(0, 0, 255));
```

```
setcolor(HSLtoRGB(240, 1, 0.5));
```

绘制图形相关函数

相关函数和数据如下：

函数或数据	描述
arc	画圆弧。
bar	画无边框填充矩形。
bar3d	画有边框三维填充矩形。
circle	画圆。
drawbezier	画 bezier 曲线
drawlines	画多条不连续线段
drawpoly	画多边形。
ellipse	画椭圆弧线。
fillellipse	画填充的椭圆。
fillpoly	画填充的多边形。
floodfill	填充区域。
getfillstyle	获取当前填充类型。（暂不支持）
getheight	获取绘图区的高度。
getlinestyle	获取当前线形。
getpixel	获取点的颜色。
getwidth	获取绘图区的宽度。
getx	获取当前 x 坐标。
gety	获取当前 y 坐标。
line	画线。
linerel	画线。
lineto	画线。
moverel	移动当前点。
moveto	移动当前点。
pieslice	画填充圆扇形。
putpixel	画点。
putpixels	画多个点。
rectangle	画空心矩形。
sector	画填充椭圆扇形。
setfillstyle	设置当前填充类型。
setlinestyle	设置当前线形。
setwritemode	设置绘图位操作模式。

特殊说明：

以下所有函数的坐标模式为，如果以有向线段表示的量，起点能取到，终点取不到。比如 line 函数，起点 x1,y1 能画上点，终点 x2,y2 不会画上点，请注意。类似的有 lineto, linerel, bar, bar3d, rectangle, ellipse, sector, 函数说明内也会对本段进行补充。

另外，这里每一个函数的最后一个参数均为一个 IMAGE* 的指针，是一个可选参数，如果不填，则绘画到当前页。如果填上，则绘画到指定的 IMAGE。

关于效率，如果打开了批量绘图，那以上所有函数的执行速度都会得到提升。不过，BeginBatchDraw 函数的执行需要花费较多时间（几十毫秒），建议如果需要，那请全局开启，不要局部使用，就是说最好在初始化，或者对时间效率不敏感的地方调用。

arc

这个函数用于画圆弧。边线颜色由 setcolor 函数决定

```
void arc(  
    int x,  
    int y,  
    int stangle,  
    int endangle,  
    int radius,  
    IMAGE* pimg = NULL  
);
```

参数:

x

圆弧的圆心 x 坐标。

y

圆弧的圆心 y 坐标。

stangle

圆弧的起始角的角度。

endangle

圆弧的终止角的角度。

radius

圆弧的半径。

返回值:

(无)

示例:

(无)

bar

这个函数用于画无边框填充矩形。其中，填充颜色由 `setfillstyle` 函数决定

```
void bar(  
    int left,  
    int top,  
    int right,  
    int bottom,  
    IMAGE* pimg = NULL  
);
```

参数:

`left`

矩形左部 x 坐标。

`top`

矩形上部 y 坐标。

`right`

矩形右部 x 坐标（该点取不到，实际右边界为 `right-1`）。

`bottom`

矩形下部 y 坐标（该点取不到，实际下边界为 `bottom-1`）。

返回值:

（无）

示例:

（无）

bar3d

这个函数用于画有边框三维填充矩形。其中，填充颜色由 `setfillstyle` 函数决定

```
void bar3d(  
    int left,  
    int top,  
    int right,  
    int bottom,  
    int depth,  
    bool topflag,  
    IMAGE* pimg = NULL  
);
```

参数：

`left`

矩形左部 x 坐标。

`top`

矩形上部 y 坐标。

`right`

矩形右部 x 坐标（该点取不到，实际右边界为 `right-1`）。

`bottom`

矩形下部 y 坐标（该点取不到，实际下边界为 `bottom-1`）。

`depth`

矩形深度。

`topflag`

为 `false` 时，将不画矩形的三维顶部。该选项可用来画堆叠的三维矩形。

返回值：

（无）

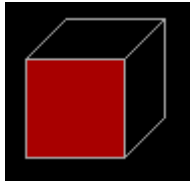
示例：

```
#include "graphics.h"
```

```
int main()  
{  
    initgraph(600, 400);  
    setfillstyle(RED);  
    bar3d(100, 100, 150, 150, 20, 1);  
}
```

```
    getch();  
    return 0;  
}
```

示例效果：



circle

这个函数用于画圆。此圆是空心的，不填充，而边线颜色由 `setcolor` 函数决定

```
void circle(  
    int x,  
    int y,  
    int radius,  
    IMAGE* pimg = NULL  
);
```

参数：

x

圆的圆心 x 坐标。

y

圆的圆心 y 坐标。

radius

圆的半径。

返回值：

（无）

示例：

（无）

drawbezier

这个函数用于画 bezier 曲线。边线颜色由 setcolor 函数决定

```
void drawbezier(  
    int numpoints,  
    const int *polypoints,  
    IMAGE* pimg = NULL  
);
```

参数:

numpoints

多边形点的个数，需要是被3除余1的数，如果不是，则忽略最后面若干个点。

polypoints

每个点的坐标（依次两个分别为 x, y），数组元素个数为 numpoints * 2。

每一条 bezier 曲线由两个端点和两个控制点组成，相邻两条则共用端点。

返回值:

（无）

示例:

（无）

drawlines

这个函数用于画多条线段。边线颜色由 setcolor 函数决定

```
void drawpoly(  
    int numlines,  
    const int *polypoints,  
    IMAGE* pimg = NULL  
);
```

参数:

numlines

线段数目。

polypoints

每个点的坐标（依次两个分别为 x, y），数组元素个数为 numlines * 4。

每两个点画一线段。

返回值:

（无）

示例:

（无）

drawpoly

这个函数用于画多边形。边线颜色由 setcolor 函数决定

```
void drawpoly(  
    int numpoints,  
    const int *polypoints,  
    IMAGE* pimg = NULL  
);
```

参数:

numpoints

多边形点的个数。

polypoints

每个点的坐标（依次两个分别为 x, y），数组元素个数为 `numpoints * 2`。

该函数并不会自动连接多边形首尾。如果需要画封闭的多边形，请将最后一个点设置为与第一点相同。

返回值:

（无）

示例:

（无）

ellipse

这个函数用于画椭圆弧线。边线颜色由 setcolor 函数决定

```
void ellipse(  
    int x,  
    int y,  
    int stangle,  
    int endangle,  
    int xradius,  
    int yradius,  
    IMAGE* pimg = NULL  
);
```

参数:

x

椭圆弧线的圆心 x 坐标。

y

椭圆弧线的圆心 y 坐标。

stangle

椭圆弧线的起始角的角度。

endangle

椭圆弧线的终止角的角度。

xradius

椭圆弧线的 x 轴半径。

yradius

椭圆弧线的 y 轴半径。

返回值:

(无)

示例:

(无)

fillellipse

这个函数用于画填充的椭圆。边线颜色由 `setcolor` 函数决定，填充颜色由 `setfillstyle` 函数决定。

```
void fillellipse(  
    int x,  
    int y,  
    int xradius,  
    int yradius,  
    IMAGE* pimg = NULL  
);
```

参数：

x

椭圆的圆心 x 坐标。

y

椭圆的圆心 y 坐标。

xradius

椭圆的 x 轴半径。

yradius

椭圆的 y 轴半径。

返回值：

（无）

示例：

（无）

fillpoly

这个函数用于画填充的多边形。边线颜色由 `setcolor` 函数决定，填充颜色由 `setfillstyle` 函数决定。

```
void fillpoly(  
    int numpoints,  
    const int *polypoints,  
    IMAGE* pimg = NULL  
);
```

参数：

`numpoints`

多边形点的个数。

`polypoints`

每个点的坐标，数组元素个数为 `numpoints * 2`。

该函数会自动连接多边形首尾。

返回值：

(无)

示例：

(无)

说明：

如果这个多边形发生自相交，那么自交次数为奇数的区域则不填充，偶数次的填充，不自交就是偶数次。不过这样说明相信非常难理解，以下给个例子：

```
#include "graphics.h"  
int main()  
{  
    initgraph(600, 400);  
    setfillstyle(RED);  
    int pt[] = {  
        0, 0,  
        100, 0,  
        100, 100,  
        10, 10,  
        90, 10,  
        0, 100,  
    };  
    fillpoly(6, pt);  
    getch();  
    return 0;  
}
```

```
}
```

运行结果：

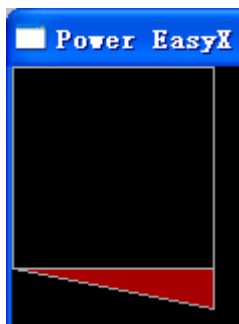


第二个例子：

```
#include "graphics.h"
```

```
int main()
{
    initgraph(600, 400);
    setfillstyle(RED);
    int pt[] = {
        0, 0,
        100, 0,
        100, 100,
        0, 100,
        0, 0,
        100, 0,
        100, 120,
        0, 100,
    };
    fillpoly(8, pt);
    getch();
    return 0;
}
```

运行结果：



floodfill

这个函数使用 `setfillstyle` 设置的填充方式对区域进行填充。填充颜色由 `setfillstyle` 函数决定。

```
void floodfill(  
    int x,  
    int y,  
    int border,  
    IMAGE* pimg = NULL  
);
```

参数：

x

填充的起始点 x 坐标。

y

填充的起始点 y 坐标。

border

填充的边界颜色。填充动作在该颜色围成的区域内填充。如果该颜色围成的区域不封闭，那么将使全屏幕都填充上。

返回值：

（无）

示例：

（无）

getheight

这个函数用于获取图片高度。

```
int getheight(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回图片高度。

示例:

(无)

getlinestyle

这个函数用于获取当前线形。

```
void getlinestyle(  
    int *plinestyle,  
    WORD *pupattern = NULL,  
    int *pthickness = NULL,  
    IMAGE* pimg = NULL  
);
```

参数：

plinestyle

返回当前线型。详见 setlinestyle。

pupattern

返回当前自定义线形数据。

pthickness

返回当前线形宽度。

返回值：

（无）

示例：

（无）

getpixel

这个函数用于获取点的颜色。

```
COLORREF getpixel(  
    int x,  
    int y  
    IMAGE* pimg = NULL  
);
```

参数：

x

要获取颜色的 x 坐标。

y

要获取颜色的 y 坐标。

返回值：

指定点的颜色。

示例：

（无）

其它说明：另有高速版的 `getpixel_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

getwidth

该函数用于获取图片宽度。

```
int getwidth(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回图片宽度。

示例:

(无)

getx

这个函数用于获取当前 x 坐标。

```
int getx(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回当前 x 坐标。

示例:

(无)

gety

这个函数用于获取当前 y 坐标。

```
int gety(  
    IMAGE* pimg = NULL  
);
```

参数:

(无)

返回值:

返回当前 y 坐标。

示例:

(无)

line

这个函数用于画线。

```
void line(  
    int x1,  
    int y1,  
    int x2,  
    int y2,  
    IMAGE* pimg = NULL  
);
```

参数：

x1

线的起始点的 x 坐标。

y1

线的起始点的 y 坐标。

x2

线的终止点的 x 坐标（该点本身画不到）。

y2

线的终止点的 y 坐标（该点本身画不到）。

返回值：

（无）

示例：

（无）

其它说明：另有高速版的 `line_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

linere1

这个函数用于画线。

```
void linere1(  
    int dx,  
    int dy,  
    IMAGE* pimg = NULL  
);
```

参数:

dx

从“当前点”cx 开始画线，沿 x 轴偏移 dx，终点为 cx+dx（终点本身画不到）。

dy

从“当前点”cy 开始画线，沿 y 轴偏移 dy，终点为 cy+dy（终点本身画不到）。

返回值:

（无）

示例:

（无）

其它说明：另有高速版的 `linere1_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

lineto

这个函数用于画线。

```
void lineto(  
    int x,  
    int y,  
    IMAGE* pimg = NULL  
);
```

参数:

x

从“当前点”开始画线，终点横坐标为 x （终点本身画不到）。

y

从“当前点”开始画线，终点纵坐标为 y （终点本身画不到）。

返回值:

（无）

示例:

（无）

其它说明：另有高速版的 `lineto_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

moverel

这个函数用于移动当前点。有些绘图操作会从“当前点”开始，这个函数可以设置该点。

```
void moverel(  
    int dx,  
    int dy,  
    IMAGE* pimg = NULL  
);
```

参数:

dx

将当前点沿 x 轴移动 dx。

dy

将当前点沿 y 轴移动 dy。

返回值:

(无)

示例:

(无)

moveto

这个函数用于移动当前点。有些绘图操作会从“当前点”开始，这个函数可以设置该点。

```
void moveto(  
    int x,  
    int y  
    IMAGE* pimg = NULL  
);
```

参数：

x

新的当前点 x 坐标。

y

新的当前点 y 坐标。

返回值：

（无）

示例：

（无）

pieslice

这个函数用于画填充圆扇形。

```
void pieslice(  
    int x,  
    int y,  
    int stangle,  
    int endangle,  
    int radius,  
    IMAGE* pimg = NULL  
);
```

