国外经典教材·计算机科学与技术

PEARSON
Addison Wesley

OpenGL: A Primer
Second Edition

OpenGL程序设计指南
(第2版)

(美) Edward Angel 著
李桂琼 张文祥 译

Pearson Education

清华大学出版社

国外经典教材•计算机科学与技术

OpenGL 程序设计指南

（第 2 版）

（美） Edward Angel 著

李桂琼 张文祥 译

清华大学出版社

北 京

内 容 简 介

本书简明扼要，通过 11 章的篇幅循序渐进地介绍了 OpenGL 程序设计。它可用作计算机图形学教材的配套教参，也可用供从事图形开发工作的程序员参考。

图书在版编目(CIP)数据

OpenGL 程序设计指南：第 2 版=OpenGL：A Primer，Second Edition /(美)安杰尔(Angel E.)著；李桂琼，张文祥译. —北京：清华大学出版社，2005.5
(国外经典教材·计算机科学与技术)
ISBN 7-302-10889-7
Ⅰ.O… Ⅱ.①安… ②李…③张… Ⅲ.图形软件，OpenGL—高等学校—教材 Ⅳ.TP391.41

中国版本图书馆 CIP 数据核字(2005)第 037978 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
文稿编辑：文开棋
封面设计：久久度文化
印 刷 者：北京季蜂印刷有限公司
装 订 者：北京鑫海金澳胶印有限公司
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：12.5　**字数**：299 千字
版　　次：2005 年 5 月第 1 版　2005 年 5 月第 1 次印刷
书　　号：ISBN 7-302-10889-7/TP·7243
印　　数：1～3500
定　　价：26.00 元

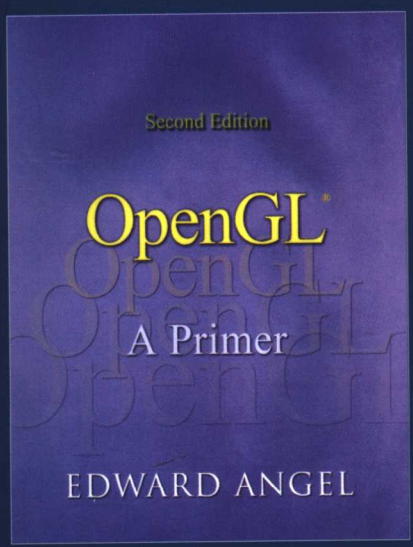
(美) Edward Angel 著

作者简介

Edward Angel是新墨西哥大学计算机科学、计算机工程和媒体艺术系的教授，该校Fine Arts学院艺术技术中心主任。他毕业于加州技术学院，并获得该校工学士学位。1968年在南加州大学获得博士学位。目前，他的主要研究领域是计算机图形和科学可视化。

他曾创作了一本非常畅销的教材“*Interactive Computer Graphics: A Top-Down Approach Using OpenGL*”。

Angel教授是Digital Pueblo Project的首席研究员。这是美国国家科学基金会赞助的项目，主要是把艺术和技术结合起来，通过合作性图像和动画项目，促进新墨西哥州各个区的经济发展。



出 版 说 明

近年来，我国的高等教育特别是计算机学科教育，进行了一系列大的调整和改革，急需一批门类齐全、具有国际先进水平的计算机经典教材，以适应当前我国计算机科学的教学需要。通过使用国外先进的经典教材，可以了解并吸收国际先进的教学思想和教学方法，使我国的计算机科学教育能够跟上国际计算机教育发展的步伐，从而培育出更多具有国际水准的计算机专业人才，增强我国计算机产业的核心竞争力。为此，我们从国外知名的出版集团 Pearson 引进这套“国外经典教材·计算机科学与技术”。

作为全球最大的图书出版机构，Pearson 在高等教育领域有着不凡的表现，其下属的 Prentice Hall 和 Addison Wesley 出版社是全球计算机高等教育的龙头出版机构。清华大学出版社与 Pearson 出版集团长期保持着紧密友好的合作关系，这次引进的“国外经典教材·计算机科学与技术”教材大部分出自 Prentice Hall 和 Addison Wesley 两家出版社。为了组织该套教材的出版，我们在国内聘请了一批知名的专家和教授，成立了一个专门的教材编审委员会。

教材编审委员会的运作从教材的选题阶段即开始启动，各位委员根据国内外高等院校计算机科学及相关专业的现有课程体系，并结合各个专业的培养方向，从 Pearson 出版的计算机系列教材中精心挑选针对性强的题材，以保证该套教材的优秀性和领先性，避免出现“低质重复引进”或“高质消化不良”的现象。

