基于 OpenGL 程序设计的优化探讨

The research of optimization based on OpenGL program design

田金琴　（西南交通大学信息学院）
（宁夏银川西北第二民族学院计算机系）
丁红胜　（西安石油大学计算机学院 710065）
（宁夏银川西北第二民族学院计算机系）

摘要：从 OpenGL 渲染流程的角度出发，按照 OpenGL 程序的执行顺序划分为三个层次：CPU 层，几何子系统，光栅化子系统。在分析出每层中的瓶颈之后，结合 OpenGL 提供的 API 函数的功能特性、实现方式和硬件支持及其程序实现的图形效果等方面综合考虑，针对每层所涉及的操作，总结了优化各层瓶颈的方法。

关键词：OpenGL　程序优化　渲染流程

1　引言

OpenGL 是一个专业性的 3D 程序接口，全称是 Open Graphics Library（开放性图形库）。是目前开放式的三维图形标准，目前 OpenGL 最新版本是 OpenGL2.0。利用 OpenGL 技术已经广泛应用于图形工程开发中，如复杂 3D 建模，音频系统，摄像机控制，粒子系统，人机交互技术等[1]。图形处理程序的一个特点是处理的数据量非常庞大，而精细、流畅、大容量信息的画面所处理的数据量更大，这使得在程序执行中会占用大量的计算机硬件资源，而合理的程序组织方式和优化的 OpenGL 程序可以得到最优的性能和最小化的硬件资源开销。本文从 OpenGL 渲染流程的角度出发，按照 OpenGL 程序的执行顺序的三个层次，CPU 层，几何子系统，光栅化子系统，在分析出每层中的瓶颈之后，结合 OpenGL 提供的 API 函数的功能特性、参数设置、实现方式和硬件支持及其所实现的图形效果等方面综合考虑，针对每层处理数据的不同特点，总结了各层的优化方法，优化各层出现的瓶颈。

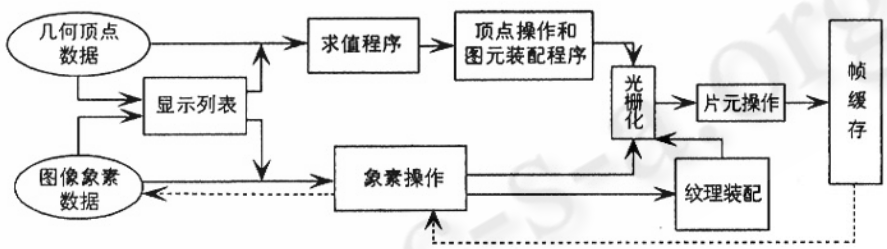

图 1　OpenGL 基本工作流程

2　OpenGL 操作流程

Open GL 的最主要工作就是将二维物体和三维物体描绘至帧缓存[2]。基于 OpenGL 绘制图形的原理如图 1 所示。

大多数 OpenGL 的实现均采用图 1 所示的渲染流水线[3]。几何数据（顶点，直线和多边形）经过求值程序和顶点操作，而象素数据（象素，图像和位图）的处理则经过象素操作，纹理装配后，这两种数据都经过相同处理步骤写入缓冲区。无论是几何数据，还是象素

数据，都可以存储在显示列表中，或者直接进行处理。所有的几何图元最终都是由顶点描述的。如果使用了求值程序，数据将被转化为顶点，并被视为顶点进行处理，还可以将顶点数据存储在顶点数组中，然后使用他们，然后经过光栅化处理变成片元，对于象素数据，执行象素操作。并将结果存储在纹理内存中。然后用多边形的点画模式，或者经过光栅化变成片元。最后，片元经过一系列的片元操作，将最终得到的象素写入帧缓存。

3 OpenGL 程序的流程及瓶颈分析

3.1 OpenGL 程序的流程

OpenGL 程序的执行与应用环境密切相关，不同的硬件系统和系统软件对其支持不太一样。从图 1 的数据处理过程来分析，OpenGL 程序的执行可以分成 3 个层次[4]：