参数：

x

圆扇形的圆心 x 坐标。

y

圆扇形的圆心 y 坐标。

stangle

圆扇形的起始角的角度。

endangle

圆扇形的终止角的角度。

radius

圆扇形的半径。

返回值：

（无）

示例：

（无）

putpixel

这个函数用于画点。

```
void putpixel(  
    int x,  
    int y,  
    COLORREF color,  
    IMAGE* pimg = NULL  
);
```

参数：

x

点的 x 坐标。

y

点的 y 坐标。

color

点的颜色。

返回值：

（无）

示例：

（无）

其它说明：另有高速版的 `putpixel_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

putpixels

这个函数用于画多个点。

```
void putpixels(  
    int nPoint,  
    int* pPoints,  
    IMAGE* pimg = NULL  
);
```

参数：

nPoint

点的数目。

pPoints

指向点的描述的指针，一个 int 型的数组，依次每三个 int 描述一个点：第一个为 x 坐标，第二个为 y 坐标，第三个为颜色值。

返回值：

（无）

示例：

（无）

其它说明：另有高速版的 `putpixels_f` 函数，参数一样，作用一样，但不进行相对坐标变换和边界检查（如果越界绘图，要么画错地方，要么程序结果莫名其妙，甚至直接崩溃），并且必须在批量绘图模式下才能使用，否则将发生不可预知的结果。

rectangle

这个函数用于画空心矩形。

```
void rectangle(  
    int left,  
    int top,  
    int right,  
    int bottom,  
    IMAGE* pimg = NULL  
);
```

参数:

left

矩形左部 x 坐标。

top

矩形上部 y 坐标。

right

矩形右部 x 坐标。

bottom

矩形下部 y 坐标。

返回值:

(无)

示例:

(无)

sector

这个函数用于画填充椭圆扇形。

```
void sector(  
    int x,  
    int y,  
    int stangle,  
    int endangle,  
    int xradius,  
    int yradius,  
    IMAGE* pimg = NULL  
);
```

参数：

x

椭圆扇形的圆心 x 坐标。

y

椭圆扇形的圆心 y 坐标。

stangle

椭圆扇形的起始角的角度。

endangle

椭圆扇形的终止角的角度。

xradius

椭圆扇形的 x 轴半径。

yradius

椭圆扇形的 y 轴半径。

返回值：

（无）

示例：

（无）

setfillstyle

这个函数用于设置当前填充类型。该函数的自定义填充部分尚不支持。

```
void setfillstyle(  
    COLORREF color,  
    int pattern = SOLID_FILL,  
    const char *pupattern = NULL,  
    IMAGE* pimg = NULL  
);
```

参数：

color

填充颜色。

pattern

填充类型，可以是以下宏或值：

宏	值	含义
NULL_FILL	1	不填充
SOLID_FILL	2	固实填充

pupattern

指定图案填充时的样式，目前无作用。

返回值：

（无）

示例：

设置蓝色固实填充：

```
setfillstyle(BLUE);
```

setlinestyle

这个函数用于设置当前线形。

```
void setlinestyle(  
    int linestyle,  
    WORD upattern = NULL,  
    int thickness = 1,  
    IMAGE* pimg = NULL  
);
```

参数：

linestyle

线型，可以是以下值：

值	含义
PS_SOLID	线形为实线。
PS_DASH	线形为：-----
PS_DOT	线形为：.....
PS_DASHDOT	线形为：- . - . - . - . - .
PS_DASHDOTDOT	线形为：- . . - . . - . . - . .
PS_NULL	线形为不可见。
PS_USERSTYLE	线形样式是自定义的，依赖于 upattern 参数。

upattern

自定义线形数据。

自定义规则：该数据为 WORD 类型，共 16 个二进制位，每位为 1 表示画线，为 0 表示空白。从低位到高位表示从起始到终止的方向。

仅当线型为 PS_USERSTYLE 时该参数有效。

thickness

线形宽度。

返回值：

（无）

示例：

设置线形为点划线： setlinestyle(PS_DASHDOT);

设置线形为宽度 3 像素的虚线： setlinestyle(PS_DASH, NULL, 3);

setwritemode

这个函数用于设置绘图位操作模式。

```
void setwritemode(  
    int mode.  
    IMAGE* pimg = NULL  
);
```

参数：

mode

二元光栅操作码（即位操作模式），支持全部的 16 种二元光栅操作码，罗列如下：

位操作模式	描述
R2_BLACK	绘制出的像素颜色 = 黑色
R2_COPYPEN	绘制出的像素颜色 = 当前颜色（默认）
R2_MASKNOTPEN	绘制出的像素颜色 = 屏幕颜色 AND (NOT 当前颜色)
R2_MASKPEN	绘制出的像素颜色 = 屏幕颜色 AND 当前颜色
R2_MASKPENNOUT	绘制出的像素颜色 = (NOT 屏幕颜色) AND 当前颜色
R2_MERGEOUTPEN	绘制出的像素颜色 = 屏幕颜色 OR (NOT 当前颜色)
R2_MERGEINPEN	绘制出的像素颜色 = 屏幕颜色 OR 当前颜色
R2_MERGEINOUTPEN	绘制出的像素颜色 = (NOT 屏幕颜色) OR 当前颜色
R2_NOP	绘制出的像素颜色 = 屏幕颜色
R2_NOT	绘制出的像素颜色 = NOT 屏幕颜色
R2_NOTCOPYPEN	绘制出的像素颜色 = NOT 当前颜色
R2_NOTMASKPEN	绘制出的像素颜色 = NOT (屏幕颜色 AND 当前颜色)
R2_NOTMERGEINPEN	绘制出的像素颜色 = NOT (屏幕颜色 OR 当前颜色)
R2_NOTXORPEN	绘制出的像素颜色 = NOT (屏幕颜色 XOR 当前颜色)
R2_WHITE	绘制出的像素颜色 = 白色
R2_XORPEN	绘制出的像素颜色 = 屏幕颜色 XOR 当前颜色

注：

1. AND / OR / NOT / XOR 为布尔位运算。
2. “屏幕颜色”指绘制所经过的屏幕像素点的颜色。
3. “当前颜色”是指通过 setcolor 设置的用于当前绘制的颜色。

返回值：

（无）

示例：

（无）

文字输出相关函数

相关函数和数据如下：

函数或数据	描述
getfont	获取当前字体样式。
LOGFONT 结构体	保存字体样式的结构体。
outtext	在当前位置输出字符串。
outtextrect	在指定矩形区域内输出字符串。
outtextxy	在指定位置输出字符串。
setfont	设置当前字体样式。
settextjustify	设置当前文字对齐方式。
textheight	获取字符串的高。
textwidth	获取字符串的宽。

getfont

这个函数用于获取当前字体样式。

```
void getfont(  
    LOGFONT *font  
    IMAGE* pimg = NULL  
);
```

参数：

font

指向 LOGFONT 结构体的指针。

返回值：

（无）

示例：

（无）

LOGFONT 结构体

这个结构体定义了字体的属性。

```
struct LOGFONT {
    LONG lfHeight;
    LONG lfWidth
    LONG lfEscapement;
    LONG lfOrientation;
    LONG lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    TCHAR lfFaceName[LF_FACESIZE];
};
```

成员

lfHeight

指定高度（逻辑单位）。
如果为正，表示指定的高度是字体的完整高度；如果为负，表示指定的高度不包含 tmInternalLeading 的高度。也就是说相同绝对值下，负的比正的会稍高一些；而实际输出的字体高度，正数时精确匹配，负数时，和实际略有偏差，绝对值比实际的略小。

lfWidth

指定字符的平均宽度（逻辑单位）。如果为 0，则比例自适应。

lfEscapement

字符串的书写角度，单位 0.1 度，默认为 0。

lfOrientation

每个字符的书写角度，单位 0.1 度，默认为 0。

lfWeight

字符的笔画粗细，范围 0~1000，0 表示默认粗细，使用数字或下表中定义的宏均可。

宏	粗细值
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500

FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

lfItalic

指定字体是否是斜体。

lfUnderline

指定字体是否有下划线。

lfStrikeOut

指定字体是否有删除线。

lfCharSet

指定字符集。以下是预定义的值：

ANSI_CHARSET

BALTIC_CHARSET

CHINESEBIG5_CHARSET

DEFAULT_CHARSET

EASTEUROPE_CHARSET

GB2312_CHARSET

GREEK_CHARSET

HANGUL_CHARSET

MAC_CHARSET

OEM_CHARSET

RUSSIAN_CHARSET

SHIFTJIS_CHARSET

SYMBOL_CHARSET

TURKISH_CHARSET

其中，OEM_CHARSET 表示字符集依赖本地操作系统。

DEFAULT_CHARSET 表示字符集基于本地操作系统。例如，系统位置是 English (United States), 字符集将设置为 ANSI_CHARSET。

lfOutPrecision

指定文字的输出精度。输出精度定义输出与所请求的字体高度、宽度、字符方向、行距、间距和字体类型相匹配必须达到的匹配程度。可以是以下值：

值	含义
OUT_DEFAULT_PRECIS	指定默认的映射行为。
OUT_DEVICE_PRECIS	当系统包含多个名称相同的字体时，指定设备字体。
OUT_OUTLINE_PRECIS	指定字体映射选择 TrueType 和其它的 outline-based 字体。
OUT_RASTER_PRECIS	当系统包含多个名称相同的字体时，指定光栅字体（即点阵字体）。

S	
OUT_STRING_PRECIS	这个值并不能用于指定字体映射，只是指定点阵字体枚举数据。
OUT_STROKE_PRECIS	这个值并不能用于指定字体映射，只是指定 TrueType 和其他的 outline-based 字体，以及矢量字体的枚举数据。
OUT_TT_ONLY_PRECIS	指定字体映射只选择 TrueType 字体。如果系统中没有安装 TrueType 字体，将选择默认操作。
OUT_TT_PRECIS	当系统包含多个名称相同的字体时，指定 TrueType 字体。

lfClipPrecision

指定文字的剪辑精度。剪辑精度定义如何剪辑字符的一部分位于剪辑区域之外的字符。

可以是以下值：

值	含义
CLIP_DEFAULT_PRECIS	指定默认的剪辑行为。
CLIP_STROKE_PRECIS	这个值并不能用于指定字体映射，只是指定光栅（即点阵）、矢量或 TrueType 字体的枚举数据。
CLIP_EMBEDDED	当使用内嵌的只读字体时，必须指定这个标志。
CLIP_LH_ANGLES	如果指定了该值，所有字体的旋转都依赖于坐标系统的方向是逆时针或顺时针。 如果没有指定该值，设备字体始终逆时针旋转，但是其它字体的旋转依赖于坐标系统的方向。 该设置影响 lfOrientation 参数的效果。

lfQuality

指定文字的输出质量。输出质量定义图形设备界面（GDI）必须尝试将逻辑字体属性与实际物理字体的字体属性进行匹配的仔细程度。可以是以下值：

值	含义
ANTIALIASED_QUALITY	指定输出质量是抗锯齿的（如果字体支持）。
DEFAULT_QUALITY	指定输出质量不重要。
DRAFT_QUALITY	草稿质量。字体的显示质量是不重要的。对于光栅字体（即点阵字体），缩放是有效的，这就意味着可以使用更多的尺寸，但是显示质量并不高。如果需要，粗体、斜体、下划线和删除线字体被合成。
NONANTIALIASED_QUALITY	指定输出质量不是抗锯齿的。
PROOF_QUALITY	正稿质量。指定字体质量比匹配字体属性更重要。对于光栅字体（即点阵字体），缩放是无效的，会选用其最接近的字体大小。虽然选中 PROOF_QUALITY 时字体大小不能精确地映射，但是输出质量很高，并且不会有畸变现象。如果需要，粗体、斜体、下划线和删除线字体被合成。

如果 ANTIALIASED_QUALITY 和 NONANTIALIASED_QUALITY 都未被选择，抗锯齿效果将依赖于控制面板中字体抗锯齿的设置。

lfPitchAndFamily

指定以常规方式描述字体的字体系列。字体系列描述大致的字体外观。字体系列用于在所需精确字体不可用时指定字体。

1~2 位指定字体间距，可以是以下值：

值	含义
DEFAULT_PITCH	指定默认间距。
FIXED_PITCH	指定固定间距。
VARIABLE_PITCH	指定可变间距。

4~7 位指定字体系列，可以是以下值：

值	含义
FF_DECORATIVE	指定特殊字体。例如 Old English。
FF_DONTCARE	指定字体系列不重要。
FF_MODERN	指定具有或不具有衬线的等宽字体。例如，Pica、Elite 和 Courier New 都是等宽字体。
FF_ROMAN	指定具有衬线的等比字体。例如 MS Serif。
FF_SCRIPT	指定设计为类似手写体的字体。例如 Script 和 Cursive。
FF_SWISS	指定不具有衬线的等比字体。例如 MS Sans Serif。

字体间距和字体系列可以用布尔运算符 OR 连接(即符号 |)。

lfFaceName

字体名称，名称不得超过 31 个字符。如果是空字符串，系统将使用第一个满足其它属性的字体。

outtext

这个函数用于在当前位置输出字符串。

```
void outtext(  
    LPCSTR textstring,  
    IMAGE* pimg = NULL  
);  
void outtext(  
    CHAR c,  
    IMAGE* pimg = NULL  
);  
void outtext(  
    LPCWSTR textstring,  
    IMAGE* pimg = NULL  
);  
void outtext(  
    WCHAR c,  
    IMAGE* pimg = NULL  
);
```

