EasyX 进阶

图像处理

https://docs.easyx.cn/zh-cn/image-func

函数用法	函数说明
void **loadimage** (IMAGE* pDstImg, // 保存图像的 IMAGE 对象指针 LPCTSTR pImgFile, // 图片文件名 int nWidth = 0, // 图片的拉伸宽度 int nHeight = 0, // 图片的拉伸高度 bool bResize = false //是否调整IMAGE的大小以适应图片)	从文件中读取图像。如果pDstImg为NULL，则读取到绘图窗口
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);	在当前设备上绘制指定图像
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)	在当前设备上绘制指定图像（指定宽高和起始位置）
void **Resize** (IMAGE* pImg, int width, int height)	调整指定绘图设备的尺寸，pImg 如果为 NULL 表示默认绘图窗口

函数用法	函数说明
void **rotateimage** (IMAGE *dstimg, IMAGE *srcimg, double radian, COLORREF bkcolor = BLACK, bool autosize = false, bool highquality = true)	旋转 IMAGE 中的绘图内容
void **saveimage** (LPCTSTR strFileName, IMAGE* pImg = NULL)	保存绘图内容至图片文件，支持 bmp / gif / jpg / png / tif 格式
void **SetWorkingImage** (IMAGE* pImg = NULL)	设定当前的绘图设备，如果参数为 NULL，表示绘图设备为默认绘图窗口
IMAGE* **GetWorkingImage** ()	获取当前的绘图设备，如果返回值为 NULL，表示当前绘图设备为绘图窗口
void **getimage** (IMAGE* pDstImg, // 保存图像的 IMAGE 对象指针 int srcX, // 要获取图像区域左上角 x 坐标 int srcY, // 要获取图像区域的左上角 y 坐标 int srcWidth, // 要获取图像区域的宽度 int srcHeight // 要获取图像区域的高度)	从当前绘图设备中获取图像
DWORD* **GetImageBuffer** (IMAGE* pImg = NULL)	获取绘图设备的显示缓冲区指针，pImg 如果为 NULL，表示默认的绘图窗口
HDC **GetImageHDC** (IMAGE* pImg = NULL)	获取绘图设备句柄(HDC)

IMAGE 类

```
class IMAGE(int width = 0, int height = 0);

公有成员
int getwidth();
返回 IMAGE 对象的宽度，以像素为单位。

int getheight();
返回 IMAGE 对象的高度，以像素为单位。

operator =
实现IMAGE对象的直接赋值。该操作仅拷贝源图像的内容，不拷贝源图像的绘图环境。
```

在内存中保存图像信息。

loadimage 函数

范例1：loadimage 直接读取图片至绘图窗口

```
#include <graphics.h>

int main()
{
    initgraph(1400, 600);

    loadimage(NULL, _T("image\\background.jpg"));    // 第一个参数为NULL时，直接读取图片至绘图窗口

    system("pause");
    closegraph();
    return 0;
}
```

注：修改窗口大小，可以显示图片部分内容，但只能从绘图窗口的坐标原点（左上角）开始显示图片

范例2：loadimage 直接读取图片至绘图窗口并进行图片或窗口缩放

```
#include <graphics.h>

int main()
{
    initgraph(700, 300);

    loadimage(NULL, _T("image\\background.jpg"), 700, 300, false);  // 将图像缩放为700*300在绘图窗口显示

    system("pause");
    closegraph();
    return 0;
}
```

注1：图片缩放后的尺寸小于窗口尺寸，则窗口会有黑边；若大于窗口尺寸，则图片显示不全

注2：第五个参数若为 true，则会调整窗口以适应图片的大小

范例3：loadimage 读取本地图片文件，输出图片宽度和高度

```
#include <graphics.h>
#include <stdio.h>

int main()
{
    initgraph(1000, 600, SHOWCONSOLE);       // 初始化绘图窗口并开启终端

    IMAGE img;                                // 定义图像对象
    loadimage(&img, _T("image\\background.jpg"));                                // 读取本地图片文件，存入图像对象
    printf("width=%d, height=%d \n", img.getwidth(), img.getheight());       // 输出图像宽度和高度

    system("pause");
    closegraph();
    return 0;
}
```

注：本例中的图片内容不会在窗口内显示

putimage 函数

范例1：putimage 在绘图窗口显示图像

```
#include <graphics.h>

int main()
{
    initgraph(1000, 600);

    IMAGE img;
    loadimage(&img, _T("image\\background.jpg"));
    putimage(0, 0, &img);            // 将图像对象显示在绘图窗口的坐标（0,0）处

    system("pause");
    closegraph();
    return 0;
}
```

注1：从磁盘中读取大量图片显示的情况下，使用 loadimage 直接读取图片至绘图窗口性能较差

注2：putimage 第四个参数是 **三元光栅操作码** ，它定义了源图像与目标图像的位合并形式，默认值为 **SRCCOPY** 详见

https://docs.easyx.cn/zh-cn/putimage

范例2：putimage 截取图像部分内容进行显示

```
#include <graphics.h>

int main()
{
    initgraph(900, 600);

    IMAGE img;
    loadimage(&img, _T("image\\background.jpg"));
    putimage(0, 0, 900, 600, &img, 115, 0);          // 从图像的(115,0)坐标处截取宽900、高600的部分内容显示在窗口(0,0)处

    system("pause");
    closegraph();
    return 0;
}
```

