对 MFC 下使用 OpenGL 实现动画的两种方法特点分析

乔　立[1]　苏鸿根[2]
[1](中国科学院研究生院　北京 100037)　[2](中国科学院软件研究所　北京 100080)

摘　要　叙述了 MFC 下使用定时器和渲染线程两种方法来实现基于 OpenGL 的动画的编程过程。本文通过两个示例对这两种方法的特点进行了分析、比较,以明确其各自的适用范围。

关键词　MFC　OpenGL　定时器　线程

FEATURES ANALYSIS OF TWO METHODS USED IN CREATE ANIMATION WITH OPENGL AND MFC

Qiao Li[1]　Su Hongen[2]
[1](*Graduate School of Chinese Academy of Sciences , Beijing 100037)*
[2](*Institute of Software , Chinese Academy of Sciences , Beijing 100080)*

Abstract　Timer and render thread are two main methods used in computer animation programming. This paper analyszed their features ,then compared them.

Keywords　MFC　OpenGL　Timer　Thread

0　引　言

计算机动画的应用领域十分广泛。在具体实现计算机动画时,采用 MFC 与 OpenGL 相结合的办法,可以充分利用 MFC 提供的 Windows 程序框架和 OpenGL 强大便捷的绘画功能,很方便地开发出功能丰富、性能优秀的动画应用程序。在具体应用中,一般使用定时器或渲染线程来实现动画显示。

1　动画性能的测量

程序实现动画的性能用帧速率来衡量。帧速率是每秒的帧数(fps) ,即 1 秒内更新动画帧的次数。为了准确地测量时间,在 Windows 操作系统中,可以使用 QueryPerformanceFrequency 和 QueryPerformanceCounter 两个函数。第一个函数用来获取高精度的时间计数器的频率;第二个函数用来获取时间计数器的当前值。两个函数都将返回值存入 LARGE_INTEGER 类型的变量中。用计数器的当前值减去初始值,再除以计数器的频率,就能得到单位为秒的时间值,其精度高于百万分之一秒[1]。程序每调用一次 SwapBuffers ,就将帧数加 1 ,由此获得某段时间内的动画帧数。

2　使用定时器

在实现动画效果方面,使用定时器是比较简单和常用的方法。用 SetTimer 函数创建一个定时器,让它每隔一段时间(以 ms 为单位)就向窗口发送 WM_TIMER 消息,然后在对 WM_TIMER 消息的响应函数中触发对图像的刷新。在需要停止动画时,使用 KillTimer 函数删除定时器。

本文首先采用定时器方法进行编程,显示一个旋转的网状球体。程序以单文档,视图/文档结构的框架为基础。本文其他示例也采用相同的框架结构,实现相同的动画效果,以便于进行对比。使用 ClassWizard ,在 CtimerLoopView 中加入对 WM_CREATE、WM_DESTROY、WM_SIZE、WM_PAINT 和 WM_TIMER 的响应处理函数。

在 OnCreate 函数中,完成设置像素格式,创建渲染场景,设置绘图模式等工作。

OnDestroy 函数在程序结束时删除渲染场景。OnSize 函数在窗口大小产生变化时,重新定义渲染区的大小和坐标映射,以保持图像形状的一致。

OnTimer 函数对定时器发出的 WM_TIMER 消息进行处理,对网格球体的位置变量进行更新,然后调用 InvalidateRect 函数使绘图矩形区无效。

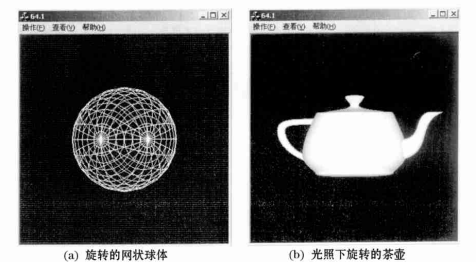
OnPaint 函数实现绘图工作,并计算帧速率。代码如下:

```
void CTimerLoopView::OnPaint()
{
  CPaintDC dc(this);
  HWND hWnd = GetSafeHwnd();
  HDC hDC = ::GetDC(hWnd);
  DrawWithOpenGL(); //绘图
  SwapBuffers(hDC); //执行缓冲区交换
   //计算帧速率
  nFrames ++;
  if(nFrames > 100) //每显示 100 帧计算一次帧速率
  {char CharBuffer[32];
```

