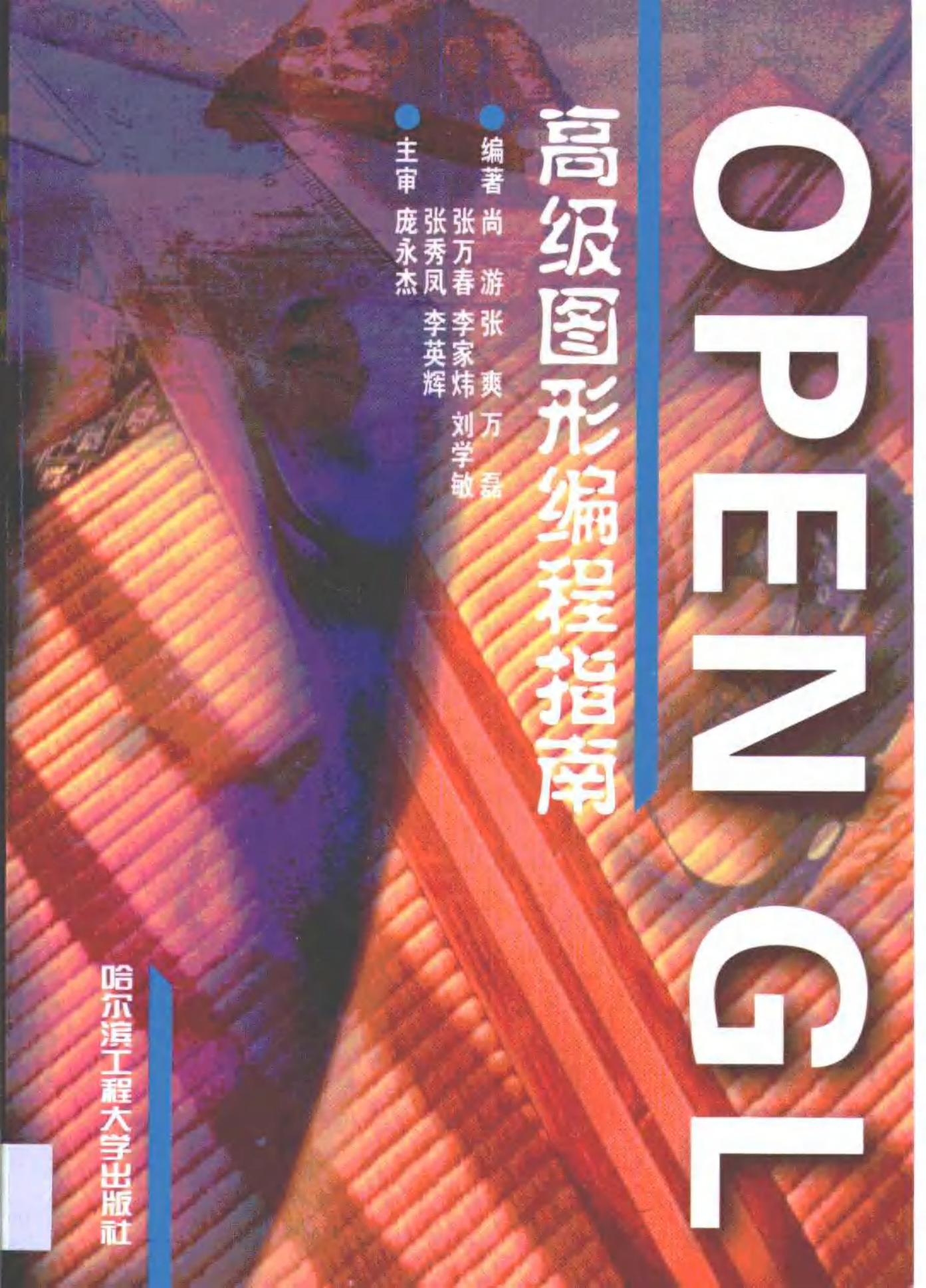

OPENGL

高级图形编程指南

编著 尚游 张爽 万磊
张万春 李家炜 刘学敏
张秀凤 李英辉
主审 庞永杰

哈尔滨工程大学出版社

OpenGL
高级图形编程指南

编　著　尚　游　张　爽　万　磊
　　张万春　李家炜　刘学敏
　　张秀凤　李英辉
主　审　庞永杰

哈尔滨工程大学出版社

内　容　简　介

本书系统地介绍了 OpenGL 高性能图形编程技术。编著者从基本的编程原理开始，由浅入深，详尽阐述了 OpenGL 高性能三维图形编程技术技巧。本书贯穿了大量的编程实例，兼顾了 PC 机和工作站用户，并创造性的设计了渲染平台与建模平台的接口，具有很强的实用性。同时，本书对基础的图形学、计算机学背景知识基本理论的透彻分析，将带着读者在计算机图形的艺术世界中遨游。

本书资料全面，并得益于强大的工程应用背景（水下智能机器人技术），是迄今为止国内最为详尽、系统的 OpenGL 参考资料。本书适用于高性能三维游戏、动画、广告制作以及动力学系统可视化仿真、虚拟现实等领域，是高校和高科技公司等单位开发图形软件的必备资料。本书还可以作为大专院校及高科技公司的培训教材，更是科技人员学习和掌握 OpenGL 图形编程技术不可多得的参考书。

OpenGL 高级图形编程指南
OpenGL Gaoji Tuxing Biancheng Zhinan
尚　游　等编著
责任编辑　陈晓军
*
哈尔滨工程大学出版社出版发行
新　华　书　店　经　销
哈尔滨工业大学印刷厂印刷
*
开本 787mm×1 092mm 1/16 印张 37.5 字数 835 千字
1999 年 1 月第 1 版　1999 年 1 月第 1 次印刷
印数：1—5 000 册
ISBN 7-81007-932-8
TP·87　定价：68.00 元