(1) CPU 子系统，应用程序在 CPU 中执行，对图形子系统发送命令；

(2) 几何子系统，包括多边形操作，如坐标转换，光照，裁减和生成纹理等；

(3) 栅格化子系统，包括各种象素的操作，如纹理映射，alpha 融合，颜色值写入帧缓存中等复杂操作。

后两层中的一些操作在有些有些硬件上可以得到加速。

3.2 确定程序中瓶颈的位置

OpenGL 程序中的瓶颈可能存在有几何体的瓶颈，栅格化的瓶颈及其它影响性能的因素(见表 1)。我们采用基本策略是"计算执行程序的时间，即在不影响程序性能的前提下，修改流程中某一阶段的代码，结果若性能的变化相差不大，则不存在瓶颈"，对各流程层次的测试方法见表 2。测试性能提升的方法有人的主管感觉和测试帧速率等方法等[5]。

4 OpenGL 程序设计的优化

OpenGL 程序性能优化我们从按照 OpenGL 程序的三个层次分类考虑，CPU 层重点是优化程序中数据的组织结构和程序流程结构，目标是按照最有效的方式组织程序的流程，避开不必要的操作，比如像算法，数据结构，循环优化等等，可以参见具体的实现语言的优化原则。我们重点探讨关于 OpenGL 中的后两个层次的优化问题。

表 1　影响性能的因素及受影响的相关程序层次

性能参数	流程层次
各多边形的数据量	所有层次
程序耗费时间	应用程序
多边形变换速率和模式设置	几何子系统
一帧中的所有多边形的数量	几何体和栅格化子系统
填充的象素数	栅格化子系统
按给定模式设置下的填充速率	栅格化子系统
清屏或清除缓冲区时间	栅格化子系统

表 2　各程序层次的瓶颈测试方法

流程层次	方法
CPU 子系统	简单的方法是用 glColor 替代 glVertex( ) 和 glNormal( )，如果性能不能提升，那么 CPU 层就是瓶颈。
几何子系统	用较少的大多边形替代很多小的多边形或者通过使光照或剪切无效来测试，如果性能提升，则说明应用程序受几何/转换的限制。
光栅化子系统	减少光栅化的工作量的方法就是是窗口变小，或者禁止针对每个象素的操作，如纹理贴图，混合，或者深度测试等，如果性能提高，则说明系统有瓶颈。

4.1 几何子系统的优化

几何子系统的优化的目的实质上是对多边形的操作的优化，降低图元操作的数量，或者在光栅化时减少每个图元处理的象素数。我们归纳了以下 9 个方面来提高几何子系统的显示性能。

4.1.1 使用最基本的对象连接，减少多边形工作量。如：GL_LINES，GL_LINE_LOOP，GL_TRIANGLE_STRIP，GL_TRIANGLE_FAN 和 GL_QUAD_STRIP，它比单个的线，三角形，或者多边形在描述对象需时用较少的顶点，这将减少数据传输。但是如果绘制非常多的矩形，最好使用 glBegin(GL_QUADS) ... glEnd( )。

4.1.2 使用平面规则多边形构造显示对象，以减少多边形分化(tesselate)和顶点处理的工作量。

4.1.3 显示列表的应用。使用显示列表封闭被画的对象，而且显示列表数据可以存放在显示子系统中的显存中，而不是在内存中，因此不需要进行内存到显存的数据移动，当远程渲染的时候显示列表非常有效。

4.1.4 使用顶点数组。顶点数组具有最高的内存复制效率，调用函数 glInterleavedArray( ) 包装好的顶点数据的执行速度更快，在处理大数据量时比显示列表更有效，调用函数 glDrawElement( ) 不但可以减少其他函数的调用，而且可以减少每个顶点由于重复使用的计算量。

