Windows 编程中的图形输出

胡春安, 欧阳城添
(江西理工大学 信息工程学院, 江西 赣州 341000)

摘要: Windows 程序的所有输出其实都是图形. 初学者在学 Windows 编程时, 对图形输出常感到困惑, 不知如何下手. 在 Windows 编程中要想使用 GDI 对图形进行输出, 首先就必须理解设备上下文、当前作图工具(如什么样的线、什么样的刷子和什么样的字体等)以及如何把 GDI 逻辑坐标映射到设备坐标等概念. 通过围绕上述几个概念对 Windows 编程中的图形输出进行了方法介绍, 并举例给出了绘制图形的过程, 最后对利用 MFC 进行图形输出也作了介绍.

关键词: GDI; DC; 绘图; 函数; MFC
中图分类号: TP316.86 文献标识码: A 文章编号: 1006-4303(2006)05-0546-04

Graphics of Windows program output

HU Chun-an, OUYANG Cheng-tian
(College of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China)

Abstract: All outputs of Windows program are graphics. Most of beginners often feel puzzled and don't know how to design a Windows program. Before using the GDI in Windows program, the properties of the device, painting tools (for example what is the line, what is the brush, what is the font), and how to map the GDI logical coordinates to the coordinates of the device have to be comprehended. In this paper, the concepts about GDI will be introduced. The case of graphics output in Windows program is explained. Finally how to use MFC for graphics output is presented.

Key words: GDI; DC; plot; function; MFC

0 引 言

用 Visual C++ 编写 Windows 程序时, 初学者对图形输出常感到困惑, 不知如何在窗口客户区域中绘制图形, 其原因是对绘图机制不甚了解. 在 Windows 编程中要想使用 GDI(Graphics Device Interface)对图形进行输出, 首先就必须理解设备上下文、当前作图工具(如什么样的线、什么样的刷子、什么样的字体等)以及如何把 GDI 逻辑坐标映射到设备坐标等概念. 笔者力图对 Windows 编程中的图形输出所需的概念进行详细阐述并对设备上下文的获取进行多种方法介绍. 只要对绘图过程有了正确的理解, 在作图时就不会感到困难.

1 图形设备接口 GDI

在 Windows 系统中包含一个图形设备接口, 简称 GDI, 它管理 Windows 环境下的所有图形输出, 主要目标之一是支持在输出设备(如显示器、打印机等)上的与设备无关的图形[1]. GDI 通过将应用程序与不同输出设备特性相隔离, 使 Windows 应用程序能够毫无问题地在 Windows 支持的任何图形输出设备上运行, 它是 Windows 操作系统的重要组成部

收稿日期: 2006-01-03
作者简介: 胡春安(1966—), 女, 江西抚州人, 高级实验师, 主要从事计算机基础教学及软件开发研究.
© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

分. 通过 GDI, 可以控制并管理实现绘图所需要的三大要素:

(1) 画布: 指图形要绘制在什么地方. GDI 提供的画布可以是屏幕上的窗口, 打印机甚至是内存.

(2) 绘图工具: 指使用什么东西进行绘制. GDI 提供了多种绘图工具, 每种绘图工具可以设定多种属性, 如画笔、画刷和字体等都是 GDI 提供的绘图工具.

(3) 绘图动作: 指怎样绘制. GDI 提供了多种绘图函数, 如直线、矩形、椭圆及文本输出等绘图函数.

2　设备上下文 DC

在 Windows 系统中, GDI 将画布和绘图工具合在一起称为"设备上下文", 简称 DC(Device Context).

由于 Windows 使用 GDI 来作为图形输出的手段, 因此 Windows 应用程序的任何图形输出都必须首先获得一个设备上下文句柄 HDC 来使用相应的输出设备. DC 实际上是由 GDI 维护着的一种数据结构, 每个 DC 都与某种显示设备(如打印机或显示器)相联系. 之所以将这种数据结构称为"设备上下文", 就是因为这种数据结构中存储着与显示或打印有关的属性, 包括显示文本时所使用的前景和背景颜色、显示模式(是覆盖还是透明)、当前的绘图工具(如什么样的线、什么样的刷子、什么样的字体等)、剪切区域以及如何把 GDI 逻辑坐标映射到设备坐标的信息等等. 有了 DC, 程序就可以在该 DC 所表示的显示区域中绘制图形.