参数:

textstring

要输出的字符串的指针。

c

要输出的字符。

返回值:

(无)

示例:

```
// 输出字符串  
char s[] = "Hello World";  
outtext(s);  
// 输出字符  
char c = 'A';  
outtext(c);  
// 输出数值, 先将数字格式化输出为字符串  
char s[5];  
sprintf(s, "%d", 1024);  
outtext(s);
```

outtextrect

这个函数用于在指定矩形范围内输出字符串。

```
void outtextrect(  
    int x,  
    int y,  
    int w,  
    int h,  
    LPCSTR textstring  
    IMAGE* pimg = NULL  
);  
void outtextrect(  
    int x,  
    int y,  
    int w,  
    int h,  
    LPCWSTR textstring  
    IMAGE* pimg = NULL  
);
```

参数:

x, y, w, h
 要输出字符串所在的矩形区域

textstring
 要输出的字符串的指针。

返回值:
(无)

outtextxy

这个函数用于在指定位置输出字符串。

```
void outtextxy(  
    int x,  
    int y,  
    LPCSTR textstring  
    IMAGE* pimg = NULL  
);  
void outtextxy(  
    int x,  
    int y,  
    CHAR c  
    IMAGE* pimg = NULL  
);  
void outtextxy(  
    int x,  
    int y,  
    LPCWSTR textstring  
    IMAGE* pimg = NULL  
);  
void outtextxy(  
    int x,  
    int y,  
    WCHAR c  
    IMAGE* pimg = NULL  
);
```

参数:

x

字符串输出时头字母的 x 轴的坐标值

y

字符串输出时头字母的 y 轴的坐标值。

textstring

要输出的字符串的指针。

c

要输出的字符。

返回值:

(无)

示例：

```
// 输出字符串
char s[] = "Hello World";
outtextxy(10, 20, s);
// 输出字符
char c = 'A';
outtextxy(10, 40, c);
// 输出数值，先将数字格式化输出为字符串
char s[5];
sprintf(s, "%d", 1024);
outtextxy(10, 60, s);
```

setfont

这个函数用于设置当前字体样式。

```
void setfont(  
    int nHeight,  
    int nWidth,  
    LPCSTR lpszFace,  
    IMAGE* pimg = NULL  
);  
  
void setfont(  
    int nHeight,  
    int nWidth,  
    LPCSTR lpszFace,  
    int nEscapement,  
    int nOrientation,  
    int nWeight,  
    bool bItalic,  
    bool bUnderline,  
    bool bStrikeOut,  
    IMAGE* pimg = NULL  
);  
  
void setfont(  
    int nHeight,  
    int nWidth,  
    LPCSTR lpszFace,  
    int nEscapement,  
    int nOrientation,  
    int nWeight,  
    bool bItalic,  
    bool bUnderline,  
    bool bStrikeOut,  
    BYTE fbCharSet,  
    BYTE fbOutPrecision,  
    BYTE fbClipPrecision,  
    BYTE fbQuality,  
    BYTE fbPitchAndFamily,  
    IMAGE* pimg = NULL  
);  
  
void setfont(  
    const LOGFONT *font,  
    IMAGE* pimg = NULL  
);
```

参数:

nHeight

指定高度 (逻辑单位)。

如果为正, 表示指定的高度包括字体的默认行距; 如果为负, 表示指定的高度只是字符的高度。

nWidth

字符的平均宽度 (逻辑单位)。如果为 0, 则比例自适应。

lpszFace

字体名称。对于此参数均有 [LPCSTR](#) 和 [LPCWSTR](#) 两个版本, 以上函数声明仅列出一种。提供两个接口是为了方便能同时使用两种不同的字符集。

nEscapement

字符串的书写角度, 单位 0.1 度。

nOrientation

每个字符的书写角度, 单位 0.1 度。

nWeight

字符的笔画粗细, 范围 0~1000。0 表示默认粗细。使用数字或下表中定义的宏均可:

宏	粗细值
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_HEAVY	900
FW_BLACK	900

bItalic

是否斜体, true / false。

bUnderline

是否有下划线, true / false。

bStrikeOut

是否有删除线, true / false。

fbCharSet

指定字符集 (详见 [LOGFONT 结构体](#))。

fbOutPrecision

指定文字的输出精度 (详见 [LOGFONT 结构体](#))。

fbClipPrecision

指定文字的剪辑精度 (详见 [LOGFONT 结构体](#))。

fbQuality

指定文字的输出质 (详见 [LOGFONT 结构体](#))。

fbPitchAndFamily

指定以常规方式描述字体的字体系列 (详见 [LOGFONT 结构体](#))。

font

指向 [LOGFONT](#) 结构体的指针。

返回值:

(无)

示例:

// 设置当前字体为高 16 像素的“宋体”(忽略行距)。

```
setfont(-16, 0, "宋体");
```

```
outtextxy(0, 0, "测试");
```

// 设置输出效果为抗锯齿

```
LOGFONT f;
```

```
getfont(&f); // 获取当前字体设置
```

```
f.lfHeight = 48; // 设置字体高度为 48 (包含行距)
```

```
strcpy(f.lfFaceName, "黑体"); // 设置字体为“黑体”
```

```
f.lfQuality = ANTIALIASED_QUALITY; // 设置输出效果为抗锯齿
```

```
setfont(&f); // 设置字体样式
```

```
outtextxy(0, 50, "抗锯齿效果");
```

settextjustify

这个函数用于设置文字对齐方式。

```
void settextjustify(  
    int horiz,  
    int vert,  
    IMAGE* pimg = NULL  
);
```

参数:

horiz

横向对齐方式, 可选值 LEFT_TEXT (默认), CENTER_TEXT, RIGHT_TEXT

vert

纵向对齐方式, 可选值 TOP_TEXT (默认), CENTER_TEXT, BOTTOM_TEXT

textstring

要输出的字符串的指针。

返回值:

(无)

textheight

这个函数用于获取字符串的高。

```
int textheight(  
    LPCTSTR textstring,  
    IMAGE* pimg = NULL  
);
```

参数：

textstring

指定的字符串指针。

返回值：

该字符串实际占用的像素高度。

示例：

（无）

textwidth

这个函数用于获取字符串的宽。

```
int textwidth(  
    LPCTSTR textstring,  
    IMAGE* pimg = NULL  
);
```

参数：

textstring

指定的字符串指针。

返回值：

该字符串实际占用的像素宽度。

示例：

（无）

图像处理相关函数

相关函数和数据如下：

函数或数据	描述
getimage	从屏幕 / 文件 / 资源 / IMAGE 对象中获取图像。
IMAGE 对象	保存图像的对象。
imagefilter_blurring	对指定图像进行图像模糊滤镜操作。
putimage	在屏幕上绘制指定图像。
putimage_alphablend	在屏幕上以半透明方式绘制指定图像。
putimage_transparent	在屏幕上以透明方式绘制指定图像。
putimage_alphatransparent	在屏幕上以透明/半透明方式绘制指定图像。
三元光栅操作码	

getimage

这个函数的四个重载分别用于从屏幕 / 文件 / 资源 / IMAGE 对象中获取图像。

// 从屏幕获取图像

```
void getimage(
    IMAGE *pDstImg,    // 保存图像的 IMAGE 对象指针
    int srcX,          // 要获取图像的区域左上角 x 坐标
    int srcY,          // 要获取图像的区域左上角 y 坐标
    int srcWidth,      // 要获取图像的区域宽度
    int srcHeight      // 要获取图像的区域高度
);
// 从图片文件获取图像(bmp/jpg/gif/emf/wmf/ico)
void getimage(
    IMAGE *pDstImg,    // 保存图像的 IMAGE 对象指针
    LPCTSTR pImgFile,  // 图片文件名
    int zoomWidth = 0,  // 设定图像缩放至的宽度 (0 表示默认宽度, 不缩放)
    int zoomHeight = 0  // 设定图像缩放至的高度 (0 表示默认高度, 不缩放)
);
// 从资源文件获取图像(bmp/jpg/gif/emf/wmf/ico)
void getimage(
    IMAGE *pDstImg,    // 保存图像的 IMAGE 对象指针
    LPCTSTR pResType,  // 资源类型
    LPCTSTR pResName,  // 资源名称
    int zoomWidth = 0,  // 设定图像缩放至的宽度 (0 表示默认宽度, 不缩放)
    int zoomHeight = 0  // 设定图像缩放至的高度 (0 表示默认高度, 不缩放)
);
// 从另一个 IMAGE 对象中获取图像
void getimage(
    IMAGE *pDstImg,    // 保存图像的 IMAGE 对象指针
    const IMAGE *pSrcImg, // 源图像 IMAGE 对象
    int srcX,          // 要获取图像的区域左上角 x 坐标
    int srcY,          // 要获取图像的区域左上角 y 坐标
    int srcWidth,      // 要获取图像的区域宽度
    int srcHeight      // 要获取图像的区域高度
);
```

参数:

(详见各重载函数原型内的注释)

返回值:

(无)

示例:

请参考 [putimage](#) 函数示例。

IMAGE 对象

保存图像的对象。

```
class IMAGE;
```

成员：

（隐藏）

由于该绘图库面向初学者，所以尽力隐藏了面向对象的内容。

示例：

以下语句可以创建一个名为 img 的 IMAGE 对象：

```
IMAGE img;
```

更多示例请参考 [putimage](#) 函数示例。

imagefilter blurring

这个函数用于对一图片区域进行模糊滤镜操作。

```
int imagefilter_blurring (  
    PIMAGE imgdest,  
    int intensity,  
    int alpha,  
    int nXOriginDest = 0,  
    int nYOriginDest = 0,  
    int nWidthDest = 0,  
    int nHeightDest = 0  
);
```

参数：

imgdest

要进行模糊操作的图片，如果为 NULL 则表示操作窗口上的图片

intensity

模糊度，值越大越模糊。当值在 0x0 - 0x7F 之间时，为四向模糊；当值在 0x80 - 0xFF 之间时，为八向模糊，运算量会大一倍

alpha

图像亮度。取值为0x100表示亮度不变，取值为0x0表示图像变成纯黑

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的矩形区域。如果 nWidthDest 和 nHeightDest 为0，表示操作整张图片。

返回值：

成功返回0，否则返回非0，若 imgdest 传入错误，会引发运行时异常。

示例：

（无）。

putimage

这个函数的几个重载用于在屏幕或另一个图像上绘制指定图像。

// 绘制图像到屏幕

```
void putimage(  
    int dstX,           // 绘制位置的 x 坐标  
    int dstY,           // 绘制位置的 y 坐标  
    IMAGE *pSrcImg,     // 要绘制的 IMAGE 对象指针  
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）  
);
```

// 绘制图像到屏幕(指定宽高)

```
void putimage(  
    int dstX,           // 绘制位置的 x 坐标  
    int dstY,           // 绘制位置的 y 坐标  
    int dstWidth,       // 绘制的宽度  
    int dstHeight,      // 绘制的高度  
    IMAGE *pSrcImg,     // 要绘制的 IMAGE 对象指针  
    int srcX,           // 绘制内容在 IMAGE 对象中的左上角 x 坐标  
    int srcY,           // 绘制内容在 IMAGE 对象中的左上角 y 坐标  
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）  
);
```

// 绘制图像到屏幕(拉伸)

```
void putimage(  
    int dstX,           // 绘制位置的 x 坐标  
    int dstY,           // 绘制位置的 y 坐标  
    int dstWidth,       // 绘制的宽度  
    int dstHeight,      // 绘制的高度  
    IMAGE *pSrcImg,     // 要绘制的 IMAGE 对象指针  
    int srcX,           // 绘制内容在 IMAGE 对象中的左上角 x 坐标  
    int srcY,           // 绘制内容在 IMAGE 对象中的左上角 y 坐标  
    int srcWidth,       // 绘制内容在源 IMAGE 对象中的宽度  
    int srcHeight,      // 绘制内容在源 IMAGE 对象中的高度  
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）  
);
```

// 绘制图像到另一图像

```
void putimage(  
    IMAGE *pDstImg,     // 目标 IMAGE 对象指针  
    int dstX,           // 绘制位置的 x 坐标  
    int dstY,           // 绘制位置的 y 坐标  
    IMAGE *pSrcImg,     // 源 IMAGE 对象指针  
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）  
);
```

// 绘制图像到另一图像(指定宽高)

```

void putimage(
    IMAGE *pDstImg,        // 目标 IMAGE 对象指针
    int dstX,              // 绘制位置的 x 坐标
    int dstY,              // 绘制位置的 y 坐标
    int dstWidth,          // 绘制的宽度
    int dstHeight,         // 绘制的高度
    IMAGE *pSrcImg,        // 源 IMAGE 对象指针
    int srcX,              // 绘制内容在源 IMAGE 对象中的左上角 x 坐标
    int srcY,              // 绘制内容在源 IMAGE 对象中的左上角 y 坐标
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）
);
// 绘制图像到另一图像(拉伸)
void putimage(
    IMAGE *pDstImg,        // 目标 IMAGE 对象指针
    int dstX,              // 绘制位置的 x 坐标
    int dstY,              // 绘制位置的 y 坐标
    int dstWidth,          // 绘制的宽度
    int dstHeight,         // 绘制的高度
    IMAGE *pSrcImg,        // 源 IMAGE 对象指针
    int srcX,              // 绘制内容在源 IMAGE 对象中的左上角 x 坐标
    int srcY,              // 绘制内容在源 IMAGE 对象中的左上角 y 坐标
    int srcWidth,          // 绘制内容在源 IMAGE 对象中的宽度
    int srcHeight,         // 绘制内容在源 IMAGE 对象中的高度
    DWORD dwRop = SRCCOPY // 三元光栅操作码（详见备注）
);