4.1.5 不计算不需要的顶点操作。比如当光照禁止时，不再需要调用 glNormal 运算，当贴图禁止时，不需调用 glTexCoord 运算等。

4.1.6 使用平面渲染(flat shading)优化绘制模式。

4.1.7 光照性能的优化。按照一般原则，我们尽可能使用简单光照模型，即在一个无限视口中使用一个无限的光源，为了获取高性能的光照，可以使用如下优化方法：

(1) 避免使用局部视口光照(因为局部光源比无限光源更费时)，不要平凡的更换 GL_SHININESS 材质参数。

(2) 有时我们可以给顶点赋以颜色而不是计算顶点向量来减少计算量。

(3) 避免使用双侧光照。

(4) 避免使用聚光灯。

4.1.8 使用 glVertex, glColor, glNormal and glTexCoord 向量版本，如 glVertex3fv(v)，而不用 glVertex3f(x,y,z)。向量版本的操作都有相同的数据格式，如果结合顶点数组，那么数据在同一片内存存储区域，这样可以加快数据的处理速度。

4.1.9 最小化 glBegin/glEnd 之间的代码

因为对于高端的显示系统来说，最大化性能提升要求数据的传输尽可能的快，避免在 glBegin/glEnd 之间写不相关的代码。比较如下两段代码分析可知，第二段优于第一段。

```
glBegin( GL_TRIANGLE_STRIP );
for (i=0; i < n; i++)
{
  if (lighting)
    {
    glNormal3fv( norm[i] );
    }
glVertex3fv( vert[i]);
}
glEnd( );
```

```
if (lighting)
    {
glBegin( GL_TRIANGLE_STRIP );
  for (i=0; i < n ;i++)
      {
   glNormal3fv( norm[i] );
   glVertex3fv( vert[i] );
      }
    glEnd( );
    }
  else {
glBegin( GL_TRIANGLE_STRIP );
  for (i=0; i < n ;i++) {
    glVertex3fv( vert[i]);
        }
    glEnd( );
    }
```

4.2 栅格化子系统的优化

把经过投影变换的点、线、多边形、位图或图象的象素转换成碎片，每个碎片对应于帧缓冲区中的象素，这个过程就是光栅化(rasterize)。光栅化常常在 OpenGL 的软件实现过程中是一种瓶颈。栅格化子系统的优化的目的实质上是优化像素的操作。我们总结了以下 8 个方面来优化栅格化子系统。

4.2.1 根据情况可以设置深度测试无效。glDisable(GL_DEPTH_TEST)，例如，如果能够保证后景对象先画，前景对象后画，在没有深度测试的时候也能够正确被绘制。

4.2.2 不需要时设置深度抖动无效容许抖动是缺省设置，(查资料)

4.2.3 尽肯能根据实际图形的特点，使用多边形背面消除技术，glEnable(GL_CULL)。如果使用的是一个封闭的多边形或其它对象，其内部是不可见的，也就没有必要再绘制这些内部的多边形。

4.2.4 在光栅化的时候，模板混合，点画，alpha 测试和逻辑操作都可能会引起额外的时间消耗，当确定不需要的时候，我们最好将这些设置成无效状态。

4.2.5 减少屏幕窗口大小或者降低分辨率。如果小窗口和低分辨率的图像能够接受的话，这是降低光栅化操作的一种最简单的方法。

4.2.6 纹理贴图的优化。纹理贴图是一种很费时的操作，但我们可以从以下几个方面来简化纹理操作的复杂性。

(1) 使用最有效的象素格式：GL_UNSIGNED_BYTE 是处理速度最快的一种数据格式。

(2) 将纹理贴图包装至纹理对象中或者显示列表中。

(3) 尽量使用 MipMap 纹理。

(4) 使用更为简单的取样函数，纹理环境为 GL_

REPLACE,纹理过虑的参数为 GL_NEAREST。避免费时的过虑操作。

4.2.7 避免像素操作。在 OpenGL 的实现中,都是使用纯软件的方法实现从系统内存到显存的复制,glBitmap,glDrawPixels,glReadPixels,glCopyPixels 这些将会中断整个图形流水线的执行,等待硬件空闲后使用 CPU 完成,它们将大大降低程序的执行效率。一种有效的方式就是使用纹理替代像素操作。