为了保证出版质量，我们为该套教材配备了一批经验丰富的编辑、排版、校对人员，制定了更加严格的出版流程。本套教材的译者，全部来自于对应专业的高校教师或拥有相关经验的 IT 专家。每本教材的责编在翻译伊始，就定期不间断地与该书的译者进行交流与反馈。为了尽可能地保留与发扬教材原著的精华，在经过翻译、排版和传统的三审三校之后，我们还请编审委员或相关的专家教授对文稿进行审读，以最大程度地弥补和修正在前面一系列加工过程中对教材造成的误差和瑕疵。

由于时间紧迫和受全体制作人员自身能力所限，该套教材在出版过程中很可能还存在一些遗憾，欢迎广大师生来电来信批评指正。同时，也欢迎读者朋友积极向我们推荐各类优秀的国外计算机教材，共同为我国高等院校计算机教育事业贡献力量。

清华大学出版社

国外经典教材·计算机科学与技术

编 审 委 员 会

前　言

2000年，我在写本书的第1版时，目标是让它用作我的计算机图形学教材 *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*（第3版）（Addison-Wesley于 2003年出版）的配套教参，但同时又希望本书能自成一体，为刚接触OpenGL的程序员（他们已经掌握了一些计算机图形知识，并希望使用 OpenGL）提供一本入门读物。对于第二组读者，我写作本书的宗旨和以前一样，并不是想让本书取代OpenGL的标准参考书，也就是分别称为“红皮书”和“蓝皮书”的两本权威 OpenGL 书籍——前者是指 *OpenGL Programming Guide*(第 4 版)，Addison-Wesley 出版，后者是指 *OpenGL 1.4 Reference Manual*(第 4 版)，Addison-Wesley 出版。我的目的是让应用程序的开发者在不必参考这些书的前提下，就能着手编写 OpenGL 应用程序。

我认为本书第1版已经达到了那些目标。而且根据反馈，我惊讶地发现第二组读者的人数要比我预计的多得多。许多学校甚至将本书第1版作为很多层次的图形课程教材。

构思第2版时，我必须做出一些艰难的决策，考虑如何对这一本我认为已经非常成功的书籍进行修改。我希望本书保持短小精悍的风格，尽可能降低它的售价，同时又要根据用户的反馈意见，更深入地讲解图形和 OpenGL API。作为对这种有点儿矛盾的要求的一个回应，本书第2版的篇幅只增加了约20%。我添加了一些例子，新增了一章的篇幅来介绍“可编程图形管道”，它代表计算机图形领域的一个重大进步。另外，本书明确描述了在过去12年间，由OpenGL ARB批准的OpenGL增补内容。当然，我还对某些表述方式进行了澄清。

本书采取自顶向下的方式来讲授计算机图形学，这种教学方法是我在 *Interactive Computer Graphics* 一书中提出的。它的基本原理在于，如果能尽可能快地编写出实际有用的应用程序，那么学生们在学习现代计算机图形的时候，就能取得最好的效果。OpenGL API也适合采取这种方式来教学。全世界的许多大学都在采用这种方法，这标志着它已经取得了极大的成功。

用过 *Interactive Computer Graphics* 这本书的学生都意识到，虽然它频繁地运用了OpenGL，但却从未将自己定位成一本 OpenGL 编程指南或者一本用户手册。结果就是教科书中有关OpenGL的知识并不完整。不仅没有覆盖所有OpenGL函数，而且还没有为函数及其参数提供一个详细的列表。对于学生来说，前者通常都不成问题，但后者确实带来诸多不便。因此，本书的宗旨就是满足那一部分的需求，同时不要求学生购买虽然很有参考价值、但却非常昂贵的 *OpenGL 1.4 Reference Manual*(第 4 版)，即蓝皮书。

大学里修图形课程的学生只占整个图形开发社区的一小部分。相较于整个编程社区，所占的比例更是小得可怜。但是，几乎所有人都多多少少对计算机图形以及图形程序的编写产生过短暂的兴趣。对于这些人来说，OpenGL 是涉足图形设计的一个非常有吸引力的途径。本书的第二个写作动机是提供一种简单的方法来掌握 OpenGL。就我个人的观点来

看，即使最终要在 Windows 平台上以 DirectX 为生的程序员，如果能从 OpenGL 开始，并经由它来掌握最基本的概念，那么以后的学习过程也会简单得多。如果需要编写科学应用程序，而且要在一个跨平台的环境中工作，那么 OpenGL 几乎是惟一可供选择的 API。