3　图形输出的过程

3.1　获取 HDC(设备上下文句柄)

Windows 提供了 5 种不同的获取 HDC 的方法[2]:

方法 **1**　在处理 WM_PAINT 消息时调用 BeginPaint() 函数, 该函数准备绘制指定的窗口, 并将有关绘制窗口的信息填入 Paintstruct 结构. 那么什么时候处理 WM_PAINT 消息呢? 当使用者改变窗口大小或是将窗口图标化之后再恢复原状, 或是来自程序的刻意要求时表示画面需要重新绘制.

hdc= BeginPaint(hwnd, & Paintstruct)

方法 **2**　与方法 1 相对应, 在处理非 WM_PAINT 消息时调用 GetDC() 函数, 获取某个窗口客户区域的 HDC.

hdc= GetDC(hwnd)

方法 **3**　利用 GetWindowDC() 函数获取整个窗口区域的设备上下文句柄. GetWindowDC() 函数获取的设备上下文, 原点是窗口的左上角, 而不是客户区域的左上角.

hdc= GetWindowDC (hwnd)

方法 **4**　利用 CreateDC() 函数来生成某个显示设备(如显示器或打印机)的设备上下文句柄.

比如, 下面语句生成整个显示器的设备上下文:

hdc= CreateDC("DISPLAY", NULL, NULL, NULL)

下面的语句生成打印机的设备上下文:

prn_hdc= CreateDC("EPSION LQ_1500", "EPSION24", "LPT1", NULL)

CreateDC() 函数的前三个参数及顺序必须与 WIN.INI 文件打印设置的 Device= 项相同.

方法 **5**　利用函数 CreateCompatibleDC() 在内存中创建与某设备兼容的设备上下文, 一般用于位图(Bitmap)操作.

Mem_hdc= CreateCompatibleDC(hdc)

CreateCompatibleDC() 函数生成一个与 hdc 参数所指定的设备相兼容的设备上下文句柄. 目前, 本函数只能为支持光栅操作的显示设备生成兼容的设备上下文.

用 CreateCompatibleDC 函数所生成的 HDC 有个特征: 它的初始大小为 0. 只有把某一个位图选入该 DC 后, 它才有了一个实际的大小, 即被选入位图的大小.

这 5 种获取 HDC 的方法中, 除了第一种方法是用于被动绘图外, 其他方法都用于主动绘图.

3.2　设置 HDC 属性

应用程序获取的任何 DC 都有其默认的属性设置. 在实际绘图之前应用程序可以改变其中的一些属性设置, 尤其是要重新设置一些绘图工具, 比如: 画笔(pen, 负责绘制线、矩形和椭圆的边框)、刷子(brush, 用来填充矩形和椭圆的内部)、字体(font)以及位图和逻辑调色板等等.

设置新 DC 属性时用得最频繁的函数就是 SelectObject() 函数. 该函数用来将新的绘图工具选入 DC, 返回值是 DC 中被替换掉的旧 GDI 对象的句柄. 该句柄一定要保存到局部变量中以便能在第 4 步恢复原来的属性设置, 比如: hOldPen= SelectObject(hdc, hNewPen).