收稿日期:2003 - 07 - 10。乔立,硕士生,主研领域:计算机图形学。

```
        LARGE_INTEGER CurrentTime;
        float fps;
        QueryPerformanceCounter(&CurrentTime); //获取当前时间
        fps = (float) nFrames/((float)(CurrentTime.QuadPart - StartTime.
QuadPart)/
              (float) TimerFrequency.QuadPart); //计算 fps
        sprintf(CharBuffer,"%0.1f",fps); //在窗口标题中显示帧速率
        GetParent()->SetWindowText(CharBuffer);
        nFrames = 0;
        QueryPerformanceCounter(&StartTime); //重新计时}
    }
```

在资源编辑器中修改主框架菜单,添加“开始动画”和“停止动画”两个命令。由“开始动画”命令启动定时器,用“停止动画”命令删除定时器。



(a) 旋转的网状球体　　(b) 光照下旋转的茶壶

图 1　使用定时器方法的运行结果

图 1(a)为示例程序 1 在 Windows2000 环境下的运行结果。程序在创建定时器时,将消息发送间隔,设置为 15ms,显示的帧速率为 64.1fps,和理论值(66fps)非常接近。事实上,当继续减小发送间隔时,会发现帧速率并不会因此提高,而延长发送间隔时,动画的帧速率也不是线性的递减,而是阶梯性的降低。图 2 为消息发送间隔时间与帧速率的关系曲线。这是因为定时器的时间精度并非 1ms。通过图 2,我们可以估算出,在 Windows2000 中,定时器的精度大约为 16ms 毫秒左右。当在 Windows98 下运行该程序时,帧速率为 18.2fps,由此估算出定时器的精度约为 55ms,这与很多技术资料上的论述是一致的[2]。定时器的这种特性,导致使用定时器实现动画时,对帧速率进行调节很不方便,因为帧速率的变化不是连续性的。

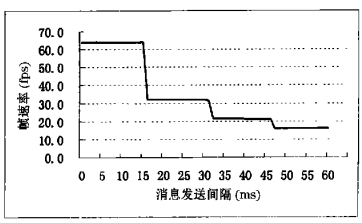


图 2　消息间隔 - 帧速率关系图

增加绘制图形的复杂程度,采用定时器方法显示灯光照射下一个旋转的茶壶,运行结果如图 1(b)所示,发现动画的帧速率依然保持在 64fps,这是因为在 16ms 的时间中,性能良好的计算机能够完成一帧比较复杂的图形的绘制任务。而 60fps 左右的帧速率,能够实现非常平滑的动画,基本能满足大多数应用的需要。在 Windows98 下,使用定时器只能达到 18fps,使得基于定时器的动画程序受到很大的制约,在 Windows2000 下,这种制约不复存在。操作系统和硬件技术的发展,显著拓宽了定时器的应用范围。但由于定时器自身的限制,不大可能获得更高的帧速率,所以不适合用在实时性要求非常高的应用中。这些应用包括军事模拟系统和商业飞行模拟系统等等。另外,定时器消息的优先级并不高,窗口收到的每个消息都会中断连续生成渲染帧的过程,在占用很多 CPU 时间的消息处理返回前,定时器消息将无法被响应。比如点击框架上的最大化按钮,在鼠标左键抬起前,动画会完全停止。对于很多游戏和动画模拟,要求窗口的随机维护活动不能中断动画循环或使帧速率明显波动,在这些应用中,也不适合采用定时器编程方法。

3　使用渲染线程的方法

作者在示例程序 2 中采用渲染线程方法编程,生成动画的重复帧。使用渲染线程在编程上比使用定时器的方法要复杂一些,结构上与基于消息处理的动画程序有很大不同。用 MFC 建立一个名为 RenderThread 的工程,在生成基本框架后,声明一个辅助线程,这个辅助线程要完成设置像素格式,创建渲染场景,绘制图像,处理窗口大小变化与删除渲染场景等工作。程序的主线程通过一个参数结构体与辅助线程通信,控制辅助线程。结构体包含有关事件状态标志,其定义如下:

```
struct THREADFLAG
{
  unsigned int Frames;      //帧计数器
  BOOL bResize;             //窗口大小变化状态标志
  BOOL bTerminate;          //渲染线程终止标志
  BOOL bModifyFlag;         //以上 2 个标志是否发生变化的标志
};
```