本书几乎没有涉及在我的教材中提到的任何数学理论。例如，第 9 章关注的是如何使用贝赛尔曲线和表面来写应用程序，但不会就这个话题进行任何延伸。第 5 章展示了如何使用旋转、转换和缩放，但不会深入其基本数学原理。本书的主题划分大致遵循教材的顺序，但同时也代表了学习 OpenGL 时的一个自然的、循序渐进的过程。首先从第 2 章开始讲解二维问题，第 3 章讲解交互性，第 4 章和第 5 章讲解基本的三维程序。第 6 章开始介绍光源和材质，第 7 章和第 8 章则讲解如何使用 OpenGL 来显示离散实体，首先讨论的是像素和位图，接着讨论的是材质映射。第 9 章讲解了曲线和表面。第 10 章展示了一个比以前各章都要长的例子，这个例子覆盖了以前介绍过的大多数主题，同时引入了一些更高级的 OpenGL 特性。第 11 章则描述了最新一代可编程图形卡。

本书既包含完整的程序，也包含代码片断。你会发现，一旦写出自己的第一批 OpenGL 应用程序，其中大多数代码都会在以后的程序中反复出现。因此，在详细讨论了第一批例子之后，我会在以后的程序中删除大量重复代码。读者可以从我的网站（*www.cs.unm.edu/~angel*）下载完整的例子，或者从 FTP 站点 *ftp.cs.unm.edu* 下载（在 pub/angel 目录中）。

目　录

第1章 基础知识

OpenGL 是一种交互式计算机图形系统，允许程序员编写出可操纵图形硬件的程序。OpenGL 能为应用程序的程序员带来两方面重要的好处。首先，它非常接近硬件，所以用 OpenGL 写的程序能高效地运行。其次，OpenGL 很容易学习和使用。本章将概要介绍 OpenGL，包括它能做（和不能做）什么，它是如何组织的，以及我们应该如何表示它。

1.1 OpenGL API

操作现代计算机时，计算机图形几乎是您所做的一切事情的重要组成部分。不管是访问网页，玩互动游戏，还是使用 CAD 软件来设计一所房子，都要用到计算机图形。现在的硬件和软件变得越来越快，越来越复杂，我们所用的图形应用程序也不例外。这些应用程序的开发者需要依赖标准化的软件接口来构建他们的应用程序。有了这些接口之后，程序员就不必为许多应用程序中通用的标准函数编写代码，同时使应用程序避免接触过多的硬件细节。如此一来，我们能更快开发出程序，而且这些程序的移植性也越来越好。应用程序的程序员通过一系列函数来接触图形系统，这些函数具有一个经过了良好定义的接口，也就是所谓的“**应用程序编程接口**”（Application Programmer's Interface，API）。

经过多年的发展，历史上出现了许多图形 API。有的 API（比如 GKS 和 PHIGS）曾经被提升为国际标准。有的 API 则广泛地应用于特定的应用程序。但是，其中大多数 API 都只具有短暂的生命期。另一些 API（比如 Microsoft 的 Direct X）只针对特定的平台。OpenGL 源自一个名为 GL 的接口，即“Graphics Library”（图形库），它最初是为 Silicon Graphics Inc.（SGI，硅谷图形公司）的硬件开发的。事实证明，GL 是一个简单但功能强大的接口。GL 构成了 OpenGL 的基础，后者随即获得了大量图形硬件的支持。在 OpenGL 中，您可以使用 200 多个函数来构建应用程序。使用 OpenGL 编写的程序可移植到支持该接口的其他任何计算机上。几乎所有硬件和操作系统都实现了 OpenGL。这些实现的范围从纯粹的软件模拟，一直到纯粹的硬件级支持。一个典型的 OpenGL 应用程序应该能在任何实现上运行——只需使用目标系统的 OpenGL 库重新编译一下。此外，OpenGL 还提供了高度的稳定性。因此，用 OpenGL 写的程序具有很长的生命期——虽然 API 也在不断地发展，不断地集成由最新的硬件提供的功能。

OpenGL API 主要关注的是图像的“**渲染**”（rendering），也就是根据几何图形对象的规范及其属性，使用虚拟的照相机和光源来构成一幅图像。OpenGL 程序是与平台无关的。因此，OpenGL API 既不包含输入，也不包含视窗函数，因为这两者都要依赖于具体的平台。然而，每个图形程序都必须和一个操作系统和本地视窗系统进行交互，不管它是 Windows，Linux，还是 Macintosh。但是，我们不必编写依赖于平台的代码，而是应该使用一个简单的开发工具箱，也就是 OpenGL Utility Toolkit（GLUT）。它的 API 包含了通用