透明贴图

范例1：通过PS制作**原图**的**掩码图**和**前景图**，再进行三元光栅操作叠加而成

```
#include <graphics.h>

int main()
{
    IMAGE imgGuoqi, imgGuohui, imgGuohuiMask, imgGuohuiFg;
    loadimage(&imgGuoqi, _T("image\\guoqi.jpg"), 1000, 600);            // 加载国旗（背景图）
    loadimage(&imgGuohui, _T("image\\guohui.jpg"), 200, 200);           // 加载国徽原图（白色周边）
    loadimage(&imgGuohuiMask, _T("image\\guohui_mask.jpg"), 200, 200);  // 加载国徽掩码图（白色周边+黑色内容）
    loadimage(&imgGuohuiFg, _T("image\\guohui_fg.jpg"), 200, 200);      // 加载国徽前景图（黑色周边+待显示内容）

    initgraph(1000, 600);

    putimage(0, 0, &imgGuoqi);                                  // 显示国旗
    putimage(0, 0, &imgGuohui);                                 // 显示国徽原图
    putimage(0, 200, &imgGuohuiMask);                           // 显示国徽掩码图
    putimage(0, 400, &imgGuohuiFg);                             // 显示国徽前景图

    // 透明贴图
    putimage(200, 0, &imgGuohuiMask, SRCAND);                   // 显示掩码图（SRCAND：按位与）
    putimage(200, 0, &imgGuohuiFg, SRCPAINT);                   // 显示前景图（SRCPAINT：按位或）

    system("pause");
    closegraph();
    return 0;
}
```

范例2：TransparentBlt 函数实现

```
#include <graphics.h>
#pragma comment(lib, "MSIMG32.LIB")                        // 链接器在链接过程中包含指定的库文件

void putimage_alpha(IMAGE* dstImg, int x, int y, IMAGE* srcImg, UINT transparentColor)
{
    HDC dstDC = GetImageHDC(dstImg);
    HDC srcDC = GetImageHDC(srcImg);
    int w = srcImg->getwidth();
    int h = srcImg->getheight();
    TransparentBlt(dstDC, x, y, w, h, srcDC, 0, 0, w, h, transparentColor);
}

int main()
{
    initgraph(1000, 600);

    IMAGE imgGuoqi, imgBaidu;
    loadimage(&imgGuoqi, _T("image\\guoqi.jpg"), 1000, 600);        // 加载国旗（背景图）
    loadimage(&imgBaidu, _T("image\\baidu.png"));                   // 加载百度LOGO（PNG格式）
    putimage(0, 0, &imgGuoqi);                                      // 显示国旗
    putimage(0, 0, &imgBaidu);                                      // 显示百度LOGO
    putimage_alpha(NULL, 0, 300, &imgBaidu, BLACK);                 // 显示百度LOGO（透明贴图）

    system("pause");
    closegraph();
    return 0;
}
```

函数说明：第1个参数为NULL，第2、3个参数是输出坐标，第4个参数是需要透明显示的图片，第5个参数是要透明的底色（若图片是透明图片，默认为BLACK）

注：此方法只支持 PNG 格式的图片

范例3：AlphaBlend 函数实现（**推荐**）

```
#include <graphics.h>
#pragma comment(lib, "MSIMG32.LIB")                        // 链接器在链接过程中包含指定的库文件

void putimage_alpha(int x, int y, IMAGE* img)
{
    int w = img->getwidth();
    int h = img->getheight();
```

```
    AlphaBlend(GetImageHDC(NULL), x, y, w, h, GetImageHDC(img), 0, 0, w, h, {
AC_SRC_OVER, 0, 255, AC_SRC_ALPHA });
}

int main()
{
    initgraph(1000, 600);

    IMAGE imgGuoqi, imgBaidu;
    loadimage(&imgGuoqi, _T("image\\guoqi.jpg"), 1000, 600);        // 加载国旗（背
景图）
    loadimage(&imgBaidu, _T("image\\baidu.png"));                   // 加载百度
LOGO（PNG格式）
    putimage(0, 0, &imgGuoqi);                                      // 显示国旗
    putimage(0, 0, &imgBaidu);                                      // 显示百度
LOGO
    putimage_alpha(0, 300, &imgBaidu);                              // 显示百度
LOGO（透明贴图）

    system("pause");
    closegraph();
    return 0;
}
```

注：此方法只支持 PNG 格式的图片

图片动画

图片动画的核心是**一系列静态的图像（动画帧）**。每一帧都是一张静态的图片，但它们之间略有不同，通常表现为物体的位置、形状或颜色的微小变化。这些帧按照特定的顺序排列，并以一定的速度连续播放，使得观者感受到运动的效果。