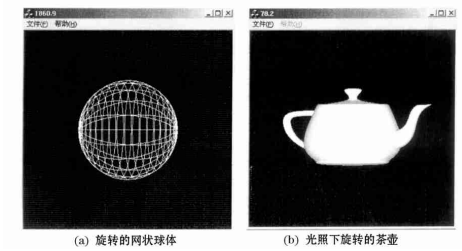
在线程间进行通信时,为防止两个线程同时访问一个系统资源,采用临界区方法来同步对结构中变量的访问。先声明一个 CRITICAL_SECTION 类型的结构,在 CRenderThreadView::OnCreate 函数中调用 InitializeCriticalSection 函数对其进行初始化。任何线程访问结构体中变量前都要调用 EnterCriticalSection 函数进入临界区,访问完后调用 LeaveCriticalSection 函数离开临界区。当程序结束时,调用 DeleteCriticalSection 函数删除临界区。

CrenderThreadView::OnCreate 函数还要负责初始化共享数据,创建并启动渲染线程,代码如下:

```
CRITICAL_SECTION g_cs; //声明一个临界区结构
 ……
int CThreadLoopView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
 ……
   InitializeCriticalSection(&g_cs); //初始化临界区
   FlagData.Frames = 0;        //初始化共享数据
   FlagData.bResize = FALSE;
   FlagData.bTerminate = FALSE;
   FlagData.bModifyFlag = TRUE;
   QueryPerformanceFrequency(&timerFrequency); //获取时钟频率
   QueryPerformanceCounter(&startTime); //帧速率计算的起始时间
     m_pThread = AfxBeginThread(RenderThreadFunc,NULL,
                                THREAD_PRIORITY_NORMAL,0,
                                CREATE_SUSPENDED); //创建渲
                                染线程
     m_pThread->ResumeThread(); //启动渲染线程
     SetTimer(1,5000,NULL); //每 5 秒计算一次帧速率
     return 0;
   }
```

每当更改 FlagData 结构中的标志时,都把 bModifyFlag 设置为 TRUE。渲染线程通过 CheckStates 函数访问 FlagData 结构中的数

据,每绘制一帧都调用它。CheckStates 函数会首先检查 bModifyFlag,以避免在没有更改任何内容的情况下检查其他标志。计算帧速率的方法也有所变化:在主线程中使用定时器,每 5 秒计算一次帧速率。程序结束时,在 CrenderThreadView::OnDestroy()函数中将 FlagData 结构中的 bTeminate 标志置为 TRUE,通过渲染线程终止。



(a) 旋转的网状球体 (b) 光照下旋转的茶壶

图 3 使用渲染线程方法的运行结果

图 3(a)为示例程序 2 的运行结果。示例 2 与示例 1 的运行环境完全相同,但可以发现动画的帧速率显著提高,达到 1860fps。使用渲染线程时的帧速率受渲染图像的复杂程度、CPU 和显卡性能的影响;图形复杂,帧速率会明显降低;硬件性能好,帧速率会大幅度提高。图 3(b)为使用渲染线程方法实现复杂动画的运行结果,其平均帧速率为 78fps 左右,明显低于实现简单动画的帧速率,这是因为绘制每一帧的工作量大大增加,所需的时间也相应变长。

由于渲染线程只是简单的循环运行,不必理会其他操作,所以能充分表现 CPU 和显卡的性能,而且在主线程处理其他消息时,动画依旧能正常显示。渲染线程的这种特征使其非常适合应用于动画演示程序或显示性能测试程序。

在实际应用中,高于显示器垂直刷新率的帧速率对观察者来说是没有意义的,因为有些帧在显示之前就被新的帧所覆盖,而且过高的帧速率会消耗很多 CPU 时间,造成资源的浪费。

针对具体应用的要求,可以在渲染线程的循环中加入一个 Sleep 函数,将其设置为睡眠状态。Sleep 函数只需要一个参数,就是渲染线程需要挂起的毫秒数,因为 Sleep 函数的计时精度是 1ms,所以可以很方便地将动画的帧速率调整到合适的范围,以便让 CPU 进行其他操作。很多支持 OpenGL 的显卡可以将显示模式设置为垂直同步,使帧缓存的刷新速率不会超过屏幕的垂直刷新率。渲染线程可以获得很高的帧速率,利于实现非常平滑的动画,在性能优越的硬件环境上,可以获得极佳的动画效果,而且不影响程序的交互性。

4 结 论

表 1 为两种方法的性能对比,其中帧速率稳定性一项是指动画帧速率是否易受其他操作的影响。表 1 中数据为本文示例程序的运行结果,好与差,困难和容易只是二者相对而言。