于大多数视窗系统的标准操作，而且允许我们在应用程序中使用鼠标和键盘。

1.2 看待 OpenGL 的三个角度

虽然从某种程度上说，OpenGL 只是一个用于操纵计算机图形功能的函数库，但是，假如能从三个不同但又相互关联的角度来看待 OpenGL，将有助于我们更好地理解 OpenGL 和图形系统函数，并使我们能够编写出更好的代码。

1.2.1 从程序员的角度

通常，大多数图形应用程序都由以下 3 个要素构成：

- 指定要渲染的一系列对象；
- 描述这些对象的属性；
- 定义用什么方式来观视这些对象。

现代图形系统支持两种不同的对象类型：几何与图像。几何对象（geometric object）存在于一个典型的三维世界，既包括像线段、点和多边形这样的图元，也包含更复杂的实体，比如二次曲面和更泛化的曲线和表面。在历史上，计算机图形系统只和几何对象打交道，本书也主要围绕它们展开讨论。然而，硬件技术的最新发展允许系统处理“图像对象”（image object）——也就是由像素构成的矩形阵列，每个像素都用一个或者多个值来表示像素的颜色、亮度以及其他属性。

大多数 API 都严格区分了“对象（一条线段或一个多边形）是什么”和“对象如何显示”这两个概念。因此，一条绿色的实线和一条红色的虚线实际是相同的对象类型，只是具有不同的外观。虽然这种区分似乎会和“面向对象编程”的许多概念相冲突，但它更接近硬件的工作方式，而且程序员可在一个时间内规划程序的逻辑流程，在另一个时间内指定众多的参数来描述平面的属性，两者互不相干。

计算机图形遵循现代物理及工程建模标准，对象的描述/行为与这些对象的观视方式是区分开的。要生成一幅图像或者图片，必须同时描述构成图像的对象本身以及形成图像的照相机或取景器。在 OpenGL 中，有一个系列函数专门负责一台虚拟照相机的安放和描述。

产生图形输出的任何程序都必须具有以上 3 个要素。除此之外，如果程序是交互式的，还必须提供输入功能。最后，所有程序都要使用一些初始化函数和终止函数，它们与本地操作系统和视窗环境进行交互。

从程序员的角度来看待 OpenGL，提供了对众多 OpenGL 函数进行分类的一种方式，我们准备在 1.3 节描述这种分类方式。不过，这个角度并没有解释 OpenGL 的工作方式。所以，必须借助另外两个角度，才能全面地理解 OpenGL。

1.2.2 OpenGL 状态机

从一个稍微不同的角度，我们可以将 OpenGL 视为一个具有输入和输出的“状态机”

（state machine），如图 1.1 所示。输入的是对几何对象（比如线段和多边形）和离散对象（比如位图）的描述。这些输入是在执行 OpenGL 函数调用时提供的。输出的就是我们在屏幕上看到的图像。在输入和输出之间，就是一台虚拟的机器，它获取对象描述，并将其转换成最终的图像。至于这个“机器”具体如何处理输入，要取决于它的状态。从这个角度看，OpenGL 提供了两种类型的函数：一种类型的函数指定到机器（对象）的输入，另一种类型的函数则更改机器的状态。负责更改状态的函数包括那些指定颜色、观视条件、材料性质以及其他许多变量的函数。“状态”决定了如何对输入进行处理。

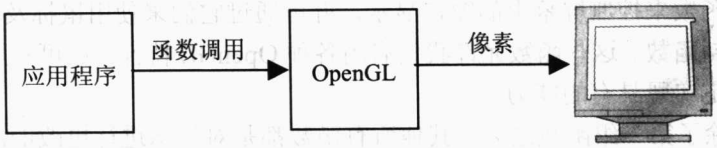


图 1.1　将 OpenGL 视为状态机

1.2.3　OpenGL 管道

从实现的角度，我们可以通过一种不同的方式来看待 OpenGL 程序的行为。OpenGL 基于所谓的“**管道模型**”（pipeline model）。图 1.2 展示了一条简单的管道。所有实时系统和低端显卡都集成了一条硬件管道。图形图元在应用程序内部生成，并经由管道传送。管道中包含了一系列模块，每个模块都对经过的图元采取一项或多项操作。有些模块提供转换功能，能旋转、转换和缩放对象。有些模块负责相对于 OpenGL 的照相机来定位对象。另一些模块则判断对象是否在照相机的视界内。在管道的末尾，那些可见的图元会被转换成屏幕上的彩色像素。