0 绪　论

0.1 简介

作为高科技的重要工具和载体，计算机在现代社会发展中起着不可估量的作用。而图形处理，是现代计算机区别于以前计算机的一个显著特征，使计算机走出了呆板的单一枯燥计算，在保证完成任务的同时更加接近真实世界，从而大大扩展了计算机应用领域，并带给编程人员和应用者丰富多彩的视听感受。

OpenGL 是 SGI 公司开发的一套高性能的图形处理系统，是图形硬件的软件界面，GL 即代表图形库(Graphics Library)。通过 OpenGL，程序员可以创建交互式应用程序，实现具有逼真效果的三维图形图像，从而在要求高度模拟真实世界的诸多领域中都可以大显身手。

以 AUTOCAD 和 3DSMAX 为代表的通用图形处理软件平台，在工程绘图或是动画制作、广告设计中各有所长，应用背景十分广泛。OpenGL 与这些软件包有着本质的不同。它超越了处理单幅的静态图形，在实时的仿真三维实体及其环境、从而得到真实的视觉效果方面显示出强大的优势。更为重要的是，在渲染场景的同时，实时得到所需要的大量数据信息，并进行跟踪计算，为复杂系统的实时仿真开辟了广阔的前景。

本书根据编者的实际应用经验，参照大量的原始资料，通过大量的实例，详尽的阐述了如何利用 OpenGL 图形系统进行程序设计，从而根据不同需要实现丰富多彩的三维视觉效果。

0.2 广阔的应用背景

应用程序可视化是计算机程序设计的发展趋势。在工程应用的许多领域中，计算机仿真的蓬勃发展使得人们可以不必在真实试验场便可以获得许多系统模型的性能，即使是真实试验，也少不了仿真系统对试验的再现。计算机仿真已经成为工程设计和软件开发人员不可或缺的重要工具。

涉及图形图像处理的应用领域，尤其是要求高实时性、大计算量和渲染量的仿真中，如：动力学系统的可视化仿真；结构应力有限元分析；人体医学扫描数据显示；交互式电子游戏开发；三维造型设计；动画制作；广告设计。

这些都是 OpenGL 图形库的用武之地。不难看出，上述各个领域都是现代计算机应用的热门方向。图形处理方兴未艾，随着要求不断提高，竞争不断加剧，OpenGL 图形库必然占据不久以后的图形处理领域。

虽然 OpenGL 目前主要应用于图形工作站中的软件平台如 IRIS Performer，但是越来越多的软件开发商们已经注意到了 OpenGL 图形库在微机上所具备的潜在力量。包括微软在内的许多著名公司所提供的编译平台，如 VISUAL C++ 5.0 和 POWER STATION 4.0

(FORTRAN90 标准)及其以上的版本中,都将 OpenGL 图形库作为其一个重要的模块扩展进来,借以大大增强该编译平台对图形的处理能力。由此将 OpenGL 实时渲染图形并得到相应数据的功能,与语言编译平台所专长的数值计算、过程实现等功能结合起来,从而得到仿真与运算相结合的新概念,并带给人全新的视觉感受。经典的三维造型设计软件 3DSMAX 在 R2 版本中也提供了与 OpenGL 图形库的接口。这不但从一个侧面表明了广大开发商对该图形库强大能力和潜在生命力的认可,同时也证明了它的良好的可移植性。我们坚信在计算机硬件、软件同时走向标准化、统一化的大趋势下,将会有越来越多的我们熟悉的编译平台将这一图形库吸纳进来。

0.3 本书的主要特色

OpenGL 是一种功能强大,应用与开发前景广阔的软件平台,顺应着计算机工程应用可视化的潮流。然而目前在图书市场上还没有较为全面和系统的操作指南和参考资料。本书便是根据市场的需求应运而生的。它具有以下几个特色。

1. **较为持久的生命力** SGI 开发的这一软件,走上应用时间并不长,目前还没有在计算机应用领域广泛的发展起来,正处在上升阶段,在几年甚至十几年内不会落伍。这保证了本书所载的 OpenGL 应用技术,跟其他的计算机应用类书籍相比将具有较强的生命力。而且 OpenGL 作为编程语言的一个特殊应用扩展,对其基本技术的全面掌握是学习这一优秀软件不可跨越的阶段。另外,本书还考虑了高级编程人员的需要,设计了库函数详解部分。这使本书在 OpenGL 从入门到精通的各个阶段都能起到重要的参考作用。

2. **丰富、完备的参考资料** 本书主要的参考资料来自 SGI 工作站上的原文在线帮助、编程指南和库函数说明,辅助的参考材料为 VISUAL C++ 5.0, 3DSMAX 和 FORTRAN POWERSTATION 的 OpenGL 帮助和库函数说明。本书的编著者均是计算机应用水平较高的博士、硕士,大多具有应用 OpenGL 编程的实际经验。结合我们编程的实际经验,加人自己的理解和分析,我们编制调试了大量的例程,某些涉及了 OpenGL 的高级应用,如纹理(texture)及变形(morph)等,已经具有相当的水平。

3. **到 PC 上的应用扩展** 勿用讳言,OpenGL 自呱呱坠地以来,其生活圈一直主要在工作站这一摇篮中。但是最近,个人计算机计算速度的大幅度提高使得它们与工作站的差距在不断缩小。虽然在相当长的一段时间内还不能期望其与专门从事图形处理的 SGI 工作站相比,但是已经具备运行 OpenGL 应用程序的能力,软件商们也不甘示弱,各个软件编译平台都将 OpenGL 模块拓展集成到自己的麾下,使得熟悉 C、C++、FORTRAN、PASCAL、BASIC 等等语言及其相应编译器 VC++、C++ Builder、POWERSTATION、DELPHI、VB 的编程人员可以在 586 以上的微机上编制非常 COOL 的 OpenGL 程序。OpenGL 已经渐渐走向成熟期。我们在编写过程中,考虑到我国 PC 机的普及,把程序的开发与编译都主要集中在微机的环境下,同时对在微机下使用、开发的过程进行了专门的介绍。本书面向的读者是广泛的。