范例：

```
#include <graphics.h>
#pragma comment(lib, "MSIMG32.LIB")

const int WINDOW_WIDTH = 1000;          //窗口宽度
const int WINDOW_HEIGHT = 600;          //窗口高度
const int FRAME = 60;                   //帧数
const int INTERVAL_MS = 15;             //动画帧间隔
const int IMAGE_NUM = 13;               //动画图片数

//显示透明图片
void putimage_alpha(int x, int y, IMAGE* img)
{
    int w = img->getwidth();
    int h = img->getheight();
    AlphaBlend(GetImageHDC(NULL), x, y, w, h, GetImageHDC(img), 0, 0, w, h, {
AC_SRC_OVER, 0, 255, AC_SRC_ALPHA });
}
```

```
int main()
{
    bool running = true;            //主循环控制
    ExMessage msg;                  //键鼠消息
    IMAGE imgBackground;            //背景图片对象
    IMAGE imgPEA[13];               //玩家动画图片
    TCHAR imgPath[256];             //动画图片文件路径
    int imgIndex = 0;               //动画帧索引
    static int timer = 0;           //动画计时器

    loadimage(&imgBackground, _T("image\\background.jpg"));     //加载背景图片
    for (int i = 0; i < IMAGE_NUM; i++)                         //加载动画图片
    {
        _stprintf_s(imgPath, _T("image\\pea\\%d.png"), i + 1);  //动画图片路径（格式转换）
        loadimage(&imgPEA[i], imgPath);                         //加载动画图片
    }

    initgraph(WINDOW_WIDTH, WINDOW_HEIGHT);
    BeginBatchDraw();

    //主循环
    while (running)
    {
        DWORD beginTime = GetTickCount();

        //消息处理
        while (peekmessage(&msg))
        {
        }

        //数据处理
        timer += 5;
        if (timer > INTERVAL_MS)                    //定时器超过预定的时间间隔时切换下一张图片
        {
            imgIndex = (imgIndex + 1) % IMAGE_NUM;          //循环切换图片：索引值0-12
            timer = 0;                                      //重置计时器
        }

        //绘图
        cleardevice();
        putimage(0, 0, &imgBackground);                     //绘制背景图片
        putimage_alpha(500, 300, &imgPEA[imgIndex]);        //绘制豌豆图片
        FlushBatchDraw();

        //帧延时处理
        DWORD endTime = GetTickCount();
        DWORD elapsedTime = endTime - beginTime;
        if (elapsedTime < 1000 / FRAME)
            Sleep(1000 / FRAME - elapsedTime);
    }

    EndBatchDraw();
```

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

Resize 函数

范例1：调整指定绘图设备的尺寸

```
#include <graphics.h>
#include <stdio.h>

int main()
{
    initgraph(1000, 700, 1);

    IMAGE img;
    loadimage(&img, _T("image\\background.jpg"), 900, 600);     // 加载并缩放图片尺寸为900*600

    putimage(0, 0, &img);
    printf("调整前图片尺寸：width=%d, height=%d\n", img.getwidth(), img.getheight());

    system("pause");
    cleardevice();
    Resize(&img, 600, 400);          // 调整IMAGE的尺寸，注意不是缩放
    putimage(0, 0, &img);
    printf("第1次调整后图片尺寸：width=%d, height=%d\n", img.getwidth(), img.getheight());

    system("pause");
    cleardevice();
    Resize(&img, 800, 600);          // 调整IMAGE的尺寸，注意不是缩放
    putimage(0, 0, &img);
    printf("第2次调整后图片尺寸：width=%d, height=%d\n", img.getwidth(), img.getheight());

    system("pause");
    cleardevice();
    Resize(NULL, 600, 400);          // 第一个参数为NULL，则调整窗口的尺寸
    putimage(0, 0, &img);
    printf("第3次调整后图片尺寸：width=%d, height=%d\n", img.getwidth(), img.getheight());

    system("pause");
    closegraph();
    return 0;
}
```

GetImageBuffer 函数

范例：GetImageBuffer 通过直接操作显示缓冲区绘制渐变的蓝色

```
#include <graphics.h>

int main()
{
    initgraph(600, 400);

    DWORD* pMem = GetImageBuffer();                              // 获取当前窗口所指图像
缓冲区的指针
    for (int i = 0; i < 600 * 400; i++)
        pMem[i] = BGR(RGB(0, 0, i * 256 / (600 * 400)));     // 直接对图像缓冲区每个
坐标像素赋值（颜色）

    system("pause");
    closegraph();
    return 0;
}
```

图像翻转

```
#include <graphics.h>

// 图像翻转
void flip_image(IMAGE* srcImg, IMAGE* dstImg)
{
    int w = srcImg->getwidth();                                  // 获取源图像宽度
    int h = srcImg->getheight();                                 // 获取源图像高度
    Resize(dstImg, w, h);                                        // 设置目标图像与源图像
宽高一致
    DWORD* src_buffer = GetImageBuffer(srcImg);                  // 获取源图像缓冲区指针
    DWORD* dst_buffer = GetImageBuffer(dstImg);                  // 获取目标图像缓冲区指
针

    for (int y = 0; y < h; y++)
    {
        for (int x = 0; x < w; x++)
        {
            int idx_src = y * w + x;
            int idx_dst = y * w + (w - x - 1);
            dst_buffer[idx_dst] = src_buffer[idx_src];       // 交换对应坐标像素的颜
色值
        }
    }
}

int main()
{
    initgraph(1400, 600);
```

```
    IMAGE img1, img2;
    loadimage(&img1, _T("image\\background.jpg"));
    flip_image(&img1, &img2);
    putimage(0, 0, &img2);

    system("pause");
    closegraph();
    return 0;
}
```

