**mfc 绘图闪烁问题的探究**

2018年07月09日 16:18:47 [atp1992](https://me.csdn.net/atp1992) 阅读数：36 标签： [mfc](http://so.csdn.net/so/search/s.do?q=mfc&t=blog)[双缓冲](http://so.csdn.net/so/search/s.do?q=%E5%8F%8C%E7%BC%93%E5%86%B2&t=blog)[闪烁](http://so.csdn.net/so/search/s.do?q=%E9%97%AA%E7%83%81&t=blog)[GDI+](http://so.csdn.net/so/search/s.do?q=GDI+&t=blog) 更多

个人分类： [C++](https://blog.csdn.net/atp1992/article/category/7156597)[software](https://blog.csdn.net/atp1992/article/category/7387705)

初学用windowsGDI进行界面绘制的时候经常会出现界面闪烁的问题，画面闪烁的直接原因是短时间内某部分的颜色的持续反差（这里的反差指肉眼可分辨的不同的颜色都称之彼此反差）变化，之所以要描述这么拗口，因为如果只是突然一两次的颜色反差变化，那只是颜色切换，画面只是变换了下，而颜色持续反差说的是闪烁部分的颜色在两个或多个差异较大的颜色之间来回变化，试想如果颜色一直变化但是缓慢过度，那么只是颜色渐变不会闪烁，如果颜色一直很快的变化，但大多数时间看到的是同一种颜色，那也看不出闪烁。

我们不要被显示的对象的复杂、颜色多样绚丽所蒙蔽，我们始终只需要关心单个像素的颜色变换，如果单个像素的颜色反复切换，那么就会闪烁，如果一个区域内有多个像素颜色反复变换，那么这个闪烁就会被观察到。

这里先看另一个广泛转载以至于没找到原帖的帖子中介绍的：

<https://www.cnblogs.com/piggger/archive/2009/05/02/1447917.html>

其实该博主对闪烁的解释和解决方法描述的很到位了，但是我还是想补充一下。

根据上面我总结的描述，然后我们看下平时遇到的几个典型场景分析，并用代码复现，代码说明见博文<http://www.straka.cn/blog/encapsulate-gdi-draw-method/>。

一对于画面前景背景颜色一致（准确的说是肉眼无法分辨出颜色差别，下同）的而言，无论如何绘制（频率）都不会闪烁。这么说好像是废话，其实是说明闪烁不会因为系统自身的绘制造成，系统绘制是个覆盖的过程，而不是一个清空再重填的过程。

1. for(i=0;i<1000;i++){
2. drawMethod.FillRect(rect,RGB(250,250,250),255);
3. drawMethod.DrawOnScreen();
4. Sleep(1); *//change it to simulate the different frequences*
5. }

二，只重复绘制一种颜色的区域，对于该区域每个像素始终只显示一个颜色，这部分也是不会闪烁的，这个其实和不停绘制背景色的情况一致。

三，对于同一个颜色（反差于背景色）在附近来回绘制，从像素角度，由于抖动造成该区域的边缘颜色来回变化，即在背景色和该前景色变化，而边缘显然是可见的，因而这样是会呈现闪烁的，

1. left = 0;
2. for(i=0;i<1000;i++){
3. offset+= 1;
4. offset %=7;
5. rect1.left = offset +10;
6. rect1.right = offset +40;
7. rect1.top = offset +10;
8. rect1.bottom = offset + 40;
9. drawMethod.FillRect(rect,RGB(0,0,0),255);
10. drawMethod.FillRect(rect1,RGB(250,250,250),255);
11. drawMethod.DrawOnScreen();
12. Sleep(1);
13. }

四，前景和背景色反差大并来回切换，这种情况下是否看得出闪烁在于那种颜色的时间占比能否达到绝对高，如果两种颜色的显示时间达到1:50，那是看不出闪烁的，而1:10，那么就很容易看出闪烁，当然这个时间比也必须是段时间内的，比如1秒内，所以可以用下述代码测试：