4. **渲染平台与建模平台的互相扩展** 与其它图形处理软件如 AUTOCAD、3DSMAX 等等相比,OpenGL 最强大的优势在于其实时渲染复杂场景的能力。如果它能够与广泛使用着的建模平台相结合起来,这将使 OpenGL 在不久的将来迅速地占领图形处理领域。我

们在编写过程中，专门组织了力量对 OpenGL 与 AUTOCAD 的文件接口进行了探索性的研究，并取得了一定的成绩。本书的第十五章就是这些成果的集中体现。

0.4 本书的主要内容安排

本书包括两大部分：第一部分为 OpenGL 编程指南，详尽而系统地介绍了 OpenGL 的编程技术和技巧。第二部分是 OpenGL 库函数详解，系统介绍了 OpenGL 图形库及相关库的函数说明。

另外，本书列出了几个附录。这些附录对帮助程序员理解 OpenGL 的原理及机制，以及编写高效的应用程序是不无裨益的，其中对库函数的介绍部分是以应用类别排列，与库函数详解相区别，力争在这块有限的空间给 OpengGL 使用者提供更大的方便。书中在这一部分还在术语表中给出了书中用到的的关键术语的定义。

0.5 预备知识与参考书目

阅读和学习本书之前，需要读者了解如何运用 C/C++ 语言编程，并具有计算机图形学方面的初步知识(如几何学，三角法，线性代数，微积分学及微分几何学)。当然，计算机图形技术的经验很少甚至没有的人，仍能够参照使用本书中大部分内容。计算机图形学是一个非常丰富的领域，为丰富学习经验，读者可以参阅国内外相关的参考资料，如计算机图形学，VISUAL C++ 编程，OpenGL 程序设计等。

第一部分　编程指南

1　OpenGL 入门

图形硬件的软件界面，OpenGL 的主要目的是在帧缓存中渲染二维和三维物体。而这些物体是由定义了几何物体的顶点序列或定义了图像的像素点来描述的。OpenGL 对这些数据通过几步处理，将其变换成帧缓存中最终图像的像素点。

本章系统的介绍了 OpenGL，概览了 OpenGL 如何进行工作。主要由以下几部分组成：

- OpenGL 概览：解释何为 OpenGL，它能实现那些功能，并介绍了 OpenGL 的执行机制。
- 一个简单的 OpenGL 例程：介绍了一个 OpenGL 的小程序，对其作了简单的讨论。在这一节还给出了一些基本的计算机图形学术语的定义。
- 基本的 OpenGL 操作：给出了关于 OpenGL 怎样处理数据，并在帧缓存中产生相应图像的高层描述。
- OpenGL 的命令语法：阐述了 OpenGL 命令所遵循的约定和符号。
- 作为状态机的 OpenGL：描述了 OpenGL 中状态变量的用法以及用于查询、激活和终止状态的命令。
- OpenGL 相关函数库：简要描述 OpenGL 相关的库函数集。
- 动画：以通用的术语解释了怎样在屏幕上制作运动的图像——动画。

1.1　OpenGL 概览

OpenGL 是图形硬件的软件接口。OpenGL 中包括大约 120 个不同的命令，程序员可以使用这些命令设定所需的物体和操作，来制作交互式的三维应用程序。

有时，图形的创建和显示并不是在一台计算机上完成，这种情况下 OpenGL 也能有效工作。比如在计算机网络环境下，运行程序和发出 OpenGL 绘图命令的计算机叫做“客户”，接收并执行绘图命令的计算机叫做“服务器”。由于从“客户”到“服务器”发出的命令（称为协议）总是相同的，这种 OpenGL 的网络透明性使得 OpenGL 程序在网络上也能够得以顺利运行。

OpenGL 是作为一种新型的接口来设计的，其与硬件无关的特性使它可以在不同的硬件平台上实现。也是为了实现这一跨平台性能，OpenGL 没有包括执行窗口任务或获取用户输入的命令，程序员必须通过所使用的窗口系统来控制正在使用的特定硬件。作为底层的支持平台，OpenGL 也不提供描述三维物体模型的高级命令（如绘制汽车、身体的一部分、飞机或者分子模型等）。使用 OpenGL 的程序员必须利用由一系列简单的点、直线和多边形等几何图元的组合来建立期望的模型，也可以在 OpenGL 的顶层建立提供这些高

级特性的较为复杂的函数库(如"Open Inventor",参见"OpenGL 相关函数库"一节)。换一句话说,OpenGL 并不是专长于建立模型,它主要的优势在于渲染。这也就是为什么我们要在本书中专辟一章讨论了的关于建模平台和渲染平台的接口的问题(参见 15 章)。

现在读者已经了解了 OpenGL 的几点局限,那么 OpenGL 到底能作些什么呢? 为了使读者有一个感性的认识,我们参照附录中 OpenGL 制作的彩图来对 OpenGL 的功能进行一个初步的了解。

图附 10-1 显示了整个场景的线框模型图。网格的每一条线对应于图元(一般是多边形)的边,比如:桌子的表面由三角形组成的,好象被切开的一整块蛋糕。需要注意的一点是,如果物体是实心的,而不是线框组成的,物体将有一部分被遮挡起来看不到。所以尽管实际上山的大部分都被墙壁遮住了,此时仍能够看到窗外整个山的模型。球是由数百个彩色分块组成的,观察者看到的是所有分块的边缘线框,包括球后面各个分块,所以整个球体看起来已经接近于实体。任何复杂的物体,都可以应用这种简单物体的组合的方式来创建。