其它函数

范例：**rotateimage** 旋转图像后用 **saveimage** 保存到文件

```
#include <graphics.h>
#define PI 3.14159

int main()
{
    initgraph(900, 600);

    IMAGE img1, img2;
    loadimage(&img1, _T("image\\background.jpg"));
    putimage(0, 0, &img1);

    system("pause");
    cleardevice();
    rotateimage(&img2, &img1, PI/4);                // 将img1逆时针旋转45度，结果保存到img2
    putimage(0, 0, &img2);                          // 显示旋转后的图像
    saveimage(_T("C:\\newimg.jpg"), &img2);         // 保存旋转后的图像到本地文件

    system("pause");
    closegraph();
    return 0;
}
```

注：rotateimage 旋转后产生的空白区域的颜色默认为黑色，可以通过第四个参数指定

范例：SetWorkingImage 指定绘图设备

```
#include <graphics.h>

int main()
{
    initgraph(600, 600);

    IMAGE img(200, 200);                    // 创建200x200像素的IMAGE对象
```

```
    SetWorkingImage(&img);                  // 设置绘图目标为IMAGE对象，后续绘图操作绘制在该对象中
    circle(100, 100, 50);
    line(0, 100, 200, 100);
    line(100, 0, 100, 200);

    SetWorkingImage();                      // 设置绘图目标为当前窗口
    putimage(200, 200, &img);               // 将IMAGE对象显示在窗口(200,200)坐标处

    system("pause");
    closegraph();
    return 0;
}
```

范例：GetImageHDC 实现 Windows GDI 函数操作

```
#include <graphics.h>

int main()
{
    initgraph(600, 400);

    HDC hdc = GetImageHDC();                // 获取默认绘图窗口的 HDC 句柄
    MoveToEx(hdc, 0, 0, NULL);              // 执行Windows GDI绘图函数，移动画笔至坐标（0,0）
    LineTo(hdc, 599, 399);                  // 从当前坐标（0,0）到坐标（599,399）画直线

    IMAGE img(200, 200);                    // 创建大小为 200x200 的 img 对象
    hdc = GetImageHDC(&img);                // 获取该 img 对象的 HDC 句柄
    Ellipse(hdc, 0, 50, 199, 150);          // 执行 Windows GDI 绘图函数
    putimage(100, 100, &img);               // 将 img 对象显示到绘图窗口上面

    system("pause");
    closegraph();
    return 0;
}
```

双缓冲绘图

定义：双缓冲绘图即在内存中创建一个与屏幕绘图区域一致的对象，先将图形绘制到内存中的这个对象上，再一次性将这个对象上的图形拷贝到屏幕上。

目的：通过减少屏幕的直接绘图操作，来加快绘图速度并消除闪烁现象。

函数用法	函数说明

函数用法	函数说明
void **BeginBatchDraw** ()	开始批量绘图
void **EndBatchDraw** () void **EndBatchDraw** (int left, int top, int right, int bottom) // 指定区域	结束批量绘制，并执行（指定区域内）未完成的绘制任务
void **FlushBatchDraw** () void **FlushBatchDraw** (int left, int top, int right, int bottom) // 指定区域	执行（指定区域内）未完成的绘制任务

https://docs.easyx.cn/zh-cn/other-func

范例1：自动移动的圆

```
#include <graphics.h>

int main()
{
    initgraph(640, 480);
    // BeginBatchDraw();            // 开启批量绘图

    setlinecolor(WHITE);            // 设置画线颜色
    setfillcolor(RED);              // 设置填充颜色
    for (int i = 50; i < 600; i++)
    {
        cleardevice();              // 清屏
        circle(i, 100, 40);         // 绘制空心圆
        floodfill(i, 100, WHITE);   // 用白色填充
        // FlushBatchDraw();        // 刷新批量绘图
        Sleep(10);                  // 延时10毫秒
    }

    // EndBatchDraw();              // 关闭批量绘图
    closegraph();
    return 0;
}
```

范例2：自动移动的圆（帧数控制）

```
//include <windows.h>
#include <graphics.h>

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

    setlinecolor(WHITE);
    setfillcolor(RED);
```

```
    for (int i = 50; i < 600; i++)
    {
        DWORD beginTime = GetTickCount();                       // 记录循环开始时间

        cleardevice();
        circle(i, 100, 40);
        floodfill(i, 100, WHITE);
        FlushBatchDraw();

        DWORD endTime = GetTickCount();                         // 记录循环结束时间
        DWORD elapsedTime = endTime - beginTime;                // 计算循环耗时
        if (elapsedTime < 1000 / 60)                            // 按每秒60帧进行补
时
            Sleep(1000 / 60 - elapsedTime);
    }

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

GetTickCount 是一个 Windows 系统函数，用于**获取从操作系统启动以来所经过的毫秒数**，通过在代码中的不同位置调用该函数，并计算两次调用之间的差值，可以得知某段代码或某个操作的执行时间。

注：GetTickCount 的值会在系统启动后约49.7天（(2^32-1) ms）后回绕到0，这是因为其返回值是一个32位无符号整数，可以使用 GetTickCount64 代替，需添加 windows.h 头文件。

消息处理

https://docs.easyx.cn/zh-cn/msg-func

消息缓冲区可以缓冲 63 个未处理的消息。每次获取消息时，将从消息缓冲区取出一个最早发生的消息。

函数用法	函数说明
ExMessage **getmessage** (BYTE filter = -1) void **getmessage** (ExMessage *msg, BYTE filter = -1)	从消息缓冲区获取一个消息。如果缓冲区中没有消息，则程序会一直等待（阻塞式）
bool **peekmessage** (ExMessage *msg, BYTE filter = -1, bool removemsg = true)	从消息缓冲区获取一个消息，并立即返回
void **flushmessage** (BYTE filter = -1)	清空消息缓冲区

参数说明：

- **msg**：指向消息结构体 ExMessage 的指针，用来保存获取到的消息。
- **filter**：指定要获取的消息范围，默认 -1 获取所有类别的消息。可以用以下值或值的组合获取指定类别的消息

标志	描述
EX_MOUSE	鼠标消息。
EX_KEY	按键消息。
EX_CHAR	字符消息。
EX_WINDOW	窗口消息。

- **removemsg**：在 peekmessage 处理完消息后，是否将其从消息队列中移除。

ExMessage 结构体

https://docs.easyx.cn/zh-cn/exmessage