```

参数：

（详见各重载函数原型内的注释）

备注：

三元光栅操作码（即位操作模式），支持全部的 256 种三元光栅操作码，常用的几种如下：

值	含义
DSTINVERT	绘制出的像素颜色 = NOT 屏幕颜色
MERGECOPY	绘制出的像素颜色 = 图像颜色 AND 当前填充颜色
MERGEPAINT	绘制出的像素颜色 = 屏幕颜色 OR (NOT 图像颜色)
NOTSRCCOPY	绘制出的像素颜色 = NOT 图像颜色
NOTSRCERASE	绘制出的像素颜色 = NOT (屏幕颜色 OR 图像颜色)
PATCOPY	绘制出的像素颜色 = 当前填充颜色
PATINVERT	绘制出的像素颜色 = 屏幕颜色 XOR 当前填充颜色
PATPAINT	绘制出的像素颜色 = 屏幕颜色 OR ((NOT 图像颜色) OR 当前填充颜色)
SRCAND	绘制出的像素颜色 = 屏幕颜色 AND 图像颜色
SRCCOPY	绘制出的像素颜色 = 图像颜色
SRCERASE	绘制出的像素颜色 = (NOT 屏幕颜色) AND 图像颜色
SRCINVERT	绘制出的像素颜色 = 屏幕颜色 XOR 图像颜色
SRCPAINT	绘制出的像素颜色 = 屏幕颜色 OR 图像颜色

注：

1. AND / OR / NOT / XOR 为布尔位运算。
2. “屏幕颜色”指绘制所经过的屏幕像素点的颜色。
3. “图像颜色”是指通过 IMAGE 对象中的图像的颜色。
4. “当前填充颜色”是指通过 setfillstyle 设置的用于当前填充的颜色。
5. 查看全部的三元光栅操作码请详见：[三元光栅操作码](#)。

返回值：

（无）

示例：

以下局部代码读取 c:\test.jpg 绘制在屏幕左上角：

```
IMAGE img;  
getimage(&img, "c:\\test.jpg");  
putimage(0, 0, &img);
```

以下局部代码将屏幕 (0,0) 起始的长宽各 100 像素的图像拷贝至 (200,200) 位置：

```
IMAGE img;  
getimage(&img, 0, 0, 100, 100);  
putimage(200, 200, &img);
```

putimage_alphablend

这个函数用于对两张图片进行半透明混合，并把混合结果写入目标图片。

```
int putimage_alphablend(  
    PIMAGE imgdest,        // handle to dest  
    PIMAGE imgsrc,         // handle to source  
    int nXOriginDest,       // x-coord of destination upper-left corner  
    int nYOriginDest,       // y-coord of destination upper-left corner  
    unsigned char alpha,    // alpha  
    int nXOriginSrc = 0,    // x-coord of source upper-left corner  
    int nYOriginSrc = 0,    // y-coord of source upper-left corner  
    int nWidthSrc = 0,      // width of source rectangle  
    int nHeightSrc = 0      // height of source rectangle  
);
```

参数：

imgdest

要进行半透明混合的目标图片，如果为 NULL 则表示操作窗口上的图片

imgsrc

要进行半透明混合的源图片，该操作不会改变源图片

nXOriginDest, nYOriginDest

要开始进行混合的目标图片坐标，该坐标是混合区域的左上角

alpha

透明度值，如果为0x0，表示源图片完全透明，如果为0xFF，表示源图片完全不透明。

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的源图矩形区域。如果 nWidthDest 和 nHeightDest 为0，表示操作整张图片。

返回值：

成功返回0，否则返回非0，若 imgdest 或 imgsrc 传入错误，会引发运行时异常。

示例：

（无）。

putimage transparent

这个函数用于对两张图片进行透明混合，并把混合结果写入目标图片。

```
int putimage_transparent(  
    PIMAGE imgdest,        // handle to dest  
    PIMAGE imgsrc,         // handle to source  
    int nXOriginDest,       // x-coord of destination upper-left corner  
    int nYOriginDest,       // y-coord of destination upper-left corner  
    COLORREF crTransparent, // color to make transparent  
    int nXOriginSrc = 0,    // x-coord of source upper-left corner  
    int nYOriginSrc = 0,    // y-coord of source upper-left corner  
    int nWidthSrc = 0,      // width of source rectangle  
    int nHeightSrc = 0      // height of source rectangle  
);
```

参数:

imgdest

要进行透明混合的目标图片，如果为 NULL 则表示操作窗口上的图片

imgsrc

要进行透明混合的源图片，该操作不会改变源图片

nXOriginDest, nYOriginDest

要开始进行混合的目标图片坐标，该坐标是混合区域的左上角

crTransparent

关键色。源图片上为该颜色值的像素，将忽略，不会改写目标图片上相应位置的像素。

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的源图矩形区域。如果 nWidthDest 和 nHeightDest 为0，表示操作整张图片。

返回值:

成功返回0，否则返回非0，若 imgdest 或 imgsrc 传入错误，会引发运行时异常。

示例:

(无)。

putimage_alphatransparent

这个函数用于对两张图片进行透明/半透明混合，并把混合结果写入目标图片。

```
int putimage_alphatransparent(  
    PIMAGE imgdest,          // handle to dest  
    PIMAGE imgsrc,           // handle to source  
    int nXOriginDest,        // x-coord of destination upper-left corner  
    int nYOriginDest,        // y-coord of destination upper-left corner  
    COLORREF crTransparent,  // color to make transparent  
    unsigned char alpha,     // alpha  
    int nXOriginSrc = 0,     // x-coord of source upper-left corner  
    int nYOriginSrc = 0,     // y-coord of source upper-left corner  
    int nWidthSrc = 0,       // width of source rectangle  
    int nHeightSrc = 0       // height of source rectangle  
);
```

参数：

imgdest

要进行半透明混合的目标图片，如果为 NULL 则表示操作窗口上的图片

imgsrc

要进行半透明混合的源图片，该操作不会改变源图片

nXOriginDest, nYOriginDest

要开始进行混合的目标图片坐标，该坐标是混合区域的左上角

crTransparent

关键色。源图片上为该颜色值的像素，将忽略，不会改写目标图片上相应位置的像素。

alpha

透明度值，如果为0x0，表示源图片完全透明，如果为0xFF，表示源图片完全不透明。

nXOriginDest, nYOriginDest, nWidthDest, nHeightDest

描述要进行此操作的源图矩形区域。如果 nWidthDest 和 nHeightDest 为0，表示操作整张图片。

返回值：

成功返回0，否则返回非0，若 imgdest 或 imgsrc 传入错误，会引发运行时异常。

示例：

（无）。

三元光栅操作码

这篇补充文档列出了 putimage 函数支持的所有三元光栅操作码。

三元光栅操作码定义了源图像与屏幕图像的位合并形式,这个合并形式是以下三个操作数对应像素的布尔运算:

操作数	含义
D	屏幕图像
P	当前填充颜色
S	源图像

布尔运算符包括以下几种:

操作	含义
a	位的 AND 运算(双目运算)
n	位的 NOT 运算(单目运算)
o	位的 OR 运算(双目运算)
x	位的 XOR 运算(双目运算)

所有的布尔操作都采用逆波兰表示法,例如,“当前填充颜色 or 源图像”可表示为: PSo。
(当然 SPo 也是等价的,这里只列举出了其中一种等价格式)

三元光栅操作码是 32位 int 类型,其高位字是布尔操作索引,低位字是操作码。布尔操作索引的 16 个位中,高 8 位用 0 填充,低 8 位是当前填充颜色、源图像和屏幕的布尔操作结果。例如, PSo 和 DPSoo 的操作索引如下:

P	S	D	PSo	DPSoo
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1
操作索引:			00FCh	00FEh

上例中, PSo 的操作索引是 00FC (从下往上读), DPSoo 的是 00FE。这些值定义了相应的三元光栅操作码在“三元光栅操作码”表格中的位置, PSo 在 252 (00FCh) 行, DPSoo 在 254 (00FEh) 行。常用的三元光栅操作码已经定义了常量名, 程序中可以直接使用。

三元光栅操作码

布尔功能 (16 进制)	光栅操作 (16 进制)	布尔功能的逆波兰表示法	常量名
00	00000042	0	BLACKNESS
01	00010289	DPSoon	
02	00020C89	DPSona	
03	000300AA	PSon	
04	00040C88	SDPona	

05	000500A9	DPon	
06	00060865	PDSxnon	
07	000702C5	PDSaon	
08	00080F08	SDPnaa	
09	00090245	PDSxon	
0A	000A0329	DPna	
0B	000B0B2A	PSDnaon	
0C	000C0324	SPna	
0D	000D0B25	PDSnaon	
0E	000E08A5	PDSonon	
0F	000F0001	Pn	
10	00100C85	PDSona	
11	001100A6	DSon	NOTSRCERASE
12	00120868	SDPxnon	
13	001302C8	SDPaon	
14	00140869	DPSxnon	
15	001502C9	DPSaon	
16	00165CCA	PSDPSanaxx	
17	00171D54	SSPxDSxaxn	
18	00180D59	SPxPDxa	
19	00191CC8	SDPSanaxn	
1A	001A06C5	PDSPaox	
1B	001B0768	SDPSxaxn	
1C	001C06CA	PSDPaox	
1D	001D0766	DSPDxaxn	
1E	001E01A5	PDSox	
1F	001F0385	PDSoan	
20	00200F09	DPSnaa	
21	00210248	SDPxon	
22	00220326	DSna	
23	00230B24	SPDnaon	
24	00240D55	SPxDSxa	
25	00251CC5	PDSPanaxn	
26	002606C8	SDPSaox	
27	00271868	SDPSxnox	
28	00280369	DPSxa	
29	002916CA	PSDPSaoxxn	
2A	002A0CC9	DPSana	
2B	002B1D58	SSPxPDxaxn	
2C	002C0784	SPDSaox	
2D	002D060A	PSDnox	
2E	002E064A	PSDPxox	
2F	002F0E2A	PSDnoan	

30	0030032A	PSna	
31	00310B28	SDPnaon	
32	00320688	SDPSoox	
33	00330008	Sn	NOTSRCCOPY
34	003406C4	SPDSaox	
35	00351864	SPDSxnox	
36	003601A8	SDPox	
37	00370388	SDPoan	
38	0038078A	PSDPoax	
39	00390604	SPDnox	
3A	003A0644	SPDSxox	
3B	003B0E24	SPDnoan	
3C	003C004A	PSx	
3D	003D18A4	SPDSonox	
3E	003E1B24	SPDSnaox	
3F	003F00EA	PSan	
40	00400F0A	PSDnaa	
41	00410249	DPSxon	
42	00420D5D	SDxPDxa	
43	00431CC4	SPDSanaxn	
44	00440328	SDna	SRCERASE
45	00450B29	DPSnaon	
46	004606C6	DSPDaox	
47	0047076A	PSDPxaxn	
48	00480368	SDPxa	
49	004916C5	PDSPDaoxxn	
4A	004A0789	DPSDoax	
4B	004B0605	PDSnox	
4C	004C0CC8	SDPana	
4D	004D1954	SSPxDSxoxn	
4E	004E0645	PDSPxox	
4F	004F0E25	PDSnoan	
50	00500325	PDna	
51	00510B26	DSPnaon	
52	005206C9	DPSDaox	
53	00530764	SPDSxaxn	
54	005408A9	DPSonon	
55	00550009	Dn	DSTINVERT
56	005601A9	DPSox	
57	00570389	DPSoan	
58	00580785	PDSPoax	
59	00590609	DPSnox	
5A	005A0049	DPx	PATINVERT

5B	005B18A9	DPSDonox	
5C	005C0649	DPSDxox	
5D	005D0E29	DPSnoan	
5E	005E1B29	DPSDnaox	
5F	005F00E9	DPan	
60	00600365	PDSxa	
61	006116C6	DSPDSaoxxn	
62	00620786	DSPDoax	
63	00630608	SDPnox	
64	00640788	SDPSoax	
65	00650606	DSPnox	
66	00660046	DSx	SRCINVERT
67	006718A8	SDPSonox	
68	006858A6	DSPDSonoxxn	
69	00690145	PDSxxn	
6A	006A01E9	DPSax	
6B	006B178A	PSDPSoaxxn	
6C	006C01E8	SDPax	
6D	006D1785	PDSPDoaxxn	
6E	006E1E28	SDPSnoax	
6F	006F0C65	PDSxnan	
70	00700CC5	PDSana	
71	00711D5C	SSDxPDxaxn	
72	00720648	SDPSxox	
73	00730E28	SDPnoan	
74	00740646	DSPDxox	
75	00750E26	DSPnoan	
76	00761B28	SDPSnaox	
77	007700E6	DSan	
78	007801E5	PDSax	
79	00791786	DSPDSoaxxn	
7A	007A1E29	DPSDnoax	
7B	007B0C68	SDPxnan	
7C	007C1E24	SPDSnoax	
7D	007D0C69	DPSxnan	
7E	007E0955	SPxDSxo	
7F	007F03C9	DPSaan	
80	008003E9	DPSaa	
81	00810975	SPxDSxon	
82	00820C49	DPSxna	
83	00831E04	SPDSnoaxn	
84	00840C48	SDPxna	
85	00851E05	PDSPnoaxn	