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

获取新的 GDI 对象有两种方法: 一种是利用 Create…类函数来创建自己的 GDI 对象; 另一种是利用 GetStockObject( ) 函数获取系统预定义的 GDI 对象. 比如, 如果想创建一个红色实心宽度为 1 的画笔, 可以这样调用 CreatePen( ) 函数:

hNewPen= CreatePen (PS_SOLID, 1, RGB (255, 0, 0))

如果想获取系统预定义的空刷子, 可以这样调用 GetStockObject( ) 函数:

hNullBrush = GetStockObject (NULL_BRUSH)

3.3 调用 GDI API 函数进行绘图操作

在正确设置完 DC 的属性后, 就可以调用 GDI 绘图函数来绘图了. 比如, 要画线就可以调用 MoveToEx( ) 函数来移动画笔的当前位置, 再调用 LineTo( ) 函数进行画线; 要画一个矩形, 只需调用 Rectangle( ) 函数; 画椭圆要调用 Ellipse( ) 函数; 输出文本可以调用 TextOut( ) 函数.

3.4 恢复 DC 属性

这一步的主要操作是选出步骤 2 选入 DC 的 GDI 对象, 恢复 DC 中 GDI 对象的原来设置, 这同样需要调用 SelectObject( ) 之类的函数. 比如, 要恢复原先选入的画笔, 可以调用:

SelectObject(hdc, holdPen)

在释放 HDC 之前, 一般要恢复 DC 原来 GDI 对象的设置. 如果不恢复原来 GDI 对象的设置(尤其是在内存中创建的兼容 DC) , 释放 HDC 的操作会失败, 这样会造成内存遗漏.

注意, 在恢复完 DC 中的 GDI 对象设置后, 就可以安全删除所创建的 GDI 对象了. 因为 GDI 对象占用系统的资源空间, 所以在使用完 GDI 对象后一定要删除它们. 删除 GDI 对象所需要的函数主要是 DeleteObject( ) .

3.5 释放 HDC

HDC 也占用系统的资源空间, 所以在使用完 HDC 之后必须释放它们. 在一次绘图操作过程中, 获取和释放 HDC 必须成对出现. 获取 HDC 的方法不同, 释放 HDC 的方法也不同. EndPaint(hwnd, & Paintstruct) 与 BeginPaint(hwnd, & Paintstruct) 相对应, 这两个函数必须成对出现. 如果你用 GetDC( ) 和 GetWindowDC 函数获取 HDC, 你必须使用 ReleaseDC(hwnd, hdc) 函数来释放 HDC. 如果你使用 CreateDC( ) 和 CreateCompatibleDC( ) 函数创建 HDC, 你必须调用 DeleteDC(hdc) 来释放 HDC.

3.6 代码实例

```
hdc= GetDC( hwnd) ; //获取 HDC
hNewPen= CreatePen( PS_SOLID, 1, RGB(0, 255, 0) ) ; //创建新的 GDI 画笔对象
hOldPen= SelectObject( hdc, hNewPen) ; //将新的画笔选入 DC
MoveToEx( hdc, MousePosX-20, MousePosY, NULL) ; //将画笔位置移动到线的左端
LineTo( hdc, MousePosX+ 20, MousePosY) ; //画线
SelectObject( hdc, hOldPen) ; //将原来画笔选回 hdc
DeleteObject( hNewPen) ; //删除创建的画笔
ReleaseDC( hwnd, hdc) ; //释放 HDC
```

4 映射方式

真正影响在客户区域上绘制图形的设备上下文属性是“映射方式”, 与映射方式属性密切相关的 4 个设备上下文属性: 窗口原点、视窗原点、窗口范围和视窗范围. 实际上应用程序在调用 GDI 绘图函数作图时, 除了需要 HDC 外, 还需要位置及大小信息. 而这些位置及大小信息都采用“逻辑单位”, 包括毫米、英寸等, 默认情况下是像素, 而 Windows 在实现这些绘图操作时必须使用显示设备的设备单位, 即像素. 因此, GDI 函数使用的逻辑单位在显示输出时必须转换成设备单位. Windows 定义了 8 种映射方式[3].

