VC 在监测系统中图形刷新问题的研究

陈喜阳 张克危 彭玉成 华中科技大学能源与动力学院(430074)

Abstract

Adopt memory virtual screen to improve on the screen display.By Creating the memory device context and display compatible bitmap,draw the graphics in the bitmap field that is compatible with the memory device context,then copy the graphics from the bitmap to the display screen.It cut down the time that used to display the graphic and needn't erase the background.Under the line of it,people can realize the non-flicker and continous real curve display.

Keywords:refresh,flicker,memory virtual screen

摘 要

采用内存虚拟屏幕改进屏幕显示,通过创建内存设备环境和显示兼容的位图,将图形绘制在与内存设备环境兼容的位图区域中,然后将图形从位图中复制到显示器屏幕上,缩短了显示图形所用的时间,并且不必擦除背景,实现了无闪烁的实时曲线连续显示。

关键词:刷新,闪烁,内存虚拟屏幕

在工业在线监测系统中,图形动态绘制和显示是必不可少的,这样操作人员可直接掌握现场中整个系统的运行情况,还可根据历史数据和实时数据形成各种趋势图,预测系统将来的运行状态。工业监测需要采集的数据量比较大,采集的频率比较高,这样在实时监测曲线的绘制过程中涉及的数据点比较多,图形刷新的频率也比较高,采用常用直接更新屏幕的方法重新绘制图形容易出现闪烁问题。因为在连续动态变化过程中,当屏幕由一个画面变化为另外一个画面时,首先将原来的背景擦除,接着将新的图形绘制在绘图区域,由于显示背景在不断擦除及绘制图形所用的时间较长,导致屏幕出现闪烁和不连续状态。而显示图形的闪烁问题容易给操作人员带来读数的不准确和判断失误,造成工业事故,因此研究图形刷新带来屏幕闪烁问题对工业监测系统的研发有着一定的必要性。

1 MFC 绘图原理[1]

Windows 中负责图形输出的是 GDI(图形设备接口),这是 Windows 与硬件无关的图形输出模式的体现。GDI 建立在硬件抽象层(HAL)之上,屏蔽了不同输出设备之间的差异,从而为用户提供了一个统一的“标准输出设备”。但是,与 DOS 不同,Windows 是多任务、进程独立的,每一个窗口都应该有一个独立的输出通道。这样,GDI 又使用了一种简单的机制来保证在窗口中画图的不同程序之间能共享“设备”而又互不干扰。这个机制就是 DC(设备上下文)。因此 Windows MFC 提供了四种不同的 DC 环境:

1)CPaintDC,用于在窗口客户区画图(在 OnPaint 处理函数中使用);

2)CClientDC,也用于在窗口客户区画图(在 OnPaint 处理函数之外使用);

3)CWindowDC,用于在窗口内任意地方画图,包括非客户区;

4)CMetaFileDC,用于绘制 GDI 图元文件。

Windows GDI 对象类型通过 MFC 库类来表示,CGdiObject 是 GDI 对象类的一个抽象的基类,Windows GDI 对象由 CGdiObject 派生类的 C++对象表示,主要的 Windows GDI 对象有下面几种:CBitmap(位图)、CBrush(画刷)、CFont(字体)、CPalette(调色板)、CPen(画笔)和 CRgn(区域)。

MFC 绘图就是通过首先获取一个 DC,创建相应的 Windows GDI 对象,在屏幕对应区域绘图。当然在这个绘制图形的过程中需要特别注意下面几个技术细节:

1)绘图模式:GDI 将像素点输出到逻辑显示平面上时,并不只是简单的输出像素颜色,而是通过一系列布尔运算将输出像素点的颜色和输出目标位置上像素点的颜色进行合成再显示。所使用的逻辑运算关系就由 DC 的绘图模式确定,使用 CDC::SetROP2()可更改绘图模式。

2)映射模式:用于确定从逻辑坐标到设备坐标的映射方式。设备坐标是指窗口中相应的像素点位置。单位只有像素一种。而逻辑坐标依映射模式不同而单位各异。Windows 支持 8 种映射模式,其中 MM_ISOTROPIC 和 MM_ANISOTROPIC 是可编程的。这意味着是用户而不是 Windows 确定从逻辑坐标到设备坐标的转换方式,也就是用户来自行定制逻辑单位的实际大小。这两种模式最常用于根据窗口尺寸按比例自动调节画图输出大小的场合。两者之间的唯一区别在于:前者中 X 方向和 Y 方向具有同一缩放比例因子,设置映射模式的函数是 CDC::SetMapMode()。

3)坐标变换:调用 CDC::LPtoDP 可将逻辑坐标值转换为设备坐标值,调用 CDC::DPtoLP 可将设备坐标值转换为逻辑坐标值。

4)原点移动:默认方式下,DC 的原点位于相应显示平面的左上角。MFC 提供了两个 CDC 类成员函数,可改变原点位置。CDC::SetWindowOrg()移动窗口原点,CDC::SetViewportOrg()移动视图原点(详见 MSDN)。