图附 10-2 例示了同一线框场景,但其中加入了深度提示(depth-cue)。读者可以注意到,距离人眼较远的线较为昏暗,这样一来给出了更加接近现实生活的生动的深度提示。

图附 10-3 例示了线框场景的另一种形式,在这里进行了锯齿现象的消除。锯齿现象的消除这一技术应用于减少锯齿现象对图像逼真度的破坏(当边缘附近的像素只有一部分属于该图像时,就会产生锯齿现象。这种锯齿现象在近水平或近垂直直线上常表现得尤其明显。)

图附 10-4 例示了具有无明暗差别的阴影(又称平影)的场景形式。场景中的物体显示为单色的实心物体。这些物体整个表面显得"无明暗差别"(指其对室内的光照条件并无反应),因此,它们看上去不很平滑。

图附 10-5 例示了具有光照及平滑阴影的场景。读者可以注意到,当加在物体上的阴影对室内的光源作出响应时,物体表面看上去非常平滑,场景看起来便更加真实,面且具有三维立体感。

图附 10-6 例示了在前面场景中上加阴影和纹理后的效果。阴影特性在 OpenGL 中并没有明确的定义(OpenGL 没有"阴影命令")。但是程序员可以应用第 14 章所介绍的技术来自己创建。纹理映射的操作使程序员能够把二维纹理应用到三维物体上。这个场景中,桌面就是纹理映射的一个最生动的例子。墙壁,地板,桌面,桌子的上方,都应用了纹理映射。

图附 10-7 例示了场景中经过运动模糊化处理的物体。这一技术使物体向前运动时,看起来就好象是被什么抓住了似的,画面上留下了物体运动的模糊轨迹。

图附 10-8 例示的是以不同的视点画出的场景。

接下来的两幅彩色图像阐释了应用 OpenGL 能够实现的更为复杂性视觉效果。

图附 10-9 例示了大气效应的使用(通常指雾),图中显示出空中存在着微粒。

图附 10-10 例示了视野效应,模拟出照相机一种特性——镜头不可能使所有照片场景中的物体处在焦距处。照相机只能对场景中的某一特定点聚焦,比该点近或远的物体,

看起来都显得有些模糊。

通过对具体图形的阐释，想必读者对利用 OpenGL 图形系统可以实现的各种效果已经大体上有了一个感性的认识。下面简要描述一下在渲染屏幕图像时，需要 OpenGL 执行的主要图形操作所遵循的次序：

① 以几何图元构造表面，从而对物体进行数学描述（在 OpenGL 中，点、直线、多边形、图形、位图都被看成是图元）；

② 在三维空间布置物体，选择用于观察组合场景的最佳观察点；

③ 计算各物体的颜色。在应用程序中可以明确地分配颜色：颜色可以通过指定光照条件来确定，也可以通过在物体上粘贴纹理来获得；

④ 将物体的数学描述和相关的颜色信息变换成屏幕像素，这一过程称为光栅化。在这一步，可以进行其它 OpenGL 操作，如：消除物体后的隐藏部分（隐藏部分不画，这样可以提高性能）。另外，在场景进行了光栅化之后，但是还没有绘制到屏幕上时，程序员可以按照自己的意愿对像素数据进行处理。在附录 1 中，对操作顺序进行了更为详尽的阐述。

1.2 简单的 OpenGL 程序举例

应用 OpenGL 图形系统来完成的事情是十分庞杂的，所以有时一个 OpenGL 程序可能相当复杂。但是对于许多大程序其基本结构是十分简单的——比如程序的任务就是初始化某些状态来控制 OpenGL 的渲染，并且指定要渲染的物体。所以在程序编制之前，弄清程序结构是十分有益的。

在研究 OpenGL 例程之前，读者需要再熟悉几个术语。

渲染，指计算机根据模型创建图像的过程；这些模型或物体，是由几何图元——点、直线、多边形等——构成的；图元可以绘制为几种不同模式。对每一种模式可以进行独立的控制（即设定一种模式不会影响其它模式的设置，最终帧缓存的结果是由模式的共同作用来决定的）。通过函数调用的形式来发送命令，可以指定图元，设置模式，或进行其它 OpenGL操作。

这些几何图元通过指定其顶点加以确定。数据（包括顶点坐标，颜色，法线，纹理坐标和边缘标志）与顶点相关联，每个顶点和其相关的数据都单独的、依次的、以同样的方式进行处理。这一准则有一个例外：为使图元尺寸适合某个特定区域而必须裁剪掉一些顶点时，顶点数据必须进行修改并产生新的顶点。裁剪的类型取决于顶点表示的是什么图元。尽管在命令作用前可能有一个不确定的延迟，总的来说，命令的处理是按照它们发送的顺序进行的。这意味着在后面的命令起作用前，能够完整的画出每个图元。这也意味着状态查询命令所返回的数据，是与前面发送的所有 OpenGL 命令的执行结果相一致的。

最终渲染的图像由屏幕上绘制的像素组成；像素（pixel，图像元素 picture element 的缩写）是由显示硬件能够输出到屏幕上的最小的可视元素。像素信息（比如像素绘制所采用的颜色）是在系统内存中组织为位平面。位平面是一块内存区域，它为屏幕上的每一个像素保存了一位的信息；比如，这一字节信息可能代表像素的颜色红的程度。位平面本身组成了帧缓存，帧缓存中包含了图形显示需要的来控制屏幕所有像素亮度的信息。