```
struct ExMessage
{
    USHORT message;                 // 消息标识
    union
    {
        // 鼠标消息的数据
        struct
        {
            bool ctrl       :1;     // Ctrl 键是否按下
            bool shift      :1;     // Shift 键是否按下
            bool lbutton    :1;     // 鼠标左键是否按下
            bool mbutton    :1;     // 鼠标中键是否按下
            bool rbutton    :1;     // 鼠标右键
            short x;                // 鼠标的 x 坐标
            short y;                // 鼠标的 y 坐标
            short wheel;            // 鼠标滚轮滚动值，为 120 的倍数
        };

        // 按键消息的数据
        struct
        {
            BYTE vkcode;            // 按键的虚拟键码
            BYTE scancode;          // 按键的扫描码（依赖于 OEM）
            bool extended   :1;     // 按键是否是扩展键
            bool prevdown   :1;     // 按键的前一个状态是否按下
        };

        // 字符消息的数据
        TCHAR ch;

        // 窗口消息的数据
        struct
        {
            WPARAM wParam;
            LPARAM lParam;
        };
    };
```

```
};
```

message：可以分为四大类：**EX_MOUSE**（鼠标11项）、**EX_KEY**（键盘2项）、**EX_CHAR**（字符1项）、**EX_WINDOW**（窗口3项）

union：共用体中存储具体消息的数据

鼠标消息

范例：跟随鼠标移动的圆

```
#include <graphics.h>

int main() {
    bool running = true;                // 主循环控制参数
    ExMessage msg;                      // 消息对象
    int x = 400;                        // 圆心X坐标
    int y = 300;                        // 圆心Y坐标
    int r = 50;                         // 圆半径
    initgraph(800, 600);                // 初始化绘图窗口
    BeginBatchDraw();                   // 开启批量绘图

    // 主循环
    while (running) {
        // 消息处理
        while (peekmessage(&msg)) {
            if (msg.message == WM_MOUSEMOVE) {              // 圆的位置随鼠标位置变
化
                x = msg.x;
                y = msg.y;
            }
            else if (msg.message == WM_LBUTTONDOWN) {       // 左键按下圆变红色
                setfillcolor(RED);
            }
            else if (msg.message == WM_LBUTTONUP) {         // 左键松开圆变白色
                setfillcolor(WHITE);
            }
            else if (msg.message == WM_RBUTTONDOWN) {       // 右键按下结束主循环
                running = false;
            }
        }
        // 绘图
        cleardevice();                  // 清除屏幕
        solidcircle(x, y, r);           // 绘制当前帧内容
        FlushBatchDraw();               // 刷新批量绘图
    }

    EndBatchDraw();                     // 关闭批量绘图
    closegraph();                       // 关闭绘图窗口
    return 0;
}
```

键盘消息

范例1：用键盘控制小球

```
#include <graphics.h>

// 用结构体封装小球属性
typedef struct Ball
{
    int x;              // 小球圆心坐标x
    int y;              // 小球圆心坐标y
    int r;              // 小球半径
    int dx;             // 小球在x轴方向移动的增量
    int dy;             // 小球在y轴方向移动的增量
    COLORREF color;     // 小球颜色
} Ball;

int main()
{
    bool running = true;
    ExMessage msg;
    Ball ball = { 300, 300, 20, 5, 5, YELLOW }; // 创建小球并初始化
    initgraph(600, 600);
    BeginBatchDraw();

    while (running)
    {
        while (peekmessage(&msg))
        {
            if (msg.message == WM_KEYDOWN)
            {
                switch (msg.vkcode)                 // 判断虚拟键代码
                {
                case 'w':                           // 上键：小球Y坐标减少
                case 'W':
                case VK_UP:
                    ball.y -= ball.dy;
                    break;

                case 's':
                case 'S':
                case VK_DOWN:                       // 下键：小球Y坐标增加
                    ball.y += ball.dy;
                    break;

                case 'a':
                case 'A':
                case VK_LEFT:                       // 右键：小球X坐标减少
                    ball.x -= ball.dx;
                    break;

                case 'd':
                case 'D':
                case VK_RIGHT:                      // 右键：小球X坐标增加
                    ball.x += ball.dx;
```