86	008617A6	DSPDSoaxx	
87	008701C5	PDSaxn	
88	008800C6	DSa	SRCAND
89	00891B08	SDPSnaoxn	
8A	008A0E06	DSPnoa	
8B	008B0666	DSPDxoxn	
8C	008C0E08	SDPnoa	
8D	008D0668	SDPSxoxn	
8E	008E1D7C	SSDxPDxax	
8F	008F0CE5	PDSanan	
90	00900C45	PDSxna	
91	00911E08	SDPSnoaxn	
92	009217A9	DPSDPoaxx	
93	009301C4	SPDaxn	
94	009417AA	PSDPSoaxx	
95	009501C9	DPSaxn	
96	00960169	DPSxx	
97	0097588A	PSDPSonoxx	
98	00981888	SDPSonoxn	
99	00990066	DSxn	
9A	009A0709	DPSnax	
9B	009B07A8	SDPSoaxn	
9C	009C0704	SPDnax	
9D	009D07A6	DSPDoaxn	
9E	009E16E6	DSPDSoaxx	
9F	009F0345	PDSxan	
A0	00A000C9	DPa	
A1	00A11B05	PDSPnaoxn	
A2	00A20E09	DPSnoa	
A3	00A30669	DPSDxoxn	
A4	00A41885	PDSPonoxn	
A5	00A50065	PDxn	
A6	00A60706	DSPnax	
A7	00A707A5	PDSPoaxn	
A8	00A803A9	DPSoa	
A9	00A90189	DPSoxn	
AA	00AA0029	D	
AB	00AB0889	DPSono	
AC	00AC0744	SPDSxax	
AD	00AD06E9	DPSDaoxn	
AE	00AE0B06	DSPnao	
AF	00AF0229	DPno	
B0	00B00E05	PDSnoa	

B1	00B10665	PDSPxoxn	
B2	00B21974	SSPxDSxox	
B3	00B30CE8	SDPanan	
B4	00B4070A	PSDnax	
B5	00B507A9	DPSDoaxn	
B6	00B616E9	DPSDPaoxx	
B7	00B70348	SDPxan	
B8	00B8074A	PSDPxax	
B9	00B906E6	DSPDaoxn	
BA	00BA0B09	DPSnao	
BB	00BB0226	DSno	MERGEPAINT
BC	00BC1CE4	SPDSanax	
BD	00BD0D7D	SDxPDxan	
BE	00BE0269	DPSxo	
BF	00BF08C9	DPSano	
C0	00C000CA	PSa	MERGECOPY
C1	00C11B04	SPDSnaoxn	
C2	00C21884	SPDSonoxn	
C3	00C3006A	PSxn	
C4	00C40E04	SPDnoa	
C5	00C50664	SPDSxoxn	
C6	00C60708	SDPnax	
C7	00C707AA	PSDPoaxn	
C8	00C803A8	SDPoa	
C9	00C90184	SPDoxn	
CA	00CA0749	DPSDxax	
CB	00CB06E4	SPDSaoxn	
CC	00CC0020	S	SRCCOPY
CD	00CD0888	SDPono	
CE	00CE0B08	SDPnao	
CF	00CF0224	SPno	
D0	00D00E0A	PSDnoa	
D1	00D1066A	PSDPxoxn	
D2	00D20705	PDSnax	
D3	00D307A4	SPDSaoxn	
D4	00D41D78	SSPxPDxax	
D5	00D50CE9	DPSanan	
D6	00D616EA	PSDPSaoxx	
D7	00D70349	DPSxan	
D8	00D80745	PDSPxax	
D9	00D906E8	SDPSaoxn	
DA	00DA1CE9	DPSDanax	
DB	00DB0D75	SPxDSxan	

DC	00DC0B04	SPDnao	
DD	00DD0228	SDno	
DE	00DE0268	SDPxo	
DF	00DF08C8	SDPano	
E0	00E003A5	PDSoa	
E1	00E10185	PDSoxn	
E2	00E20746	DSPDxax	
E3	00E306EA	PSDPaoxn	
E4	00E40748	SDPSxax	
E5	00E506E5	PDSPaoxn	
E6	00E61CE8	SDPSanax	
E7	00E70D79	SPxPDxan	
E8	00E81D74	SSPxDSxax	
E9	00E95CE6	DSPDSanaxxn	
EA	00EA02E9	DPSao	
EB	00EB0849	DPSxno	
EC	00EC02E8	SDPao	
ED	00ED0848	SDPxno	
EE	00EE0086	DSO	SRCPAINT
EF	00EF0A08	SDPnoo	
F0	00F00021	P	PATCOPY
F1	00F10885	PDSono	
F2	00F20B05	PDSnao	
F3	00F3022A	PSno	
F4	00F40B0A	PSDnao	
F5	00F50225	PDno	
F6	00F60265	PDSxo	
F7	00F708C5	PDSano	
F8	00F802E5	PDSao	
F9	00F90845	PDSxno	
FA	00FA0089	DPo	
FB	00FB0A09	DPSnoo	PATPAINT
FC	00FC008A	PSO	
FD	00FD0A0A	PSDnoo	
FE	00FE02A9	DPSoo	
FF	00FF0062	1	WHITENESS

鼠标相关函数

鼠标消息缓冲区可以缓冲 1024 个未处理的鼠标消息。每一次 GetMessage 将从鼠标消息缓冲区取出一个最早发生的消息。当鼠标消息缓冲区满了以后，将自动清除旧消息。

相关函数和数据如下：

函数或数据	描述
FlushMouseMsgBuffer	清空鼠标消息缓冲区。
GetMessage	获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待。
GetMousePos	获取当前鼠标位置。无等待。
MouseHit	检测当前是否有鼠标消息。
ShowMouse	设置鼠标显示状态
MOUSEMSG 结构体	保存鼠标消息的结构体。

FlushMouseMsgBuffer

这个函数用于清空鼠标消息缓冲区。

```
void FlushMouseMsgBuffer();
```

参数:

(无)

返回值:

(无)

示例:

(无)

GetMouseMsg

这个函数用于获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待。

```
MOUSEMSG GetMouseMsg();
```

参数：

（无）

返回值：

返回保存有鼠标消息的结构体。

示例：

请参见示例程序中的“鼠标操作范例”。

GetMousePos

这个函数用于获取一个鼠标消息。如果当前鼠标消息队列中没有，就一直等待。

```
int GetMousePos(int *x, int *y);
```

参数：

*x, *y

指向两个 int 的指针，用于接收鼠标坐标

返回值：

总是返回0。

示例：

请参见论坛教程里的用户交换的鼠标部分。

MouseHit

这个函数用于检测当前是否有鼠标消息。

```
int MouseHit();
```

参数:

(无)

返回值:

如果存在鼠标消息，返回 1；否则返回 false。

示例:

(无)

ShowMouse

这个函数用于检测设置鼠标隐藏。

```
int ShowMouse(int bShow);
```

参数：

bShow

为0则不显示，非0为显示。默认显示。

返回值：

返回上一次调用时设置的值，第一次调用的话返回1。

示例：

（无）

MOUSEMSG 结构体

这个结构体用于保存鼠标消息，定义如下：

```
struct MOUSEMSG
{
    UINT uMsg;        // 当前鼠标消息
    int  mkCtrl;       // Ctrl 键是否按下
    int  mkShift;      // Shift 键是否按下
    int  mkLButton;    // 鼠标左键是否按下
    int  mkMButton;    // 鼠标中键是否按下
    int  mkRButton;    // 鼠标右键是否按下
    int  x;            // 当前鼠标 x 坐标
    int  y;            // 当前鼠标 y 坐标
    int  wheel;        // 鼠标滚轮滚动值
};
```

成员：

uMsg：

指定鼠标消息类型，可为以下值：

值	含义
WM_MOUSEMOVE	鼠标移动消息。
WM_MOUSEWHEEL	鼠标滚轮拨动消息。
WM_LBUTTONDOWN	左键按下消息。
WM_LBUTTONUP	左键弹起消息。
WM_LBUTTONDBLCLK	左键双击消息。
WM_MBUTTONDOWN	中键按下消息。
WM_MBUTTONUP	中键弹起消息。
WM_MBUTTONDBLCLK	中键双击消息。
WM_RBUTTONDOWN	右键按下消息。
WM_RBUTTONUP	右键弹起消息。
WM_RBUTTONDBLCLK	右键双击消息。

mkCtrl

Ctrl 键是否按下

mkShift

Shift 键是否按下

mkLButton

鼠标左键是否按下

mkMButton

鼠标中键是否按下

mkRButton

鼠标右键是否按下

x

当前鼠标 x 坐标

y

当前鼠标 y 坐标

wheel

鼠标滚轮滚动值，一般情况下为 120 的倍数或者约数。

示例：

（无）

时间函数

相关函数和数据如下：

函数或数据	描述
API Sleep	实际调用 Sleep，因直接调用 Sleep 会被转化为调用 delay
delay	同 delay_ms。
delay_ms	延迟以毫秒为单位的时间。
delay_fps	延迟以 FPS 为准的时间，以实现稳定帧率。
delay_jfps	延迟以 FPS 为准的时间，以实现稳定帧率（带跳帧）。
fclock	获取当前程序从初始化起经过的时间，以秒为单位。

API Sleep

与 Sleep 函数完全相同，单纯延迟指定时间（精确程度由系统 API 决定），其它事情什么都不干。

```
VOID API_Sleep(DWORD dwMilliseconds);
```

参数：

dwMilliseconds

要延迟的时间，以毫秒为单位，如果为0则不产生延时的作用（相当于无意义调用）。不会附带刷新窗口的作用。

返回值：

（无）

示例：

（无）

delay

与 delay_ms 完全相同，详见 delay_ms。

```
void delay(int Milliseconds);
```

参数：

Milliseconds

要延迟的时间，以毫秒为单位

返回值：

（无）

示例：

（无）

delay_ms

延迟以毫秒为单位的时间。

```
void delay_ms(int Milliseconds);
```

参数:

Milliseconds

要延迟的时间，以毫秒为单位, 并更新 FPS 计数值。如果为0，则智能刷新窗口。

返回值:

(无)

示例:

(无)

delay_fps

延迟以 FPS 为准的时间，以实现稳定帧率。

```
void delay_fps(int fps);
```

参数：

fps

要得到的帧率，平均延迟 $1000/\text{fps}$ 毫秒, 并更新 FPS 计数值。这个函数一秒最多能调用 fps 次。

返回值：

（无）

示例：

（无）

delay_jfps

延迟以 FPS 为准的时间，以实现稳定帧率（带跳帧）。

```
void delay_jfps(int fps);
```

参数：

fps

要得到的帧率，平均延迟 $1000/\text{fps}$ 毫秒, 并更新 FPS 计数值。这个函数一秒最多能调用 fps 次。注意的是，即使这帧跳过了，仍然会更新 FPS 计数值。

返回值：

（无）

示例：

（无）

fclock

获取当前程序从初始化起经过的时间，以秒为单位。

```
double fclock();
```

参数：

（无）

返回值：

返回一个以秒为单位的浮点数，精度比 API 的 GetTickCount 稍高。程序中使用一般用于求时间差，一般不要直接使用这个值。

示例：

（无）

数学函数

相关函数和数据如下：

函数或数据	描述
<u>rotate_point3d_x</u>	把一个3d 点绕 x 轴旋转
<u>rotate_point3d_y</u>	把一个3d 点绕 y 轴旋转
<u>rotate_point3d_z</u>	把一个3d 点绕 z 轴旋转（实现2d 旋转时使用这个函数）
<u>VECTOR3D</u>	库提供的3d 向量类，以下为类的成员简介
<u>VECTOR3D::operator =</u>	向量复制
<u>VECTOR3D::operator +</u>	3d 向量加法
<u>VECTOR3D::operator -</u>	3d 向量减法
<u>VECTOR3D::operator *</u>	与浮点数相乘时为向量缩放，与向量相乘时为点乘
<u>VECTOR3D::operator &</u>	向量叉乘
<u>VECTOR3D::GetAngel</u>	计算两个3d 向量的夹角
<u>VECTOR3D::GetModule</u>	计算3d 向量的模
<u>VECTOR3D::GetSqrModule</u>	计算3d 向量模的平方
<u>VECTOR3D::Rotate</u>	3d 向量绕另一任意向量旋转，或者按指定旋转角旋转
<u>VECTOR3D::SetModule</u>	在保持方向不变的情况下把3d 向量长度改为设定值