4.2.8 使用同时清除颜色缓存和深度缓冲区的方法。glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);这种同时操作比两次单独的 glClear(GL_COLOR_BUFFER_BIT);glClear(GL_DEPTH_BUFFER_BIT)操作更快。

4.3 其它优化

4.3.1 动画显示时,保证可行的帧速率,适当降低图像质量。

当高速的动画和高质量的 OpenGL 图像不能同时兼得时,最好得折中处理方式是:图像静止时,提高图像显示的精细度;图像运动时,降低图像显示的精细度,保证画面的流畅性。提高动画流畅性,可以采用如下方法:

(1) 使用双缓存功能,加速动画速度。帧速率越高,动画越流畅,双缓存可以让系统硬件在显示前缓存帧的同时,给后缓存中进行绘制,然后使用 SwapBuffers()将后缓存中的数据交换到前缓存中显示,只要保证生成图形的计算时间小于硬件的刷屏时间,它就能够提供平滑的动画。

(2) 优化帧速率,减少绘制帧时间。如:禁止使用专门为显示精细画面设置而又耗时的操作,如抖动,平滑阴影,纹理贴图,反走样等。

4.3.2 在 Window 系统环境下的优化。

OpenGL 程序和 Window 系统结合时,Window 提供了自己的一套管理图形显示模式,为了最小化系统开销,我们应该注意以下几个方面。

(1) 最小化调用 glXMakeCurrent();这个操作的会引起硬件之间不同部件之间的数据的大量移动操作。

(2) 权衡性能,设置像素颜色格式。在微机上,颜色深度为 24 位的图形比 12 位或 8 位的要慢,我们可能需要在性能和质量之间作出平衡来设置颜色深度。

(3) 避免使用 OpenGL 和当前 windows 的混合渲染

OpenGL 容许自己和当前 Windows 系统对同一个窗口界面进行渲染。要使用这种功能,要求正确的同步功能使用。例如,用 OpenGL 绘制了三为场景,调用 glXWaitX()同步,然后再调用 Windows 绘制文本。但这种同步会降低系统性能。

(4) 在 SwapBuffer()函数执行后,不要立即调用 OpenGL 函数,这是因为 OpenGL 函数的执行要等到上一帧显示后才能执行。但在垂直跟踪之间的循环等待过程中可以进行不带图形操作的命令。

5 总结

优化的过程在有些情况下实质上是一种显示效果和性能的折衷,当然,尽量避免一些不必要的的开销会给程序执行在不损失效果的前提下带来明显的性能提升。本文从 OpenGL 渲染流程的角度出发,按照 OpenGL 程序的执行顺序的三个层次,CPU 层,几何子系统,光栅化子系统,确定了系统中每层的瓶颈之后,针对每层处理数据的特点,总结了各层的优化方法,这些方法的合理应用,将会设计出高性能的 OpenGL 程序。

参考文献

1 和平工作室,OpenGL 高级编程与可视化系统开发[M],中国水利出版社,2003。

2 杨承志等,基于 OpenGL 的 ActiveX 控件的实现[J],昆明理工大学学报(理工版)2005 年 2 月,30(1)。

3 OpenGL 体系结构审核委员会 著 OpenGL 编程指南(第四版)[M] 2005.4 7 杨日容。

4 乔林、费广正,OpenGL 程序设计[M],北京 清华大学出版社,2000.4。

5 Richard S. Wright, Jr. Michael Sweet ,徐波译,OpenGL 超级宝典(第三版),2005 年 9 月。