1. for(i=0;i<1000;i++){
2. drawMethod.FillRect(rect,RGB(0,0,0),255);
3. drawMethod.FillRect(rect1,RGB(250,250,250),255);
4. drawMethod.DrawOnScreen();
5. Sleep(1);
6. drawMethod.FillRect(rect,RGB(0,0,0),255);
7. drawMethod.FillRect(rect1,RGB(170,0,0),255);
8. drawMethod.DrawOnScreen();
9. Sleep(50);*//change this to simulate different proportions*
10. }

五，对于一个显示对象（颜色区域，复杂图形等）缓慢移动的情况，快速移动其实等同于上面的情况四中比例很接近的情况，因为我们始终考察的是某个小范围的颜色变化，对于快速移动的目标区域而言，是从原先的显示对象切换成了当前的对象，如果该目标区域反复切换则也是闪烁，而缓慢移动，因为对于沿路的内容只会从原先显示对象的颜色变为背景色，因而并不会闪烁。

1. for(i=0;i<1000;i++){
2. offset += 1;
3. offset %= 500;
4. rect1.left = offset;
5. rect1.right = offset+40;
7. drawMethod.FillRect(rect,RGB(0,0,0),255);
8. drawMethod.FillRect(rect1,RGB(250,250,250),255);
9. drawMethod.DrawOnScreen();
10. Sleep(10); *//change this to control the moving speed*
11. }

综上，我们经常遇到的闪烁分解下无外乎上述几种情况，对于常见的控件绘制闪烁，是由于重绘时的背景色和前景图案反差较大，每次重绘先用背景色擦除就会造成闪烁，而解决的办法基本都是屏蔽背景重绘和靠双缓冲绘制。

屏蔽背景重绘很简单:

在强制重绘的时候Invalidate(FALSE);传入false不重绘背景，true会重绘背景，

或者实现WM\_ERASEBKGND 消息的时候直接return true;

双缓冲原理也不复杂，就是每次绘制前先在内存中画好，然后直接将内存中的图案显示到显示设备上。由于我们要显示的画面内容经常是前后帧连续变化，差异较小的，也就是上述五缓慢移动的情形，因而是不容易闪烁的。双缓冲的代码我就直接引用前文链接中的代码，另外在我的另一篇博客里面也有封装好的代码利用了双缓冲减轻闪烁。

1. CDC MemDC; *//首先定义一个显示设备对象*
2. CBitmap MemBitmap;*//定义一个位图对象*
3. *//随后建立与屏幕显示兼容的内存显示设备*
4. MemDC.CreateCompatibleDC(NULL);
5. *//这时还不能绘图，因为没有地方画 ^\_^*
6. *//下面建立一个与屏幕显示兼容的位图，至于位图的大小嘛，可以用窗口的大小*
7. MemBitmap.CreateCompatibleBitmap(pDC,nWidth,nHeight);
8. *//将位图选入到内存显示设备中*
9. *//只有选入了位图的内存显示设备才有地方绘图，画到指定的位图上*
10. CBitmap \*pOldBit=MemDC.SelectObject(&MemBitmap);
11. *//先用背景色将位图清除干净，这里我用的是白色作为背景*
12. *//你也可以用自己应该用的颜色*
13. MemDC.FillSolidRect(0,0,nWidth,nHeight,RGB(255,255,255));
14. *//绘图*
15. MemDC.MoveTo(……);
16. MemDC.LineTo(……);
17. *//将内存中的图拷贝到屏幕上进行显示*
18. pDC->BitBlt(0,0,nWidth,nHeight,&MemDC,0,0,SRCCOPY);
19. *//绘图完成后的清理*
20. MemBitmap.DeleteObject();
21. MemDC.DeleteDC();

下载试验demo代码等见原博客：http://www.straka.cn/blog/flickering-in-mfc/