OpenGL提供了相当直接的对二维或三维图形的基本操作控制，包括设定变换矩阵、光照方程系数、锯齿现象的消除方法和像素更新等操作的参数。然而，它并没有提供描述复杂几何体的方法。因此，发送的 OpenGL 命令指定了如何得到某种结果（遵循何种过程），而不是描述这种结果看起来象什么。这就是说，OpenGL 从本质上是程序性的而不是描述性的。正是这种程序性帮助我们了解 OpenGL 怎样工作（如执行操作的次序），从而能够更好地使用它。

现在让我们看一个 OpenGL 例程的基本框架。

例 1－1 是在黑色背景上对一个白色矩形进行渲染，其结果如图 1－1 所示。

例 1－1 一个简单的 OpenGL 例程主框架。

```
# include < whateverYouNeed.h >

main( )
{
   OpenAWindowPlease( );
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho( - 1.0, 1.0, - 1.0, 1.0, - 1.0, 1.0);
   glBegin(GL_POLYGON);
   glVertex2f( - 0.5, - 0.5);
   glVertex2f( - 0.5, 0.5);
   glVertex2f(0.5, 0.5);
   glVertex2f(0.5, - 0.5);
   glEnd( );
   glFlush( );
   KeepTheWindowOnTheScreenForAWhile( );
}
```

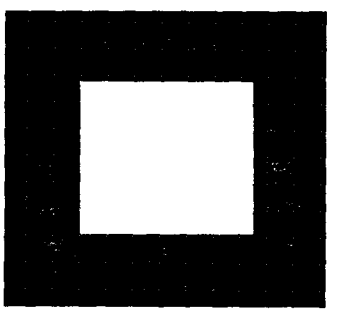

图 1－1 黑色背景下的白色矩形

主程序第一行是在屏幕上打开一个窗口：函数 OpenAWindowPlease( )是窗口系统指定子程序的占位符。接下来的两行 OpenGL 命令将窗口清为黑屏：glClearColor( )命令设定窗口清空颜色；而 glClear( )实际进行清空操作。清空颜色一旦设置，无论何时调用 glClear( )命令，窗口都将被清为该颜色。类似地，glClearColor( )命令用于确定以何种颜色绘制物体——在本例中，绘图颜色为白色。从这一条语句开始，直到另一次颜色设置函数调用为止，绘制的所有物体都使用这种颜色。

在这个程序中使用的下一条命令为 glOrtho( )，该命令用来指定 OpenGL 用以绘制最终图像的坐标系，同时指定了图像如何映射到屏幕上。接下来的调用都括在 glBegin( )和 glEnd( )这对命令中间，定义了要绘制的物体——在本例中，要绘制的物体是一个有 4 个顶点的多边形。多边形的“角”由 glVertex2f( )命令给出定义。根据参数，即(x,y)坐标组，可以判断出这个多边形是矩形。

最后，glFlush( )保证了绘图命令能够得以实际执行，而不是存储在缓冲区中等待另

外的 OpenGL 命令。占位程序 ForAWhile( )强迫画面保留在屏幕上,而不是马上消失。

完整的程序请见例 1-2。

1.3 基本 OpenGL 操作

下图给出了 OpenGL 处理数据的高层模块简要框图。在图 1-2 中,函数从左侧进入,通过一系列类似管道的处理过程。一些命令指定要绘制的几何物体,另一些在不同的操作阶段控制对物体的处理。

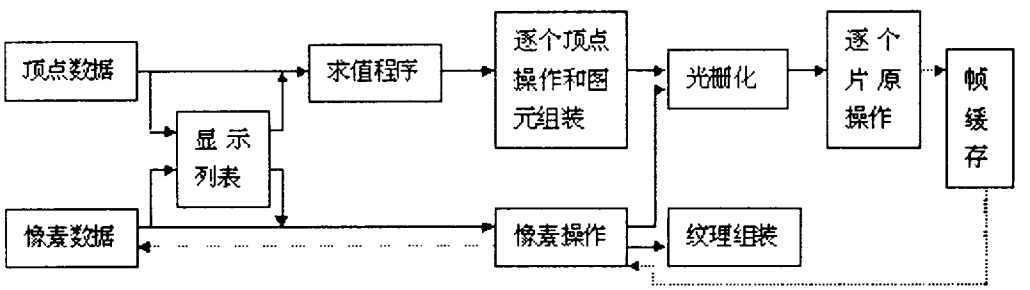

图 1-2 OpenGL 模块框图

如同图中第一个模块所示,OpenGL 并不让每个函数立即通过处理管道,而是把它们累积在一个显示列表中,稍后一次性进行处理。

处理管道的求值阶段,通过输入数据的求值多项式,提供了一种高效的方法来近似绘制几何曲线和曲面。下一阶段进行对每个顶点的操作和图元组合。对顶点进行变换和光照处理,对图元进行裁剪以适合视区大小,这样便为下一阶段的处理做好了准备。

在光栅操作中,通过点、线段、多边形的二维描述,产生一系列的帧缓存地址和相关数值。这样产生的每一片原都传入最后的处理阶段——"逐个片原操作",对数据进行最后的处理,然后将它们作为像素存储在帧缓存中。这些操作包括:基于输入和先前储存的 z 值对帧缓存(z 缓存)进行有条件的更新,混合输入像素颜色与存储颜色、屏蔽等其它对像素值的逻辑操作。

输入数据也可以是像素形式,而不是顶点形式。这类数据(可以描述用于纹理映射中的一幅图像)将跳过上述处理的第一阶段——"求值程序",作为像素值直接在像素操作阶段进行处理。这一阶段的处理结果,或者存储为纹理内存以用于光栅操作阶段,或者采取同几何数据一样的形式进行光栅化,并将结果片原融合入帧缓存中。

所有 OpenGL 状态的要素,包括纹理内存的内容表,甚至帧缓存,都可以在 OpenGL 管道运行过程中获得。

更详细的内容读者可以进一步参考"库函数详解"中的相关内容。

1.4 OpenGL 的命令语法

从前一节的简单例程中,读者可能已经观察到,所有 OpenGL 的命令,都使用前缀"gl"和词首字母大写的单词共同组成命令名(如 glClearColor( ))。类似的,OpenGL 定义的常量