暂时本分类只做函数名介绍，详细参数见头文件

其它函数

相关函数和数据如下：

函数或数据	描述
BeginBatchDraw	开始批量绘图。
EndBatchDraw	结束批量绘制，并执行未完成的绘制任务。
FlushBatchDraw	执行未完成的绘制任务。
GetFPS	获取当前窗口刷新率（FPS = Frame Per Second）
GetHWnd	获取当前窗口句柄。
InputBoxGetLine	使用对话框让用户输入一个字符串
keystate	判断某按键是否按下
random	生成某范围内的随机数
randomf	生成0.0-1.0范围内的随机数（0.0取到，1.0取不到）
randomize	初始化随机数序列

BeginBatchDraw

这个函数用于开始批量绘图。执行后，任何绘图操作都将暂时不输出到屏幕上，直到执行 FlushBatchDraw 或 EndBatchDraw 才将之前的绘图输出。

```
void BeginBatchDraw();
```

参数：

（无）

返回值：

（无）

示例：

以下代码实现一个圆从左向右移动，会有比较明显的闪烁。

请取消 main 函数中的三个注释，以实现批绘图功能，可以消除闪烁，并且提升绘图效率。

```
#include "graphics.h"

int main()
{
    initgraph(640, 480);

    setcolor(WHITE);
    setfillstyle(RED);

    // BeginBatchDraw();
    for(int i=50; i<600; i++)
    {
        cleardevice();
        circle(i, 100, 40);
        floodfill(i, 100, WHITE);
        Sleep(10);
        // FlushBatchDraw();
    }
    // EndBatchDraw();

    closegraph();
    return 0;
}
```

EndBatchDraw

这个函数用于结束批量绘制，并执行未完成的绘制任务。

// 结束批量绘制，并执行未完成的绘制任务

```
void EndBatchDraw();
```

// 结束批量绘制，并执行指定区域内未完成的绘制任务

```
void EndBatchDraw(
```

```
    int left,
```

```
    int top,
```

```
    int right,
```

```
    int bottom
```

```
);
```

参数：

left

指定区域的左部 x 坐标。

top

指定区域的上部 y 坐标。

right

指定区域的右部 x 坐标。

bottom

指定区域的下部 y 坐标。

返回值：

（无）

示例：

请参见 [BeginBatchDraw](#) 的示例。

注意：

在 EGE 下，以上坐标无作用。

FlushBatchDraw

这个函数用于执行未完成的绘制任务。

// 执行未完成的绘制任务

```
void FlushBatchDraw();
```

// 执行指定区域内未完成的绘制任务

```
void FlushBatchDraw(
```

```
    int left,
```

```
    int top,
```

```
    int right,
```

```
    int bottom
```

```
);
```

参数:

left

指定区域的左部 x 坐标。

top

指定区域的上部 y 坐标。

right

指定区域的右部 x 坐标。

bottom

指定区域的下部 y 坐标。

返回值:

(无)

示例:

请参见 [BeginBatchDraw](#) 的示例。

注意:

在 EGE 下，以上坐标无作用。

GetFPS

这个函数用于获取当前刷新率。

```
float GetFPS(  
    int flag = 1  
);
```

参数:

flag

仅能为0或者1, 如果为1, 查询的是逻辑帧数; 如果为0, 查询的是渲染帧数。两者之差可以得到无效帧数(被跳过渲染的帧数, 仅在调用 `delay_jfps` 会产生)。如果没有调用过 `delay_jfps`, 那么两者无区别。

返回值:

返回当前刷新率。

说明:

FPS (Frames Per Second): 每秒传输帧数。通常, 这个帧数在动画或者游戏里, 至少要达到30才能基本流畅。现代液晶显示器均使用60FPS 的刷新率, 所以, 如果你希望在你的显示器上达到最佳效果, 那你需要至少60FPS。

而使内部 FPS 计数增加的方式是当你绘图后, 调用 `delay` 族函数, 如: `FlushBatchDraw`, `delay`, `delay_ms`, `delay_fps`, `Sleep`, 否则你不调用这些函数时, FPS 永远为0而不会变化。

示例:

参见示例程序中的“星空”

GetHwnd

这个函数用于获取绘图窗口句柄。

```
HWND GetHwnd();
```

参数:

(无)

返回值:

返回绘图窗口句柄。

说明:

在 Windows 下，句柄是一个窗口的标识，得到句柄后，可以使用 Windows SDK 中的各种命令实现对窗口的控制。

示例:

```
// 获得窗口句柄
HWND hWnd = GetHwnd();
// 使用 API 函数修改窗口名称
SetWindowText(hWnd, TEXT("Hello!"));
```

InputBoxGetLine

使用对话框让用户输入一个字符串

```
int InputBoxGetLine(LPCSTR title, LPCSTR text, LPSTR buf, int len);
int InputBoxGetLine(LPCWSTR title, LPCWSTR text, LPWSTR buf, int len);
```

参数:

title

对话框标题

text

对话框内显示的提示文字，可以使用'\n' 或者'\t' 进行格式控制。

buf

用于接收输入的字符串指针，指向一个缓冲区

len

指定 buf 指向的缓冲区的大小，同时也会限制在对话框里输入的最大长度

返回值:

返回 1 表示输入有效，buf 中的内容为用户所输入的数据，返回 0 表示输入无效，同时 buf 清空。

示例:

```
#include "graphics.h"

int main()
{
    initgraph(640, 480);
    char str[100];
    InputBoxGetLine("这是一个对话框",
                    "请随便\n输入一串字符，输入完请回车",
                    str,
                    sizeof(str)/sizeof(*str));
    outtextxy(0, 0, str);
    getch();
    return 0;
}
```


keystate

这个函数用于判断某按键是否被按下。

```
int keystate(int key);
```

参数:

key

虚拟键码, 一个 Windows 定义的, 能与键盘按键一一对应的码表。如果是字母键或者数字键, 则其虚拟键码与 ASCII 字符值相同, 比如 'A' 键, 它的虚拟键码也是 'A', 但不能是 'a'。小键盘上的数字则用类似 VK_NUMPAD3 的宏表示, 详细可以查看 VK_XXX 这一系列宏的定义。

返回值:

返回非0表示这个按键按下了, 返回0表示没有按下。该函数全局有效, 即使窗口没有得到输入焦点, 一样照样取得键盘的实际状态。

示例:

```
if (keystate(VK_ESCAPE))
{
    // ESC键按下了
}
```

random

这个函数用于生成某范围内的随机整数。

```
unsigned int random(unsigned int n = 0);
```

参数：

n

生成0至 n-1之间的整数。

如果 n 为0，则返回0 - 0xFFFFFFFF 的整数。

返回值：

返回一个随机整数。

其它说明：

建议不要使用 stdlib 里的 rand 函数，而改用本函数。本函数使用专业的随机数生成算法，随机性远超系统的 rand 函数。另，千万不要自己 random() % n 的方式取获得一个范围内的随机数，请使用 random(n)，切记。本随机序列的初始化只能调用 randomize 函数，不能使用 srand。

randomf

这个函数用于生成0-1范围内的随机浮点数。

```
double randomf();
```

参数:

无

返回值:

返回一个0-1之间的随机浮点数，0.0可能取得到，1.0一定取不到。

其它说明:

本随机序列的初始化只能调用 `randomize` 函数，不能使用 `srand`。

randomize

这个函数用于初始化随机数序列。如果不调用本函数，那么 random 返回的序列将会是确定不变的。

```
void randomize();
```

参数：

（无）

返回值：

（无）

示例：

（无）

示例程序

基础级别示例

字符阵

```
// 编译该程序，请先安装 EGE
#include "graphics.h"
#include <time.h>

int main()
{
    // 设置随机函数种子
    srand((unsigned) time(NULL));

    // 初始化图形模式
    initgraph(640, 480);

    int x, y;
    char c;

    setfont(16, 8, "Courier"); // 设置字体

    while(!kbhit())
    {
        for (int i=0; i<479; i++)
        {
            setcolor(GREEN);
            for (int j=0; j<3; j++)
            {
                x = (rand() % 80) * 8;
                y = (rand() % 20) * 24;
                c = (rand() % 26) + 65;
                outtextxy(x, y, c);
            }

            setcolor(0);
            line(0, i, 639, i);

            Sleep(10);
            if (kbhit()) break;
        }
    }

    // 关闭图形模式
    closegraph();
    return 0;
}
```

鼠标操作范例

```
// 编译该程序，请先安装 EGE
#include "graphics.h"
int main()
{
    // 初始化图形窗口
    initgraph(640, 480);

    MOUSEMSG m; // 定义鼠标消息

    while(true)
    {
        // 获取一条鼠标消息
        m = GetMouseMsg();

        switch(m.uMsg)
        {
            case WM_MOUSEMOVE:
                // 鼠标移动的时候画红色的小点
                putpixel(m.x, m.y, RED);
                break;

            case WM_LBUTTONDOWN:
                // 如果点左键的同时按下了 Ctrl 键
                if (m.mkCtrl)
                {
                    // 画一个大方块
                    rectangle(m.x-10, m.y-10, m.x+10, m.y+10);
                }
                else
                {
                    // 画一个小方块
                    rectangle(m.x-5, m.y-5, m.x+5, m.y+5);
                }
                break;

            case WM_RBUTTONDOWN:
                return; // 按鼠标右键退出程序
        }
    }

    // 关闭图形窗口
    closegraph();
    return 0;
}
```

彩虹

```
// 编译该程序，请先安装 EGE
#include "graphics.h"

int main()
{
    float H, S, L;

    initgraph(640, 480);

    // 画渐变的天空(通过亮度逐渐增加)
    H = 190; // 色相
    S = 1;    // 饱和度
    L = 0.7f; // 亮度
    for(int y = 0; y < 480; y++)
    {
        L += 0.0005f;
        setcolor( HSLtoRGB(H, S, L) );
        line(0, y, 639, y);
    }

    // 画彩虹(通过色相逐渐增加)
    H = 0;
    S = 1;
    L = 0.5f;
    setlinestyle(PS_SOLID, NULL, 2); // 设置线宽为 2
    for(int r = 400; r > 344; r--)
    {
        H += 5;
        setcolor( HSLtoRGB(H, S, L) );
        circle(500, 480, r);
    }

    getch();
    closegraph();
    return 0;
}
```

初等级别示例

星空

```
// 编译该程序，请先安装 EGE
// 本版本是修改版，真正的全屏流畅绘图
#include "graphics.h"

#include <time.h>
#include <stdio.h>

#define MAXSTAR 2000 // 星星总数

int sc_width, sc_height; // 记录窗口宽高

struct STAR
{
    double x;
    int y;
    double step;
    int color;
};

STAR star[MAXSTAR];

// 初始化星星
void InitStar(int i)
{
    star[i].x = 0;
    star[i].y = random(sc_height);
    star[i].step = random(50000) / 1000.0 + 1;
    star[i].color = (int)(star[i].step * 255 / 6.0 + 0.5); // 速度越快，颜色越亮
    if (star[i].color > 255)
    {
        star[i].color = 255;
    }
    star[i].color = RGB(star[i].color, star[i].color, star[i].color);
}

// 移动星星
void MoveStar(int i)
{
    // 擦掉原来的星星
    putpixel((int)star[i].x, star[i].y, 0);

    // 计算新位置
    star[i].x += star[i].step;
    if (star[i].x > sc_width) InitStar(i);

    // 画新星星
    putpixel((int)star[i].x, star[i].y, star[i].color);
}

// 主函数
int main()
```



```

{
    randomize(); // 初始化随机种子
    setinitmode(1, 0, 0); // 指定初始化为无边框窗口，并且窗口左上角坐标为(0, 0)
    initgraph(-1, -1); // 打开图形窗口，以全屏模式
    sc_width = getwidth();
    sc_heigh = getheight();

    // 初始化所有星星
    for(int i=0; i<MAXSTAR; i++)
    {
        InitStar(i);
        star[i].x = rand() % sc_width;
    }

    // 绘制星空，按任意键退出
    setfont(12, 6, "宋体");
    BeginBatchDraw();
    for(; kbhit() == 0; delay_fps(120)) //delay_fps已经调用了 FlushBatchDraw()
    {
        for(int i=0; i<MAXSTAR; i++)
        {
            MoveStar(i);
        }
        {
            float fps = GetFPS();
            char str[20];
            sprintf(str, "%.2f FPS", fps);
            outtextxy(0, 0, str); //显示fps
        }
    }
    EndBatchDraw();
    closegraph(); // 关闭图形窗口
    return 0;
}

```

滚动字幕

```
#include "graphics.h"

#include <stdio.h>
#include <time.h>

int main()
{
    initgraph(400, 300);
    setfont(24, 12, "宋体");
    {
        char str[] = "滚动字幕示例, Hello EGE !! Welcome to graphics programming !!!!!";
        int w = textwidth(str);          //记录下字幕的完整宽度
        int view_x = 100, view_w = 200; //设置可见区的位置和大小（只要x方向）
        int t = clock(), roll_time = 10000; //记录下起始时间，和滚动完所需要的时间
        BeginBatchDraw();
        for ( ; ; )
        {
            int nt = clock(); //取得当前时间，nt-t就是时间差
            // (nt-t) / roll_time 就是当前时间应该滚动的比例
            cleardevice();
            if (nt - t > roll_time) //和总时间比较，如果已经完成，t=nt重新设置起始时间
            {
                t = nt;
            }
            else
            {
                //设置绘图区域，在这区域外的不会绘画，以达到裁剪的目的
                //同时，坐标(0,0)将变成从这区域里开始算
                setviewport(view_x, 100, view_x + view_w, 300);
                //以下这个表达式: view_w - (w + view_w) * (nt-t) / roll_time 需要详细解释一下
                //我们要从右向左，那么，view_w最右端就是基准点
                //w + view_w 是我们需要滚动的总长度，总长度乘以当前的滚动比例，得到实际的位置
                outtextxy(view_w - (w + view_w) * (nt-t) / roll_time, 0, str);
                setviewport(0, 0, 400, 300); //还原绘图区
            }
            FlushBatchDraw();
        }
        EndBatchDraw();
    }
    return 0;
}
```