5)用户坐标与屏幕坐标:前者是设立在窗口客户区左上角的设备坐标值,后者是原点位于屏幕左上角的设备坐标值,两者的相互转换用函数是 CWnd::ClientToScreen () 和 CWnd::ScreenToClient()。

2 图形刷新闪烁问题分析与解决方法

工业在线监测系统软件开发中,若每采集回一个数据后就更新相应的视图,屏幕会出现难以忍受的闪烁现象。因为图形绘制过程通常放在 OnDraw 或者 OnPaint 函数中,OnDraw 在进行屏幕显示时是由 OnPaint 进行调用的。当窗口由于任何原因需要重绘时,总是先用背景色将显示区清除,然后才调用 OnPaint,而背景色往往与绘图内容反差很大,这样在短时间内背

景色与显示图形的交替出现，同时由于绘制图形所用的时间较长，导致屏幕出现闪烁和不连续状态。

工业监测系统中实时曲线是对监测现场测量数据进行可视化，反映的是现场数据的实时性，以监测该点在现场工况变化的情况下的控制稳定性，因此在实现时需显示曲线的动态变化。通常当前点在曲线的最右端显示，随着时间的推进整个曲线动态地向左移单位间距的线段以消隐该旧线段，然后用最新的数据绘制最右端线段。实现曲线的动态平移要涉及曲线消影和重绘。图形刷新实现形式有下面几种方式[2]：

1)直接重新绘制图形，也就是调用 OnDraw()函数。这样通常由于背景色和前景色差异比较大，容易形成闪屏现象。

2)记录曾发生事件的区域，在窗口需要刷新时候重新调用窗口函数来执行这个事件。本方法主要考虑一个窗口中往往要显示成千上万个图元，而每个图元又是由点、线、圆等基本图形构成。如果真要在一次重绘过程重画这么多图元，可想而知这个过程是非常漫长的。这里采用 pDC->GetClipBox()得到裁剪区，去掉裁剪区外的绘图工作，显然可以提高绘制图形的效率，对解决屏幕刷新问题有一定的帮助。

3)在内存中保持一个显示输出的拷贝，当需要时候将拷贝复制到所需要显示的区域。这种方法也就是本文所提到的基于内存虚拟屏幕绘制图形的方法，比较通俗的说法就是将图形绘制在内存设备环境以及与显示兼容的位图中，然后从内存环境复制到显示器，这样就不必擦除背景，并且在图形绘制在屏幕之前，已经将图形绘制在位图中，然后直接复制到屏幕上，消除了在屏幕上直接绘图的时间和背景的擦除，从而一定程度上消除了闪烁。

3 基于内存虚拟屏幕绘制实时图形的实现

基于内存虚拟屏幕绘制实时图形也就是采用双缓冲技术：设置背景和前景两块内存设备上下文，将要画的图形先画在前景内存中，然后贴到显示缓存，要画新的一帧图象时，用背景内存覆盖前景内存，擦去前面一帧，再在前景内存中作经过位移的波形图象，然后再将前景内存贴到显示缓存。由于擦除的工作在内存中完成，因此基于内存虚拟屏幕绘制法可以避免图象抖动，避免屏幕闪烁。

在工业在线监测系统中，实时曲线的绘制通常有下列两种情形[3]：

1)无背景图形，实时曲线平滑移动。此种情况比较简单，图形的绘制在一个无背景色彩的绘图区域完成。当实时曲线还没有到达绘图区域的最右边时，曲线不需要平行移动。当到达绘图区域右边界后，每次只需要将实时曲线左移一单元的数据采集点间距，然后用背景色绘制最右边一单元数据采集点间距线段，擦去原来的线段，接着在最右边一单元数据采集点间距绘制实时更新后的线段。这样每次只需要重新绘制最右端数据采集单元间距的曲线，系统开销比较小，效率比较高。

2)有背景色的实时曲线。在工业在线监测系统通常采纳这种情况，这个时候通常应当采用基于内存虚拟屏幕绘制实时图形。这种情况下显然不能够直接采用用背景色 去删除最右单元的线段，而是采用下列策略。这里假设实时曲线绘制的显示矩形区域为 rect，添加辅助的矩形区域 rect1，rect1 的高度等于 rect 的高度，rect1 的宽度为 rect 的两倍。这样图形绘制可以分成三种情况：(a)当 rect 区域的曲线还没有绘制到它最右边界时候，直接在 rect 矩形区域绘制。(b)当曲线已经绘制到了 rect 区域的最右边界时候，将曲线从 rect 区域 BitBlt 到 rect1 中，并从 rect1 中部接着绘制曲线，随后从 rect1 中右边开始取 rect 宽度的曲线 BitBlt 到 rect 中，这样就实现了实时曲线和背景一起平行移动。(c)当曲线绘制到 rect1 的最右边界时候，将 rect1 中的图形清空，接着将重复步骤 b，将 rect 中的图形 BitBlt 拷贝到 rect1 中，接着在 rect1 的中部继续绘制图形，再拷贝到 rect 中显示。如图 1 所示意表示情况(b)的操作原理，首先将 rect 显示区域的图形拷贝到辅助区域 rect1 中，接着从 rect1 中部开始绘制图形，最后将 rect1 中相应位置的图形拷贝到 rect 区域进行显示。