中,都以 GL_ 开头并且所有的字母都大写,单词之间以下划线来分隔(如 GL_COLOR_BUFFER_BIT)。

读者可能注意到了在某些命令后面,附加一些看起来不相关的字符(比如,glColor3f()中的后缀 3f)。确实,作为设置当前颜色的命令的定义,该命令名中的 Color 部分就已经足够了。然而,为了使得程序员可以使用不同类型的参数,定义的这类命令不止是一个。详细的说,后缀中的 3 表示给出的参数为 3 个; Color 命令的另一种形式带有 4 个参数。后缀中的 f 表示参数为浮点型数值。某些 OpenGL 命令接受多至 8 种不同的数据类型作为它们的参数。表 1-1 列出了对于 OpenGL 的 ANSI C 实现,用作后缀以指定参数数据类型的字母,以及相应的 OpenGL 数据类型定义。程序员所采用的 OpneGL 特定实现不一定严格遵循这种方案;比如:OpenGL 利用 C++ 或 Ada 的实现,就未必遵循这种方案。

表 1-1　命令后缀参数数据类型

后缀定义	数据类型	相应的 C 语音类型	OpenGL 类型定义
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	float	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	326-bit unsigned integer	unsigned long	GLushort, GLenum, Glibtfield

因此,下面两个命令:

```
glVertex2i(1, 3);
glVertex2f(1.0, 3.0);
```

是等价的,除了第一条命令以 32 位整型数指定顶点坐标,而第二条命令则是以单精度浮点数来指定。

有一些 OpenGL 命令的最后面带有一个字母 v,这表示该命令带有的是一个指向矢量(或数组)值的指针参数,面不是一系列单独的参数。许多命令都有矢量和非矢量两种参数形式,但是一些命令只接受独立参数,而另一些命令则要求至少某些参数要指定为矢量。下面几行命令例示了设置当前颜色命令的矢量与非矢量形式。

```
glColor3f(1.0, 0.0, 0.0);
float color_array[] = {1.0, 0.0, 0.0};
glColor3fv(color_array);
```

在本书的其余部分(实际的源代码例程除外),仅以其基本命名表示 OpenGL 命令,面在末尾包括进来一个“*”号表示命名中的其它部分。比如,glColor*( )代表设置当前颜色命令的所有变化形式。如果我们希望对某一特定命令的一种形式做一特殊的说明,则将定义该形式的必要后缀包括进去。比如:glVertex*v( )命令代表的是用于指定顶点的命令的所有矢量形式。

最后，OpenGL 定义了常量 Glvoid；如果用 C 语言编写程序，读者可以使用该常量来代替 void。

1.5 作为状态机的 OpenGL

OpenGL 是一个状态机。这就是说，将 OpenGL 设置为各种状态（或各种模式）后，这些状态将一直有效地保存在 OpenGL 中，直到改变了这些状态的值为止。正如读者已经看到的那样，当前颜色就是一个状态变量。可以将当前颜色设置为白色、红色或其它任意一种颜色，接下来所有的物体都将以该颜色画出，直到程序员改变了当前颜色设置为其它的某种颜色为止。当前颜色仅仅是 OpenGL 保存的多种状态变量之一。其它的状态变量控制另外的状态，如当前视图和投影变换、直线和多边形点划模式、多边形绘画模式、像素包装协议、灯光的位置和特征、被画物体的材料属性等。许多状态变量都指的是可以使用命令 glEnable( )或 glDisable( )来激活或取消的模式。

每一个状态变量或模式都有一个缺省值，程序员可以在程序中的任意位置对系统进行查询，以获得每一个变量的当前值。一般地，程序员可以使用下列四个命令之一来完成查询操作：gLGetBooleanv( )，glGetDoublev( )，glGetFloatv( )或 glGetIntegerv( )。采用哪一个命令取决于程序员期望以何种数据类型给出查询结果。某些状态变量有更为特别确定的查询命令（如 glGetLight * ( )，glGetError( )或 glGetPolygonStipple( )）.

另外，程序员可以用 glPushAttrib( ) 和 glPopAttrib( )命令将有关属性堆栈的状态变量集的值保存起来，可以随时进行恢复。一般情况下，由于用 glPushAttrib( ) 和 glPopAttrib( )更为有效，要尽量使用这些查询命令而不是其它的任意命令。

可供查询的状态变量的完整列表见附 2。对每一个变量，该附录还列出了返回变量值的 glGet * ( )命令，所属属性类，以及变量的缺省值。

1.6 OpenGL 相关函数库

OpenGL 提供了功能强大、操作简单的一套渲染命令，所有的高级绘图必须通过这些命令来完成。因此，为了简化特定的程序任务的编写与执行，程序员可以在 OpenGL 的顶层编写自己的函数库。同样，程序员还可以编写一些子程序，来使 OpenGL 程序更具有针对性，容易在自己的窗口系统中得以运行。事实上，这类可以提供特殊特性的函数库和子程序，有一些已经编制出来了。这些库在下面将进行初步的介绍。

1.6.1 OpenGL 相关函数库简介

OpenGL 实用库（GLU）中的函数通过底层 OpenGL 命令来完成这样一些任务，如为特定的观察方位和投影设置矩阵，完成多边形的网格化，以及渲染表面等等。这个库是作为 OpenGL 实现工具的一部分提供的。在附 3 及库函数详解中对其进行了详细的描述。其它的一些更为实用的 GLU 子函数将在本书的相关章节进行讨论。GLU 子程序使用前缀 glu 来标志。