```
                    break;

                case VK_ESCAPE:                         // ESC键：结束主循环
                    running = false;
                    break;
                }
            }
        }

        cleardevice();                              // 清除屏幕
        setfillcolor(ball.color);                   // 设置填充颜色
        solidcircle(ball.x, ball.y, ball.r);        // 绘制无边框填充圆;
        FlushBatchDraw();
    }

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

虚拟键代码 https://learn.microsoft.com/zh-cn/windows/win32/inputdev/virtual-key-codes

案例缺陷：

- CPU占用率高
- 小球不能斜向移动
- 小球越界

范例2：用键盘控制小球（支持斜向运动）

```
#include <graphics.h>

// 用结构体封装小球属性
typedef struct Ball
{
    int x;                          // 小球圆心坐标x
    int y;                          // 小球圆心坐标y
    int r;                          // 小球半径
    int dx;                         // 小球在x轴方向移动的增量
    int dy;                         // 小球在y轴方向移动的增量
    COLORREF color;                 // 小球颜色
    bool isMoveUp = false;          // 小球是否向四个方向移动
    bool isMoveDown = false;
    bool isMoveLeft = false;
    bool isMoveRight = false;
} Ball;

int main()
{
    bool running = true;
    ExMessage msg;
    Ball ball = { 300, 300, 20, 5, 5, YELLOW };     // 创建小球并初始化
```

```
    initgraph(600, 600);
    BeginBatchDraw();

    while (running)
    {
        DWORD beginTime = GetTickCount();           // 记录循环开始时间

        // 消息处理
        while (peekmessage(&msg))
        {
            if (msg.message == WM_KEYDOWN)          // 按下按键处理
            {
                switch (msg.vkcode)
                {
                case 'w':
                case 'W':
                case VK_UP:
                    ball.isMoveUp = true;
                    break;

                case 's':
                case 'S':
                case VK_DOWN:
                    ball.isMoveDown = true;
                    break;

                case 'a':
                case 'A':
                case VK_LEFT:
                    ball.isMoveLeft = true;
                    break;

                case 'd':
                case 'D':
                case VK_RIGHT:
                    ball.isMoveRight = true;
                    break;

                case VK_ESCAPE:
                    running = false;
                    break;
                }
            }
            if (msg.message == WM_KEYUP)            // 松开按键处理
            {
                switch (msg.vkcode)
                {
                case 'w':
                case 'W':
                case VK_UP:
                    ball.isMoveUp = false;
                    break;

                case 's':
                case 'S':
```

```
                case VK_DOWN:
                    ball.isMoveDown = false;
                    break;

                case 'a':
                case 'A':
                case VK_LEFT:
                    ball.isMoveLeft = false;
                    break;

                case 'd':
                case 'D':
                case VK_RIGHT:
                    ball.isMoveRight = false;
                    break;
                }
            }
        }

        // 数据处理：根据小球的移动状态设置其坐标
        if (ball.isMoveUp)
            ball.y -= ball.dy;
        if (ball.isMoveDown)
            ball.y += ball.dy;
        if (ball.isMoveLeft)
            ball.x -= ball.dx;
        if (ball.isMoveRight)
            ball.x += ball.dx;

        // 绘图
        cleardevice();                                  // 清除屏幕
        setfillcolor(ball.color);                       // 设置填充颜色
        solidcircle(ball.x, ball.y, ball.r);            // 绘制无边框填充圆;
        FlushBatchDraw();

        // 帧延时
        DWORD endTime = GetTickCount();                 // 记录循环结束时间
        DWORD elapsedTime = endTime - beginTime;        // 计算循环耗时
        if (elapsedTime < 1000 / 60)                    // 按每秒60帧进行补时
            Sleep(1000 / 60 - elapsedTime);
    }

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

范例3：改进斜向运动问题

```
#include <graphics.h>
#include <math.h>

// 用结构体封装小球属性
```

```
typedef struct Ball
{
    int x;                      // 小球圆心坐标x
    int y;                      // 小球圆心坐标y
    int r;                      // 小球半径
    int dx;                     // 小球在x轴方向移动的增量
    int dy;                     // 小球在y轴方向移动的增量
    COLORREF color;             // 小球颜色
    bool isMoveUp = false;      // 小球是否向四个方向移动
    bool isMoveDown = false;
    bool isMoveLeft = false;
    bool isMoveRight = false;
} Ball;

int main()
{
    bool running = true;
    ExMessage msg;
    Ball ball = { 300, 300, 20, 5, 5, YELLOW };     // 创建小球并初始化
    initgraph(600, 600);
    BeginBatchDraw();

    while (running)
    {
        DWORD beginTime = GetTickCount();           // 记录循环开始时间

        // 消息处理
        while (peekmessage(&msg))
        {
            if (msg.message == WM_KEYDOWN)          // 按下按键处理
            {
                switch (msg.vkcode)
                {
                case 'w':
                case 'W':
                case VK_UP:
                    ball.isMoveUp = true;
                    break;

                case 's':
                case 'S':
                case VK_DOWN:
                    ball.isMoveDown = true;
                    break;

                case 'a':
                case 'A':
                case VK_LEFT:
                    ball.isMoveLeft = true;
                    break;

                case 'd':
                case 'D':
                case VK_RIGHT:
                    ball.isMoveRight = true;
```