可以调用函数 SetMapMode(hdc, nMapMode) 来设置这 8 种映射方式中的一种, 调用 GetMapMode(hdc) 获取设备上下文的映射方式. hdc 用来标识设备上下文, nMapMode 可以取 MM_TEXT, MM_LOMETRIC, MM_HIMETRIC 等 8 个值中的一个. 在设置了映射方式之后, 到下一次设置映射方式之前, Windows 一直使用这种映射方式. 如果想要获取当前的映射方式, 可用 nMapMode= GetMapMode(hdc) , 在设置了映射方式之后, 就规定了逻辑单位的大小和增量的方式, 在 GDI 画图函数中, 可以不必考虑这些内容而直接使用逻辑数字, 例如:

```
SetMapMode( hdc, MM_TEXT)
TextOut( hdc, 20, 26, szBuffer, nLength)
```

即正文从客户区域左起第 20 个像素, 顶边起第 26 个像素的位置开始写输出.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

5　MFC 程序中的图形输出

以上介绍的是 Windows API 编程中图形输出所需掌握的技能, 我们再来看看利用 MFC 编程时图形输出的方法. MFC 即 Microsoft Fundation Classsesr 的缩写, 就是利用微软提供的基础类库进行编程. 当然其图形输出概要及过程与上述大至相同, 只是对一些对象进行封装, 使得应用更加快捷方便.

MFC 对 Windows 的 GDI 进行了封装, 将 GDI 中的绘图工具封装到 CGdiObject 类中, 称为 GDI 对象类. 将前面提到的画布、绘图动作和 GDI 对象封装到 CDC(Class Device Context) 类中, 称为设备环境类.

5.1　在 MFC 程序中的绘图方法

(1) 根据绘图目标(如窗口) 取得 CDC 对象.

(2) 选择绘图使用的 CGdiObject 对象.

(3) 将 CGdiObject 对象附加到 CDC 对象.

(4) 使用该 CDC 对象的绘图成员函数在目标(如窗口) 上绘制图形.

5.2　设备环境类

(1) CPaintDC

该类以窗口内客户区为目标, 可以在窗口内绘制图形. 该类的对象在构造函数中自动调用 BeginPaint 取得目标设备环境, 在析构函数中自动调用 EndPaint 释放设备环境. CPaintDC 适用于在接到 WM_PAINT 消息时进行绘图操作.

(2) CClientDC

该类以窗口内客户区为目标, 可以在窗口内绘制图形, 该类的对象在构造函数中自动调用 GetDC()函数取得目标设备环境, 在析构函数中自动调用 ReleaseDC()函数释放设备环境. CClientDC 适用于处理非 WM_PAINT 消息时进行绘图操作, 在窗口客户区画一条绿线代码如下:

```
void CToolbarView::OnLine()
{
    CClientDC dc(this);
    CPen * pOldpen;//指向原来笔的指针
    CPen Newpen(PS_SOLID, 1, RGB(0, 255, 0));//创建一支宽度为 1 的绿色实心笔
    pOldpen= dc.SelectObject(&Newpen);//选用新笔, 同时保留原来的笔
    dc.MoveTo(50, 50);
    dc.LineTo(150, 50);
    dc.SelectObject(pOldpen);//选回原来的笔
    DeleteObject(Newpen);//删除创建的画笔
}
```

6　结　论

Windows 下的应用程序使用图形设备接口(GDI) 来进行图形输出, GDI 屏蔽了不同设备的差异, 提供了设备无关的图形输出能力, Windows 应用程序只要发出设备无关的 GDI 请求, 由 GDI 去完成实际的图形输出操作. 因此, Windows 程序员必须对图形设备接口(GDI) 的概念要非常熟悉, 还要熟悉 Windows 中提供丰富的 GDI 图形函数用于图形输出, 这对输出图形是相当有用的.

参考文献:

[1] 张志强. Windows 编程技术[M]. 北京: 机械工业出版社, 2003.

[2] 杨开城. Visual C++ 应用程序开发教程[M]. 北京: 高等教育出版社, 2001.

[3] 侯俊杰. 深入浅出 MFC[M]. 北京: 华中科技大学出版社, 2001.

(责任编辑: 刘　岩)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net