OpenGL 对 X 窗口系统的扩展库（GLX）提供了创建 OpenGL 的内容表，并使其与计算机（该计算机使用 X 窗口系统）上的可画窗口相联系的一种手段。GLX 是 OpenGL 的一个附件。在附 4 和库函数详细解中对其进行了更详细的阐述。GLX 库中的一个子程序（用

于交换帧缓存)将在动画一节中进行描述。GLX 库函数使用的前缀为 glX。

OpenGL 辅助库将在附 5 及库函数详细解中进行详细的讨论。辅助库子程序使用的前缀为 aux。

Open Inventor 是面向对象的编程工具包,它基于 OpenGL,提供了用于创建交互式三维图形应用程序的物体和方法。Open Inventor 也是由 SGI 公司(Silicon Graphics Inc.)开发的,用 C++ 进行编写,提供了预先创建的物体和用于用户交互的嵌入式事件模型,同时提供了用于创建及编辑三维场景的高级应用组件,以及进行物体打印、与其它图形格式交换数据的能力。

1.6.2 OpenGL 辅助库(GLAUX)

尽管 OpenGL 包括渲染命令,但却是独立于与任何窗口系统或操作系统而设计出来的。因此,OpenGL 并不包括用来打开窗口以及从键盘或鼠标读取事件的命令。只要编写一个完整的图形程序,就必须打开一个窗口,要不然无法进行任何操作。许多程序都要求一些用户输入,或者需要操作系统或窗口系统提供其它的一些服务。OpenGL 辅助库便可以用来简化诸如打开窗口、检测输入等等操作。

由于 OpenGL 的绘图命令仅限于生成简单的几何图元(点,直线,和多边形),因此,辅助库包括了一些用于创建更复杂的三维物体,如圆球、圆环及茶壶等物体的子程序。这样,不但使程序的编制变得轻松,而且还将增加不少乐趣。OpenGL 发展至今,辅助库已经作为其一个部分,在几乎所有的编译平台中都集成了进来,读者已不必担心找不到 GLAUX 而在复杂物体绘制上大伤脑筋了。

辅助库的设计者们编制的这些函数十分简单易用,本书中的例程都是建立在辅助库的基础之上的。虽然以它为基础实现大的应用程序仍然是相当困难的,但是从学习的角度,将是一个良好的开端。本节的剩余部分简要地描述了这个辅助库,以便于读者跟着本书后面的例程开始学习。关于这些程序的详细说明请参见附 5 与库函数详解。

1. 窗口管理

为了初始化并打开一个窗口,需要调用三个函数完成必要的任务。

1) auxInitWindow( )命令,用来在屏幕上打开一窗口,窗口名称在函数参数中给出。Esc 键将被激活,以随时终止程序。同时将窗口背景颜色设置为缺省黑色。

2) auxInitPosition( )命令,通知 auxInitWindow( )命令应该把窗口放置在屏幕上什么位置。

3) auxInitDisplayMode( )命令,通知 auxInitWindow( )命令是创建 RGBA 模式窗口还是创建颜色索引模式窗口。程序员还可以指定一个单缓存或双缓存窗口。(如果在颜色索引模式下工作,有时要将某种颜色加载到颜色表中;这一任务可以使用 auxSetOneColor( )命令来完成。)也可以使用该程序来指示出程序员希望该窗口具有的相关深度、模板、和/或累积缓存。

2. 处理输入事件

程序员可以利用下列函数来注册回调函数,这一回调函数将在指定事件发生时加以激活。

1) auxReshapeFunc( )命令,表示在窗口尺寸改变、移动或暴露时应该执行的动作;

2）auxKeyFunc( )命令和 auxMouseFunc( )命令。在键盘或鼠标被压下或被释放时，这两个命令将键盘键值或鼠标按键与调用的程序联结起来。

3. 绘制三维物体

OpenGL 辅助库包括下列一些绘制三维物体的命令。

球体(sphere)	八面体(octahedron)
立方体(cube)	十二面体(dodecahedron)
圆环面(torus)	二十面体(icosahedron)
圆柱体(cylinder)	茶壶(teapot)
圆锥体(cone)	

程序员可以以线框或是定义了表面法线的有阴影效果的实体这两种不同形式来绘制物体，例如，下面是绘制圆球和圆环的函数。

```
void auxWireSphere(GLdouble radius);
void auxSolidSphere(GLdouble radius);
void auxWireTorus(GLdouble innerRadius, GLdouble outerRadius);
void auxSolidTorus(GLdouble innerRadius, GLdouble outerRadius);
```

绘制的所有模型都以原点为中心。当以单位比例因子进行绘制时，这些模型都内接于坐标从 -1 到 1 的正方体。通过对函数参数的调整可以对物体进行缩放。

4. 管理后台进程

程序员可以用函数 auxIdleFunc( )指定一个函数，在没有其它事件未决(pending)的时候——比如当事件循环就要空闲时，便执行这个函数。该函数以一个指针指向后台函数作为其唯一的参数。传递参数 0 时将取消函数的执行。

5. 运行程序

在主程序 main( )中，调用 auxMainLoop( )命令，并把重画场景物体的函数名作为参数传递给该命令。例程 1-2 例示了如何使用辅助库创建图 1-1 所示的简单程序，此处是在白色的屏幕上绘制一个蓝色的矩形。