```
                break;

            case VK_ESCAPE:
                running = false;
                break;
            }
        }
        if (msg.message == WM_KEYUP)            // 松开按键处理
        {
            switch (msg.vkcode)
            {
            case 'w':
            case 'W':
            case VK_UP:
                ball.isMoveUp = false;
                break;

            case 's':
            case 'S':
            case VK_DOWN:
                ball.isMoveDown = false;
                break;

            case 'a':
            case 'A':
            case VK_LEFT:
                ball.isMoveLeft = false;
                break;

            case 'd':
            case 'D':
            case VK_RIGHT:
                ball.isMoveRight = false;
                break;
            }
        }
    }

    // 数据处理：根据小球的移动状态设置其坐标
    //if (ball.isMoveUp)
    //  ball.y -= ball.dy;
    //if (ball.isMoveDown)
    //  ball.y += ball.dy;
    //if (ball.isMoveLeft)
    //  ball.x -= ball.dx;
    //if (ball.isMoveRight)
    //  ball.x += ball.dx;

    // 斜向移动：计算不同方向(包括同时)按下时的速度增量
    int directX = ball.isMoveRight - ball.isMoveLeft;
    int directY = ball.isMoveDown - ball.isMoveUp;
    double directXY = sqrt(directX * directX + directY * directY);
    if (directXY != 0)
    {
        double factorX = directX / directXY;    //计算X、Y方向的标准化分量
```

```
            double factorY = directY / directXY;
            ball.x += (int)ball.dx * factorX;       //小球坐标 = 方向增速 * 方向的标
准化分量
            ball.y += (int)ball.dy * factorY;
        }

        // 绘图
        cleardevice();                                  // 清除屏幕
        setfillcolor(ball.color);                       // 设置填充颜色
        solidcircle(ball.x, ball.y, ball.r);            // 绘制无边框填充圆;
        FlushBatchDraw();

        // 帧延时
        DWORD endTime = GetTickCount();                 // 记录循环结束时间
        DWORD elapsedTime = endTime - beginTime;        // 计算循环耗时
        if (elapsedTime < 1000 / 60)                    // 按每秒60帧进行补时
            Sleep(1000 / 60 - elapsedTime);
    }

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

范例4：窗口边缘碰撞检测

```
#include <graphics.h>
#include <math.h>
#define WIN_WIDTH 600
#define WIN_HEIGHT 600

// 用结构体封装小球属性
typedef struct Ball
{
    int x;                      // 小球圆心坐标x
    int y;                      // 小球圆心坐标y
    int r;                      // 小球半径
    int dx;                     // 小球在x轴方向移动的增量
    int dy;                     // 小球在y轴方向移动的增量
    COLORREF color;             // 小球颜色
    bool isMoveUp = false;      // 小球是否向四个方向移动
    bool isMoveDown = false;
    bool isMoveLeft = false;
    bool isMoveRight = false;
} Ball;

int main()
{
    bool running = true;
    ExMessage msg;
    Ball ball = { 300, 300, 20, 5, 5, YELLOW };     // 创建小球并初始化
    initgraph(WIN_WIDTH, WIN_HEIGHT);
    BeginBatchDraw();
```

```
while (running)
{
    DWORD beginTime = GetTickCount();           // 记录循环开始时间

    // 消息处理
    while (peekmessage(&msg))
    {
        if (msg.message == WM_KEYDOWN)          // 按下按键处理
        {
            switch (msg.vkcode)
            {
            case 'w':
            case 'W':
            case VK_UP:
                ball.isMoveUp = true;
                break;

            case 's':
            case 'S':
            case VK_DOWN:
                ball.isMoveDown = true;
                break;

            case 'a':
            case 'A':
            case VK_LEFT:
                ball.isMoveLeft = true;
                break;

            case 'd':
            case 'D':
            case VK_RIGHT:
                ball.isMoveRight = true;
                break;

            case VK_ESCAPE:
                running = false;
                break;
            }
        }
        if (msg.message == WM_KEYUP)            // 松开按键处理
        {
            switch (msg.vkcode)
            {
            case 'w':
            case 'W':
            case VK_UP:
                ball.isMoveUp = false;
                break;

            case 's':
            case 'S':
            case VK_DOWN:
                ball.isMoveDown = false;
```

```
                    break;

                case 'a':
                case 'A':
                case VK_LEFT:
                    ball.isMoveLeft = false;
                    break;

                case 'd':
                case 'D':
                case VK_RIGHT:
                    ball.isMoveRight = false;
                    break;
                }
            }
        }

        // 斜向移动：计算不同方向(包括同时)按下时的速度增量
        int directX = ball.isMoveRight - ball.isMoveLeft;
        int directY = ball.isMoveDown - ball.isMoveUp;
        double directXY = sqrt(directX * directX + directY * directY);
        if (directXY != 0)
        {
            double factorX = directX / directXY;    //计算X、Y方向的标准化分量
            double factorY = directY / directXY;
            ball.x += (int)ball.dx * factorX;       //小球坐标 = 方向增速 * 方向的标
准化分量
            ball.y += (int)ball.dy * factorY;
        }

        // 边缘检测
        if (ball.y - ball.r <= 0)                   // 上
            ball.y = ball.r;
        if (ball.y + ball.r >= WIN_HEIGHT)          // 下
            ball.y = WIN_HEIGHT - ball.r - 1;
        if (ball.x - ball.r <= 0)                   // 左
            ball.x = ball.r;
        if (ball.x + ball.r >= WIN_WIDTH)           // 右
            ball.x = WIN_WIDTH - ball.r - 1;

        // 绘图
        cleardevice();                              // 清除屏幕
        setfillcolor(ball.color);                   // 设置填充颜色
        solidcircle(ball.x, ball.y, ball.r);        // 绘制无边框填充圆;
        FlushBatchDraw();

        // 帧延时
        DWORD endTime = GetTickCount();             // 记录循环结束时间
        DWORD elapsedTime = endTime - beginTime;    // 计算循环耗时
        if (elapsedTime < 1000 / 60)                // 按每秒60帧进行补时
            Sleep(1000 / 60 - elapsedTime);
    }

    EndBatchDraw();
    closegraph();
```

```
    return 0;
}
```