表 1 两种方法的性能对比

性能 / 方法	平均帧速率		帧速率可调节性	帧速率稳定性	编程难易度
	简单动画	复杂动画			
定时器	64fps	64fps	差	差	容易
渲染线程	1860fps	78fps	好	好	困难

通过对两种方法的分析对比,可以得出以下结论:使用定时器实现动画,编程简单,在较好的运行环境下可以获得不错的帧速率,能满足大多数一般动画应用的要求。其不足之处是受其他操作的影响严重,帧速率不够高而且不易调整,不适用于对动画品质要求很高,交互性强的应用,比如 3D 实时游戏,军事或商业上的模拟操作等等。使用渲染线程实现动画,在程序设计上比较复杂,需要考虑的因素较度多。渲染线程可以充分发挥硬件的性能,可以获得很高的帧速率,不易受其他操作的影响,而且可以很方便的调节帧速率。渲染线程方法的适用范围很广,如高品质动画的演示,便件显示性能的测试,3D 游戏,虚拟现实场景等等。

参 考 文 献

[1] OPenGL 超级宝典(第二版),北京:人民邮电出版社,2001 年.

[2] Jeff Prosise, Programming Windows with MFC(Second Edition),北京:北京大学出版社,2000 年.

(上接第 28 页)

$$= \frac{1}{n}\sum_{i=1}^{n}[-t_A(x_i)\log_2 t_A(x_i) + (-f_A(x_i)\log_2 f_A(x_i))]$$

为 vague 集 A 的模糊熵。

注 1:vague 集 A 的模糊正熵刻画了 U 中元素 x 属于 vague 集 A 的不确定度,E_O 越小,x 属于 vague 集 A 的不确定度越小;vague 集 A 的模糊负熵刻画了 x 不属于 vague 集 A 的不确定度,E_N 越小,x 不属于 vague 集 A 的不确定度越小;vague 集 A 的模糊熵刻画了 U 中元素 x 与 vague 集 A 之间关系的不确定程度,E 越大,我们对 x 与 A 的关系(是属于 A 还是不属于 A)了解得越少。

注 2:当 vague 集 A 退化为普通 fuzzy 集时,A 的模糊熵与文[4]中的普通 fuzzy 集的模糊熵相一致。并且,当 $A(x)=0.5$, $\forall x \in U$ 时,A 的模糊熵取得最大值 1,这与直觉是相一致的。

注 3:顺便指出:文[4]中的式(2.2) $h(u) := -u\ln u-(1-u)\ln(1-u)$ 应改为 $h(u) := -u\log_2 u-(1-u)\log_2(1-u)$,否则,不满足当 $A(x)=0.5$, $\forall x \in U$ 时,A 的模糊熵取得最大值 1。

例 3 设 $A=\{<0.2,0.3>\mid x\in U\}$, $B=\{<0.8,0.9>\mid x\in U\}$。则:

$$E_O(A)=\frac{2}{n}\sum_{i=1}^{n}-0.2\log_2(0.2)=-0.4\log_2(0.2);$$

$$E_N(A)=\frac{2}{n}\sum_{i=1}^{n}-0.7\log_2(0.7)=-1.4\log_2(0.7);$$

$$E_O(B)=\frac{2}{n}\sum_{i=1}^{n}-0.8\log_2(0.8)=-1.6\log_2(0.8);$$

$$E_N(B)=\frac{2}{n}\sum_{i=1}^{n}-0.1\log_2(0.1)=-0.2\log_2(0.1);$$

显然,$E(A)\neq E(B)$。

参 考 文 献

[1] 李凡、卢安、蔡立晶,"关于 Vague 集的模糊熵及其构造",《计算机应用与软件》,19(2002),No.2,10～12.

[2] 李凡、卢安、余智,"一类 Vague 集模糊熵的构造方法",《华中科技大学学报》,29(2001),No.9,1～3.

[3] 何颖瑜、王国俊,"关于 Fuzzy 格的若干注记——兼评直觉主义模糊集",《模糊系统与数学》,11(1997),No.4,1～4.

[4] Wen-Jun Wang, Chih-Hui Chiu, Entropy and information energy for fuzzy sets, Fuzzy Sets and System 108(1999),333～339.

[5] W.L. Gau and D.J. Buehrer, Vague sets, IEEE Trans. Systems Man Cybernet. 23(1993),610～614.

[6] L.A. Zadeh, Fuzzy sets, Inform. and Control 8(1965),338～356.