例 1-2 利用辅助库编写一个简单的 OpenGL 程序：simple.c

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glaux.h>
void main( )
{
    // OpenAWindowPlease( );
    auxInitDisplayMode (AUX_SINGLE | AUX_RGBA);
    auxInitPosition (0, 0, 500, 500);
    auxInitWindow ("编程指南例程 1-1");

    glClearColor(1.0, 1.0, 1.0, 0.0); // 指定清空颜色
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glColor3f(0.0, 0.0, 1.0);      // 指定绘制物体颜色
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
                                          // 正交投影模式
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);   // 此处可以用 glColor3f 设置
        glVertex2f(-0.5, 0.5);    // 每个顶点的颜色生成一个
        glVertex2f(0.5, 0.5);     // 五颜六色的多边形。
        glVertex2f(0.5, -0.5);

    glEnd( );
    glFlush( );
// KeepTheWindowOn-
// TheScreenForAWhile( );
    _sleep(2000);
}
```

运行结果如图 1-1。

1.7 动　　画

在图形计算机上,运动画面的绘制可能是最令人感兴趣的事情。无论对于一位试图察看所设计的机械零件所有侧面的工程师,还是利用模拟器学习飞机驾驶的飞行员,或者仅仅是一位计算机游戏迷,动画都是人们头脑中的计算机图形学的一个重要的组成部分。

在电影院里,通过以一定的速度(每秒 24 帧)放映一系列的画面,同时将其以每秒 24 帧的速度投影到屏幕上,就可以得到运动的画面。当每一帧画面移到镜头后面时,快门打开,这一帧画面就显示出来。胶片向下一帧移动时,快门立即关闭,到了画面处快门再次打开将该画面显示出来,如此反复。尽管观众每秒钟看到的是 24 帧不同的画面,但是大脑中的视觉暂留将其混合成一段连贯的动画。早期的查理　卓别林的电影中,以每秒钟 16 帧进行放映,这使画面看起来有很明显的闪烁跳动。多数的现代放映机,常常以每秒 48 帧的速度将每帧画面显示两次,这样一来就大大减少了闪烁。图形计算机的屏幕刷新(重新绘制画面)的频率一般大约是每秒钟 60 到 76 次,有的甚至以每秒 120 次的频率进行刷新。很明显,60 赫兹比 30 赫兹更连贯,而 120 赫兹比 60 赫兹的效果更佳。然而,超过 120 赫兹的刷新频率已经超过了衰减返回点——人的眼睛的性能仅此而已。

制作运动画面投影的关键思想是:在显示每一帧画面时,该画面应是完整的。如果用类似下面的程序制作有百万帧画面的计算机动画。

```
open_window( );
for(i = 0; i < 1000000; i++)
   clear_the_window( );
   draw_frame(i);
   wait_until_a_24th_of_a_second_is_over( );
}
```

由于系统清屏和绘制关键帧画面都需要时间，而在这里未加考虑，所以这个程序画出的将是有闪动的画面，闪动的程度取决于清屏和绘图所花的时间与 1/24 秒的接近程度。假定作图所花的时间与 1/24 秒几乎接近，那么最先绘制的各物体在 1/24 秒内是可见的，并且在屏幕上显示为一幅实体图像；当程序运行到下一帧时，先前所画的各个物体立即被清除了，因此这样显现的图像是非常可怕的——在这个 1/24 秒中的大部分时间，眼睛观察到的是清除后的背景，后面需要画出的那些物体正在缓存内进行绘制。问题在于该程序没有把绘制的关键帧画面显示完整，而是边绘制时边进行观察。

解决上述问题的一个简单的途径是提供双缓存——提供两个完整的颜色缓存的硬件或软件。当一个缓存用来作图时，另一个用来显示。当画完一帧时，交换两个缓存。这样，刚才用来显示的缓存现在用来作图。这就象一个只以两帧进行循环的电影放映机，当一帧画面被投影到屏幕上时，有一位画家要迅速地擦掉这幅画面并以重画另一帧。只要画家的动作足够的快，观众就注意不到这套设备与只是将事先已经画好的一帧一帧画面简单地进行播放的设有什么差别。这在实际中当然是不可能做到的，然而计算机可以。应用双缓存，每一帧画面只在绘制完成之后才显示出来，因此观察者不会看到只绘制了一部分的画面。

将前面的程序稍作修改，便可以连贯性地显示动画图形。程序如下：

```
open_window_in_double_buffer_mode( );
for (i = 0; i < 1000000; i++)
   clear_the_window( );
   draw_frame(i);
   swap_the_buffers( );
}
```

除了简单地在视图缓存和绘图缓存之间进行切换以外，为了保证前而的缓存得以完整的显示，程序中的 swap_the_buffers( )函数将一直等到当前屏幕更新期结束才进行循环的下一步。该函数还允许从头开始完整地显示一个新的缓存。假设一个系统每秒更新 60 次，这就意味着以最快的速度，在一秒内最多可以显示 60 帧画面，如果两个帧缓存都可以在 1/60 秒之内清除和画出，那么动画就能够以该速度平滑地运行。

在这样的系统中经常发生这种情况：画而太复杂，在 1/60 秒内无法画出，这样，每一帧的显示将超时。另外，如果绘制一帧要 1/45 秒，而每秒只放映 30 帧，这样每放映一帧画面就出现了 1/30 - 1/45 = 1/90 秒的空闲时间。尽管浪费 1/90 秒是很短的时间，但是每 1/30 秒就浪费了 1/90 秒，这意味着实际上有三分之一的时间被浪费掉了。

另外，视频的更新速度是不变的，这就可能产生一些意想不到的运行结果。比如，以 1/60 秒的显示器刷新速度以及恒定的帧速，程序员能够以 60,30,20,15,12 帧/秒等(60/1,60/2,60/3,60/4,60/5……)速度运行程序。这意味着，如果程序员正在编写一个应用程序，并逐渐加入某些特性(比如一个飞行模拟器，正在增加地面场景)，最开始所增加的特性对整体性能都没有什么影响——仍旧可以获得 60 帧/秒的速度。但是突然，在又加入一个新的特性后，由于系统已经不能在 1/60 秒内绘制完整幅画面，这样就错过了第一次可能进行缓存交换的时间，只好等到下一次进行，应用程序的性能将降低一半——每秒显