其它函数

设置窗口标题

范例：使用 **GetHWnd** 和 **SetWindowText** 函数设置窗口标题

```
#include <graphics.h>

int main()
{
    initgraph(600, 600);

    HWND hWnd = GetHWnd();                          // 获得窗口句柄
    SetWindowText(hWnd, _T("植物大战僵尸"));       // 使用 Windows API 修改窗口名称

    system("pause");
    closegraph();
    return 0;
}
```

弹窗消息

在Visual C++（VC）中，MessageBox 函数是一个常用的 Windows API 函数，用于显示一个模态对话框，其中包含文本、标题、图标和按钮等。以下是函数的详细用法：

```
int MessageBox(
  HWND   hWnd,            // 父窗口句柄。如果为NULL，则消息框没有父窗口
  LPCTSTR lpText,          // 要显示的消息文本
  LPCTSTR lpCaption,       // 消息框的标题
  UINT    uType            // 指定消息框的内容和行为的标志
);
```

参数说明

1. **hWnd**：指定消息框的父窗口句柄。如果此参数为NULL，则消息框没有父窗口，且作为顶级窗口显示。
2. **lpText**：要在消息框中显示的文本。
3. **lpCaption**：消息框的标题。如果此参数为NULL，则默认标题为“Error”。
4. **uType**：用于指定消息框的内容和行为的标志。这可以是一个或多个以下常量的组合：
    - MB_OK：消息框包含一个“确定”按钮。
    - MB_OKCANCEL：消息框包含“确定”和“取消”按钮。
    - MB_YESNO：消息框包含“是”和“否”按钮。
    - MB_YESNOCANCEL：消息框包含“是”、“否”和“取消”按钮。

- MB_ICONEXCLAMATION、MB_ICONWARNING、MB_ICONINFORMATION、MB_ICONQUESTION、MB_ICONERROR等：用于指定消息框中显示的图标。

返回值

函数返回一个整数值，表示用户点击的按钮。例如：

- **IDOK**：用户点击了“确定”按钮。
- **IDCANCEL**：用户点击了“取消”按钮。
- **IDYES**：用户点击了“是”按钮。
- **IDNO**：用户点击了“否”按钮。

```
#include <graphics.h>

int main()
{
    initgraph(1000, 600);

    HWND hWnd = GetHWnd();
    MessageBox(hWnd, _T("你被僵尸吃掉了！"), _T("游戏结束"), MB_OK | MB_ICONERROR);

    closegraph();
    return 0;
}
```

注：在使用 MessageBox 函数之前，需要包含 **windows.h** 头文件（如果已经包含了 graphics.h 头文件则可以省略）

播放音频

mciSendString 是 Windows API 中的一个函数，用于向媒体控制接口（Media Control Interface，MCI）设备发送命令字符串。这个函数常用于控制多媒体设备，如音频和视频播放，支持 MPEG, AVI, WAV, MP3 等多种格式。

范例：播放背景音乐

```
#include <graphics.h>
//#include <windows.h>               // 此项在导入graphics.h头文件后可以省略
#pragma comment(lib, "winmm.lib")    // 加载多媒体静态库

int main()
{
    initgraph(1000, 600);

    mciSendString(_T("open audio\\bg.mp3 alias BGM"), 0, 0, 0);          // 打开音乐文件，alias指定别名
    mciSendString(_T("play BGM repeat"), 0, 0, 0);                       // 使用别名播放音乐，repeat重复播放

    IMAGE img;
    loadimage(&img, _T("image\\background.jpg"));
    putimage(0, 0, &img);
```

```
    system("pause");
    closegraph();
    return 0;
}
```

随堂练习

练习1：鼠标+键盘绘图

功能要求：

1. 鼠标按下左键画 10*10 的正方形、按下右键画半径为 10 的圆（鼠标点击的位置为图形中心）
2. 同时按下 Ctrl 键和鼠标左、右键，分别画 20*20 的正方形和半径为 20 的圆
3. 按下键盘 C 键清屏、R/G/B/W 键分别修改画图颜色为红、绿、蓝、白，ESC 键退出

练习2：弹球

功能要求：

1. 一个自由运动的小球，碰到左、右、上边界会反弹，碰到下边界结束程序
2. 一个键盘控制的挡板，在屏幕下方只能左右移动，不能超出屏幕边界，小球碰到挡板后会反弹