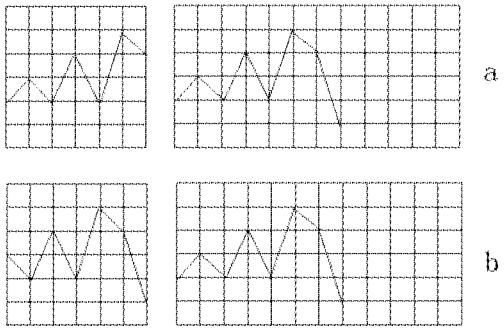

图 1 背景图形和实时曲线平行移动示意图

下面给出基于内存虚拟屏幕绘制有背景色的实时曲线的部分代码：

首先在类 CTestDrawDlg 的头文件中申明变量

```
public:
    int offsetx; //测试时候 x 轴坐标偏移量
    CDC m_dc; //设备上下文
    CBitmap m_bmp;//位图对象
```

在类 CTestDrawDlg 的实现文件中分别完成下列函数编码。

```
void CTestDrawDlg::OnPaint()
{
    CPaintDC dc(this); // 设备上下文
    CBrush bkbrush(HS_CROSS,RGB(0,128,0));//定义背景色刷子
    CRect rc;//定义矩形对象
    GetDlgItem(IDC_STATIC)->GetClientRect(&rc);//得到静态控件上的绘制图形区域
/* 分别创建内存兼容的 DC 和位图 */
    if(m_dc.GetSafeHdc()==NULL)
    {
    m_dc.CreateCompatibleDC(&dc);
      m_bmp.CreateCompatibleBitmap (&dc,2*rc.Width (),rc.Height ());
      m_dc.SelectObject(m_bmp);
      m_dc.SetBkColor(RGB(0,0,0));
      m_dc.FillRect(rc,&bkbrush);
    }
}
void CTestDrawDlg::OnTimer(UINT nIDEvent)
{
    static int x=0;
    static int y=0;
      static int flag=0;
    CDC* dc=GetDlgItem(IDC_STATIC)->GetDC();//得到静态控件的 DC
      CRect rect;//显示区域矩形
      GetDlgItem(IDC_STATIC)->GetClientRect(&rect);
      CRect rect1 (rect.right ,rect.top ,2*rect.Width (),rect.Height () );//辅助矩形
      CBrush bkbrush(HS_CROSS,RGB(0,128,0));
      x=x+offsetx;
    CPen pen1(PS_SOLID,0,RGB(255,0,8));
      CPen *oldpen1=NULL;
```

```
    oldpen1=m_dc.SelectObject(&pen1);
    if(x<=rect.right)//当绘制曲线没有超过 rect 最右边界时的编码
     {
        m_dc.MoveTo(x-offsetx,y);
    y=rect.Height ()-rand()%(int(rect.Height ()*0.8));
        m_dc.LineTo(x,y);
        if(flag==0)
            {
              dc ->BitBlt  (rect.righ,rect.top  ,rect.Width (),rect.
Height (),&m_dc,0,0 ,
                    SRCCOPY);
            }
        else
            {
             dc ->BitBlt  (rect.right  ,rect.top  ,rect.right -x,rect.
Height (),&m_dc,
    rect.right +x,0,SRCCOPY);
       dc->BitBlt(rect.right+rect.Width()-x,rect.top,x,rect.Height(),
&m_dc,
    0,0,SRCCOPY);
            }
      }
if(x<=rect.Width()+rect.right &&x>rect.right)
      {
          //按照有背景色的实时曲线的情形 b 编码
      }
if(x>rect.Width()*2+rect.left  )
      {
     //按照有背景色的实时曲线的情形 c 编码
      }
      dc->SelectObject(oldpen1);
      CDialog::OnTimer(nIDEvent);
}
```

本方法所绘制的实时曲线在刷屏时候没有闪烁,曲线平滑,和背景色一起进行平移。

4 结束语

本文采用基于内存虚拟屏幕绘制实时图形，主要思想是通过设置背景和前景两块内存设备上下文，将要画的图形先画在前景内存中，再通过与显示区域之间的转换连续动态显示相应实时曲线，克服了在线监测系统中实时曲线绘制过程中屏幕闪烁问题,对于在线监测系统软件编制有一定的实际意义。

参考文献

1 陈建春.Microsoft Visual C++图形系统开发技术基础.北京:电子工业出版社,1998
2 齐邦强.Windows 编程中图形刷新问题.计算机应用,2002:22(4)
3 刘定晟,杨俊,蒋迪清.用 Visual Basic 实现测控软件中的实时曲线和历史曲线.工业控制计算机,2001:14(4)

[收稿日期:2003.10.13]