下雪

```
#include "graphics.h"

#define MAXSNOW 1000
struct SNOW
{
    int n;
    int lx, ly;
    int color;
    double x;
    double y;
    double cx;
    double dcx;
    double vx;
    double vy;
    double k;
};

int sc_width = 640;
int sc_height = 480;

void InitSnow(struct SNOW *snow)
{
    double vx = 0.5;
    double fvx = 0.5;
    double vy = (double)random(1000) / 1000;
    snow->x = random(sc_width);
    snow->y = 0;
    snow->cx = snow->x;
    snow->dcx = (double)random(1000) / 1000 * 2 * vx - vx;
    snow->k = (double)random(1000) / 1000 * -0.001;
    snow->vx = (double)random(1000) / 1000 * 2 * fvx - fvx;
    snow->vy = vy * 2.0 + 0.5;

    snow->color = (int)(192 * vy) + 64;
    snow->color = (((snow->color << 8) | snow->color) << 8) | snow->color;
}

void MoveSnow(struct SNOW *snow, double dt)
{
    double d = dt * 60 / 1000;

    putpixel(snow->lx, snow->ly, 0);

    snow->x += snow->dcx * d;
    snow->cx += snow->dcx * d;
    snow->y += snow->vy * d;

    if (snow->y > sc_height)
        InitSnow(snow);

    snow->lx = (int)snow->x;
    snow->ly = (int)snow->y;
    putpixel(snow->lx, snow->ly, snow->color);
}
```

```

int OnUpdate(void* param, float ms)
{
    struct SNOW* snow = (struct SNOW *)param;
    int i;
    for (i = 1; i <= snow->n; i++)
    {
        snow[i].vx += (snow[i].x - snow[i].cx) * snow[i].k * 0.99;
        snow[i].x += snow[i].vx;
    }
    return 0;
}

int OnRender(void* param, float ms)
{
    int i;
    for (i = 1; i <= ((struct SNOW*)param)->n; i++)
        MoveSnow((struct SNOW*)param+i, ms);
    return 0;
}

int main()
{
    {
        setinitmode(1, 0, 0);
        initgraph(-1, -1);
        sc_width = getwidth();
        sc_height = getheight();
        randomize();
    }
    {
        struct SNOW snow[MAXSNOW+1];
        int i;
        snow[0].n = MAXSNOW;
        for (i = 1; i <= MAXSNOW; i++)
        {
            InitSnow(snow + i);
            snow[i].y = random(sc_height);
        }
        {
            double t = fclock();
            BeginBatchDraw();
            for (int f=0; kbhit() == 0; delay_fps(120))
            {
                double dt = fclock();
                if (f++ == 1)
                {
                    f = 0;
                    OnUpdate(snow, 0);
                }
                OnRender(snow, (float)(dt-t) * 1000.0f);
                t = dt;
            }
            EndBatchDraw();
        }
    }

    closegraph();
    return 0;
}

```

中等级别示例

变幻线

```
#include "graphics.h"
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int width = 640, height = 480;

struct point //定义点, 包含坐标, 速度
{
    double x;
    double y;
    double dx;
    double dy;
};

struct poly //定义多边形, 包含点的个数, 和点数组
{
    int n_point;
    point p[20];
};

struct polys //定义多边形队列组
{
    int n_poly;           //多边形队列长度
    int color;            //颜色
    int nextcolor, prevcolor; //上一次的颜色, 目标颜色
    int ctime, nowtime;   //过渡变化时间, 当前时间
    int time;             //距离下次改变颜色的时间
    poly p[100];          //多边形数组
};

double rand_float(double dv, double db) //返回一个db 到 db+dv之间的随机浮点数
{
    return random(10000)*dv/10000 + db;
}

void movepoint(struct point* b) //根据点的速度属性移动这个点, 如果移出屏幕则进行反弹计算
{
    double dv = 2.0, db = 1.0;
    double tw = width / 640.0, th = height / 480.0;
    if (b->x < 0) b->dx = rand_float(dv, db) * tw;
    if (b->y < 0) b->dy = rand_float(dv, db) * th;
    if (b->x > width) b->dx = -rand_float(dv, db) * tw;
    if (b->y > height) b->dy = -rand_float(dv, db) * th;
    b->x += b->dx;
    b->y += b->dy;
}

void movepoly(struct poly* p) //移动单个多边形, 内部调用点的移动
{
    int i;
```

```

    for (i=0; i<p->n_point; ++i)
    {
        movepoint(&(p->p[i]));
    }
}

void movepolys(struct polys* p) //移动多边形队列，包含时间检测，颜色计算
{
    int i;
    for (i=p->n_poly-1; i>0; --i)
    {
        p->p[i] = p->p[i-1];
    }
    movepoly(p->p);
    ++(p->nowtime);
    if (-(p->time) <= 0)
    {
        p->prevcolor = p->color;
        p->nextcolor = HSVtoRGB((float)random(360), 1.0f, (float)rand_float(0.5, 0.5));
        p->time = random(1000);
        p->chtime = random(1000)+60;
        p->nowtime = 0;
    }
    if (p->nowtime >= p->chtime)
    {
        p->color = p->nextcolor;
    }
    else
    {
        double dr = p->prevcolor&0xFF, dg = (p->prevcolor>>8)&0xFF, db = (p->prevcolor>>16)&0xFF;
        double dt = 1 - p->nowtime / (double)(p->chtime);
        dr -= p->nextcolor&0xFF, dg -= (p->nextcolor>>8)&0xFF, db -= (p->nextcolor>>16)&0xFF;
        dr *= dt, dg *= dt, db *= dt;
        dr += p->nextcolor&0xFF, dg += (p->nextcolor>>8)&0xFF, db += (p->nextcolor>>16)&0xFF;
        p->color = ((int)dr) | ((int)dg<<8) | ((int)db<<16);
    }
}

void initpolys(struct polys* p, int npoly, int npoint) //初始化多边形队列组
{
    int i, j;
    p->n_poly = npoly;
    p->color = 0;
    p->time = 1000;
    p->prevcolor = p->color;
    p->nextcolor = HSVtoRGB((float)random(360), 1.0f, 0.5f);
    p->chtime = 1000;
    p->nowtime = 0;
    j = 0;
    p->p[j].n_point = npoint;
    for (i=0; i<npoint; ++i)
    {
        p->p[j].p[i].x = random(width);
        p->p[j].p[i].y = random(height);
        p->p[j].p[i].dx = (random(5)+1);
        p->p[j].p[i].dy = (random(5)+1);
    }
    for (j=1; j<npoly; ++j)

```

```

    {
        p->p[i] = p->p[i-1];
    }
}

void draw_poly(struct poly* p, int color) //绘制一个多边形
{
    int points[100];
    int i;
    for (i=0; i<p->n_point; ++i)
    {
        points[i*2] = (int)(p->p[i].x+.5f);
        points[i*2+1] = (int)(p->p[i].y+.5f);
    }
    points[i*2] = (int)(p->p[0].x+.5f);
    points[i*2+1] = (int)(p->p[0].y+.5f);
    setcolor(color);
    drawpoly(p->n_point+1, points);
}

void draw_polys(struct polys* p) //绘制多边形队列（只画第一个和最后一个，最后一个用于擦除）
{
    draw_poly(&(p->p[p->n_poly-1]), 0);
    draw_poly(&(p->p[0]), p->color);
}

void drawfps() //绘制帧数
{
    char str[100] = "";
    setfillstyle(0x0);
    sprintf(str, "帧率%.2f fps", GetFPS());
    bar(0, 0, textwidth(str), textheight(str));
    setcolor(0xFFFFFF);
    outtextxy(0, 0, str);
}

int main()
{
    static struct polys p[10] = {{0}};
    int n_points[10] = {3, 4, 5, 6, 7};
    int n_poly[10] = {40, 80, 10, 5, 1};
    int n_polys = 2, i;
    int fps=0, n_time = 0;
    char str[100]={0};
    randomize();
    //图形初始化
    {
        setinitmode(1, 0, 0);
        initgraph(-1, -1);
        width = getmaxx();
        height = getmaxy();
    }
    //多边形对象初始化
    for (i=0; i< n_polys; ++i)
    {
        initpolys(&p[i], n_poly[i], n_points[i]);
    }
    setfont(12, 6, "宋体");
    //主循环

```

```

BeginBatchDraw();
for ( ; ; delay_fps(120)) //内部已经调用了FlushBatchDraw()
{
    if (kbhit() > 0) //有按键按下就退出
    {
        break;
    }
    for (i=0; i< n_polys; ++i)
    {
        movepolys(&(p[i]));
    }
    for (i=0; i< n_polys; ++i)
    {
        draw_polys(&(p[i]));
    }
    drawfps();
}
EndBatchDraw();
closegraph();
return 0;
}

```


高等级别示例

俄罗斯方块

```
#include "graphics.h"

#include <time.h>
#include <stdio.h>
#include <string.h>

const int g_width = 400;
const int g_height = 520;

/*记录7种形状及其4种变化的表*/
static int g_trs_map[8][4][4][4];
/*变化数目表*/
static int g_map_mod[] = {1, 4, 4, 4, 2, 2, 2, 1, 0};

/*初始化全局数据及图形显示*/
void initgr() {
    initgraph(g_width, g_height);
    setfont(12, 6, "宋体");
    int Trs_map[8][4][4][4] =
    {
        {{0}}, {{
            {0, 0, 0, 0}, {1, 1, 1, 0}, {0, 1, 0, 0},
        }}, {
            {0, 1, 0, 0}, {1, 1, 0, 0}, {0, 1, 0, 0},
        }}, {
            {0, 1, 0, 0}, {1, 1, 1, 0},
        }}, {
            {0, 1, 0, 0}, {0, 1, 1, 0}, {0, 1, 0, 0},
        }}, {{
            {2, 2, 0, 0}, {0, 2, 0, 0}, {0, 2, 0, 0},
        }}, {
            {0, 0, 2, 0}, {2, 2, 2, 0},
        }}, {
            {0, 2, 0, 0}, {0, 2, 0, 0}, {0, 2, 2, 0},
        }}, {
            {0, 0, 0, 0}, {2, 2, 2, 0}, {2, 0, 0, 0},
        }}, {{
            {0, 3, 3, 0}, {0, 3, 0, 0}, {0, 3, 0, 0},
        }}, {
            {0, 0, 0, 0}, {3, 3, 3, 0}, {0, 0, 3, 0},
        }}, {
            {0, 3, 0, 0}, {0, 3, 0, 0}, {3, 3, 0, 0},
        }}, {
            {3, 0, 0, 0}, {3, 3, 3, 0},
        }}, {{
            {4, 4, 0, 0}, {0, 4, 4, 0},
        }}, {
            {0, 0, 4, 0}, {0, 4, 4, 0}, {0, 4, 0, 0},
        }}, {{
            {0, 5, 5, 0}, {5, 5, 0, 0},
        }}, {
            {0, 5, 0, 0}, {0, 5, 5, 0}, {0, 0, 5, 0},
        }}, {{
            {0, 0, 0, 0}, {6, 6, 6, 6},
        }}, {
            {0, 0, 6, 0}, {0, 0, 6, 0}, {0, 0, 6, 0}, {0, 0, 6, 0},
        }}, {{
            {0, 0, 0, 0}, {0, 7, 7, 0}, {0, 7, 7, 0},
        }},
    },
    };
    memcpy(g_trs_map, Trs_map, sizeof(Trs_map));
}

class Game {
public:
    /*状态表*/
    enum {
        ST_START, /*游戏重新开始*/
        ST_NEXT, /*准备下一个方块*/
        ST_NORMAL, /*玩家控制阶段*/
        ST_OVER /*游戏结束, F2重新开始*/
    };
    Game(int w, int h, int bw, int bh) {
        int colormap[10] = {0, 0xA0, 0x50A0, 0xA0A0, 0xC000,
```

```

    0xA0A000, 0xC04040, 0xA000A0, 0x808080, 0xFFFFFFFF};
memcpy(m_colormap, colormap, sizeof(m_colormap));

int Keys[8] = {VK_F2, VK_LEFT, VK_RIGHT, VK_DOWN, VK_UP, VK_NUMPAD0, VK_SPACE};
memcpy(m_Keys, Keys, sizeof(Keys));

memset(m_KeyState, 0, sizeof(m_KeyState));
m_gamepool_w = w;
m_gamepool_h = h;
m_base_w = bw;
m_base_h = bh;

randomize();
m_ctl_t = -1;
m_pcb = new IMAGE;
for (int i=0; i<10; ++i) {
    drawtile(bw * i, 0, bw, bh, 5, colormap[i]);
}
getimage(m_pcb, 0, 0, bw*10, bh);
m_state = ST_START;
}
/*状态转换处理*/
int deal () {
    int nRet = 0;
    if ( m_state == ST_START ) { //初始化
        m_next1_s = random(7) + 1;
        m_next2_s = random(7) + 1;
        m_pause = 0;
        memset(m_gamepool, 255, sizeof(m_gamepool));
        for (int y = 1; y <= m_gamepool_h; ++y) {
            for (int x = 1; x <= m_gamepool_w; ++x)
                m_gamepool[y][x] = 0;
        }
        m_forbid_down = 0;
        m_ctl_t = -1;
        nRet = 1;
        m_state = ST_NEXT;
    } else if ( m_state == ST_NEXT ) {
        m_ctl_x = (m_gamepool_w - 4) / 2 + 1;
        m_ctl_y = 1;
        m_ctl_t = 0;
        m_ctl_s = m_next1_s;
        m_ctl_dy = 0;
        m_next1_s = m_next2_s;
        m_next2_s = random(7) + 1;
        m_curtime = m_droptime;
        m_curxtime = 0;
        nRet = 1;
        if ( isCrash() ) {
            m_gray_y = m_gamepool_h * 2;
            m_over_st = 0;
            m_state = ST_OVER;
        } else {
            m_state = ST_NORMAL;
        }
    } else if ( m_state == ST_NORMAL ) {
        /*处理自由下落*/
        int i, j;
        if ( m_KeyState[3] == 0 || m_forbid_down ) {
            --m_curtime, m_cursubtime = 1;
        }
        if ( m_curxtime ) {
            if (m_curxtime<0)
                m_curxtime++;
            else
                m_curxtime--;
        }
        /*按键处理*/
        for (i = 1, j = 1; i<=2; ++i, j--=2) {
            for ( ; m_KeyFlag[i] > 0; --m_KeyFlag[i]) {
                m_ctl_x -= j;
                if ( isCrash() )
                    m_ctl_x += j;
                else
                    m_curxtime = m_movxtime * j;
            }
        }
        m_ctl_dx = float(double(m_curxtime) / m_movxtime); //处理x方向平滑
        for (i = 4, j = 1; i<=5; ++i, j--=2) {
            for (int t ; m_KeyFlag[i] > 0; --m_KeyFlag[i]) {
                m_ctl_t=((t=m_ctl_t)+g_map_mod[m_ctl_s]+j)%g_map_mod[m_ctl_s];
                if ( isCrash() ) m_ctl_t = t;
            }
        }
        if ( m_forbid_down == 0 && (m_KeyState[3] ) ) {
            m_curtime -= m_cursubtime++;
        }
    }
}

```

```

        if (m_curtime<0) {
            ++m_ctl_y;
            if ( isCrash() ) {
                --m_ctl_y;
                merge();
                m_ctl_dy = 0; m_ctl_dx = 0; m_ctl_t = -1;
                if ( m_KeyState[3] )
                    m_forbid_down = 1;
                m_state = ST_NEXT;
            } else {
                m_curtime += m_droptime;
            }
        }
        if (m_state == ST_NORMAL) {
            m_ctl_dy = float(double(m_curtime) / m_droptime); //处理y方向平滑
        }
    } else if (m_state == ST_OVER) {
        if ( m_gray_y>0 && (m_gray_y % 2) == 0 )
            for (int x = 1; x <= m_gamepool_w; ++x)
                if ( m_gamepool[m_gray_y>>1][x] )
                    m_gamepool[m_gray_y>>1][x] = 8;
        m_gray_y--;
        ++m_over_st;
        if ( m_KeyFlag[0] > 0 )
            m_state = ST_START;
    }
    memset(m_KeyFlag, 0, sizeof(m_KeyFlag));
    return nRet;
}
/*碰撞检测*/
bool isCrash() {
    for (int y=0; y<4; ++y) {
        for (int x=0; x<4; ++x)
            if ( g_trs_map[m_ctl_s][m_ctl_t][y][x] ) {
                if ( m_ctl_y + y < 0 || m_ctl_x + x < 0
                    || m_gamepool[m_ctl_y + y][m_ctl_x + x] )
                    return true;
            }
    }
    return false;
}
void merge() {
    int y, x, cy = m_gamepool_h;
    /*合并处理*/
    for (y=0; y<4; ++y) {
        for (x=0; x<4; ++x)
            if ( g_trs_map[m_ctl_s][m_ctl_t][y][x] )
                m_gamepool[m_ctl_y + y][m_ctl_x + x]
                    = g_trs_map[m_ctl_s][m_ctl_t][y][x];
    }
    /*消行计算*/
    for (y = m_gamepool_h; y >= 1; --y) {
        for (x = 1; x <= m_gamepool_w; ++x) {
            if ( m_gamepool[y][x] == 0 )
                break;
        }
        if ( x <= m_gamepool_w ) {
            if ( cy != y ) {
                for (x = 1; x <= m_gamepool_w; ++x)
                    m_gamepool[cy][x] = m_gamepool[y][x];
            }
            --cy;
        }
    }
    for (y = cy; y >= 1; --y) {
        for (x = 1; x <= m_gamepool_w; ++x)
            m_gamepool[y][x] = 0;
    }
}
/*逻辑更新主函数*/
void update(int k) {
    while ( k && (k = getch(1)) ) {
        for (int i=0; i<8; ++i) {
            if ((k & 0xFFFF) == (m_Keys[i] & 0xFFFF)) {
                if (k & KEYMSG_DOWN) {
                    m_KeyFlag[i]++;
                    m_KeyState[i] = 1;
                } else if (k & KEYMSG_UP) {
                    m_KeyFlag[i] = 0;
                    m_KeyState[i] = 0;
                    if ( i == 3 )
                        m_forbid_down = 0;
                }
            }
        }
    }
    k = kbhit(1);
}

```

```

    while ( deal() );
}
void drawedge(int x, int y, int w, int h, int color, int bdark = 1) {
    setcolor(getchangelcolor(color, 1.5f));
    line(x, y+h, x, y);
    lineto(x+w, y);
    if ( bdark )
        setcolor(getchangelcolor(color, 0.7f));
    lineto(x+w, y+h);
    lineto(x, y+h);
}
void drawtile(int x, int y, int w, int h, int d, int color) {
    w--, h--;
    setfillstyle(color);
    bar(x+1, y+1, x+w, y+h);
    drawedge(x, y, w, h, color);
    drawedge(x+1, y+1, w-2, h-2, color);
}
void drawframe(int x, int y, int w, int h, int d = 0) {
    int coll[] = {0x400040, 0x600060, 0xA000A0, 0xFF00FF,
        0xA000A0, 0x600060, 0x400040};
    setfillstyle(0x010101);
    bar(x, y, x + w--, y + h--);
    for (int i=0; i<7; ++i) {
        --x, --y, w += 2, h += 2;
        drawedge(x, y, w, h, coll[i], 0);
    }
}
void draw44(int bx, int by, int mat[4][4],
    float dx=0, float dy=0, int nc=0, int deep=5) {
    for (int y = 3; y >= 0; --y) {
        for (int x = 0, c; x < 4; ++x) {
            if ( c = mat[y][x] ) {
                if ( nc ) c = nc;
                drawtile(int(bx + (x + dx) * m_base_w + 1000.5) - 1000,
                    int(by + (y - dy) * m_base_h + 1000.5) - 1000,
                    m_base_w, m_base_h, deep,
                    m_colormap[c]);
            }
        }
    }
}
void drawfps() {
    static char str[100];
    static int n_time = 0, fps = 0;
    setcolor(0xFFFFF);
    setbkmode(TRANSPARENT);
    sprintf(str, "帧率 %.2f fps", GetFPS());
    outtextxy(0, 0, str);
}
/*图形更新主函数*/
void render() {
    int x, y, c, bx, by, deep = 5;
    /*画背景框*/
    cleardevice();
    drawframe( m_base_x + 5 * m_base_w,
        m_base_y,
        m_gamepool_w * m_base_w,
        m_gamepool_h * m_base_h);
    drawframe(m_base_x, m_base_y, 4*m_base_w, 4*m_base_h);
    drawframe(m_base_x, m_base_y + 5*m_base_h, 4*m_base_w, 4*m_base_h);
    /*画主游戏池*/
    bx = m_base_x + 4 * m_base_w;
    by = m_base_y - 1 * m_base_h;
    for (y = m_gamepool_h; y >= 1; --y) {
        for (x = 1; x <= m_gamepool_w; ++x) {
            if ( c = m_gamepool[y][x] )
                putimage(bx + x * m_base_w, by + y * m_base_h,
                    m_base_w, m_base_h, m_pcb,
                    c * m_base_w, 0);
        }
    }
    /*画控制块*/
    if ( m_ctl_t >= 0 ) {
        bx = m_base_x + (m_ctl_x + 4) * m_base_w;
        by = m_base_y + (m_ctl_y - 1) * m_base_h;
        draw44(bx, by, g_trs_map[m_ctl_s][m_ctl_t], m_ctl_dx, m_ctl_dy);
    }
    /*画下一块和下二块*/
    bx = m_base_x;
    by = m_base_y;
    draw44(bx, by, g_trs_map[m_next1_s][0]);
    bx = m_base_x;
    by = m_base_y + 5 * m_base_h;
    draw44(bx, by, g_trs_map[m_next2_s][0], 0, 0, 8);
    setcolor(0xFFFFF);
    if ( m_state == ST_OVER ) { // 结束提示文字显示

```

```

        outtextxy(m_base_x+5*m_base_w, m_base_y, "Press F2 to Restart game");
    }
    drawfps();
}
static int dealbit(int a, float dt) {
    a = int(a * dt);
    if ( a>255 ) a = 255;
    else if ( a<0 ) a = 0;
    return a;
}
static int getchangcolor(int Color, float t) {
    int r = GetRValue(Color), g = GetGValue(Color), b = GetBValue(Color);
    r = dealbit(r, t);
    g = dealbit(g, t);
    b = dealbit(b, t);
    return RGB(r, g, b);
}
public:
    int m_base_x, m_base_y, m_base_w, m_base_h;
    int m_droptime;
    int m_curttime;
    int m_cursubtime;
    int m_movxtime;
    int m_curxtime;
private:
    int m_gamepool_w, m_gamepool_h;
    int m_gamepool[30][30]; //从1为起始下标, 0用于边界碰撞检测
    int m_ctl_x, m_ctl_y, m_ctl_t, m_ctl_s; //当前控制块属性
    float m_ctl_dx, m_ctl_dy;
    int m_next1_s, m_next2_s;
    int m_forbid_down;
    int m_colormap[10];
public:
    int m_pause;
    int m_state; //游戏主状态
    int m_gray_y;
    int m_over_st;
    int m_Keys[8];
    int m_KeyFlag[8];
    int m_KeyState[8];
private:
    IMAGE* m_pcb;
};

int main() {
    int fps = 120;
    initgr();

    Game game(10, 20, 24, 24);
    game.m_base_x = 20;
    game.m_base_y = 20;
    game.m_droptime = fps/2;
    game.m_movxtime = 10;

    BeginBatchDraw();
    for (int k; (k = kbhit(1)) != -1; delay_fps(fps)) {
        game.update(k);
        game.render();
    }
    EndBatchDraw();
    return 0;
}

```

与 Borland BGI 绘图库的兼容情况

基于以下几个原因，EGE 库中的函数原型不再仿效 Borland BGI 的绘图库：

- 当初写 EGE 库的时候，打算将 Turbo C 2.0 下面基于 BGI 绘图库的程序移植到 VC 下。我曾经一个个的模拟了几乎所有的 BGI 绘图函数、常用中断函数。但是在模拟的过程中，发现了一些 BGI 绘图库的不足。
- BGI 的许多数据定义在 windows.h 中都有，只不过名字不同。记忆两套名字容易混淆，毕竟，BGI 绘图库已经是历史了。EGE 库部分复用了 Windows 中的定义。
- Borland 早就停止更新 BGI 了，可是 EGE 库还在不停的完善，部分接口为了实际编程更方便，设计的和 BGI 并不相同。不过基本绘图函数接口会保证在以后的版本都不会发生变化的，所以大家可以放心使用。新版本发布的时候，只会增加新接口，或者扩展原接口，或者修正接口的 BUG。

所以，请不要把本库当成 BGI 的移植库使用，但老版本 EGE 写出来的代码，一定可以无缝移植到新版本上。

如果您实在需要移植原 Turbo C 的绘图程序，需要注意以下几个方面（改动的量完全看那个程序的长度和框架，有些框架是根本无法移植的，像中断调用）：

- 颜色。EGE 库支持了真彩色，这点与 TC 差别很大。
- getimage / putimage，这组函数较 BGI 做了很多扩充。
- 设置线性、填充类型的函数，和 BGI 的略有区别。
- 位操作，BGI 的功能很局限，EGE 库实现了所有的位操作。
- 字体操作，不使用 settextstyle，而改使用 setfont。
- 其他扩充函数，例如鼠标、批操作等。

联系我们

当前 EGE 版本:

20110401

EGE 官方论坛社区:

<http://easyx.uueasy.com>

EGE 作者:

rtGirl

QQ 群:

143350042

UC 群:

10240121