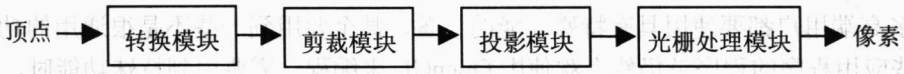


图 1.2　OpenGL 管道模型

我们可将管道视为 OpenGL 状态机的一种特定的实现。更重要的是，管道模型近似于大多数图形硬件系统的构建方式。管道模型还强调了对图元的处理是单独进行的。换言之，每个图元的颜色及其可见性都是独立于其他所有图元来确定的。正是因为这个原因，所以如果要保证管道构架的高吞吐速度，就不能集成一些全局性的特性，比如要依赖于图元之间的关系的阴影渲染。设计者必须在这两个方面做出取舍。

1.3　OpenGL 中有什么

OpenGL 包含 200 多个函数。按照功能对这些函数进行分类是大有帮助的：

- **图元函数**。这些函数定义了可在屏幕上生成图像的元素。有两种类型的图元：几何图元（例如多边形），它们可采用 2 个、3 个或者 4 个维度来定义；以及图像图元，比如位图。

- **属性函数**。这些函数控制图元的外观。这些函数定义了颜色、线条类型、材料性质（material property）、光源以及材质。
- **观视函数**。这些函数决定了照相机的属性。OpenGL 提供了一个虚拟照相机，可相对于图元函数定义的对象来定位它和调整它的方向。还可对照相机的镜头进行控制，使其产生广角与远距拍摄效果。
- **视窗函数**。这些函数不是 OpenGL 核心的一部分。但是，由于它们对于交互式应用程序的重要性，所以使用单独的库（比如 GLUT）来容纳它们。我们可利用这些函数来控制屏幕上的窗口显示，并可通过它们来使用鼠标及键盘。
- **控制函数**。这些函数允许我们启用各种 OpenGL 特性。还可利用它们来了解一个特定实现具有的能力。

注意，除了第一组函数之外，其他所有函数都是对状态进行更改的函数，而且它们本身不能生成对显示的改变。

1.4 OpenGL 的版本和扩展

OpenGL 由 OpenGL Architectural Review Board（ARB，OpenGL 构架评委会）进行控制。该组织的成员来自 Silicon Graphics，IBM 和 NVIDIA 这样的公司。OpenGL 目前的版本是 1.5。这个版本的 OpenGL 非常稳定，根据之前规范编写的大多数程序都能毫无问题地在最新版本上运行。大多数图形系统都支持 OpenGL 中的各种功能。即便一个特定的平台没有在硬件级别上实现某些特性，但也往往能通过软件来实现这些特性。本书只需用到版本 1.2 的功能，第 11 章将总结版本 1.3，1.4 和 1.5 的附加特性。

许多高端用户都要使用具有特殊功能的设备，甚至要用到一些不是很通用的功能。然而，这些应用程序的程序员仍然喜欢使用 OpenGL 来编程。需要用到特殊功能时，我们要用“OpenGL 扩展”（OpenGL extension）。这些扩展已经通过了 ARB 的认证，但尚未纳入核心 OpenGL 规范。例如，“图像处理”就是一个非常流行的扩展，它扩展了第 7 章要讨论的像素操作。但是，这种扩展往往需要用到高端图形工作站支持的一些特殊硬件能力。

1.5 语　　言

我们打算使用 OpenGL 的 C 语言绑定（C binding），这是最流行的做法。另外还有一个正式的 Fortran 版本。但遗憾的是，其他流行的语言（比如 Java）目前还没有用于 OpenGL 的正式绑定。不过，您可以在网上搜索到一些非正式的 Java 语言绑定。

1.6 编程约定

OpenGL 函数包含在两个通常叫做 gl 和 glu（或 GL 和 GLU）的库中。第一个库是核

心 OpenGL 库，它包含了所有必要的 OpenGL 函数；第二个库是 OpenGL 工具库（OpenGL Utility Library），其中包含了用核心库的函数编写的扩展函数，用户使用这个库时，可领略到意想不到的方便。核心库的函数名以 gl 开头，比如 glVertex3f()，而工具库的函数名以 glu 开头，比如 gluOrtho2D()。

OpenGL 不是面向对象的，所以我们应该使用它的 C 语言绑定。正是因为这个原因，API 没有利用面向对象语言（比如 C++）所支持的一些特性，比如函数重载。当然，程序员完全可以使用 C++来开发面向对象的应用程序，然后与 OpenGL C 实现进行链接。为了支持 C 程序员使用的大量数据类型，许多 OpenGL 函数都具有多种形式。例如，glVertex3f()要求获取浮点实参，而 glVertex3i()要求获取整型实参。为此，我们使用 glVertex*()这样的标注形式来反映 Vertex 函数的所有形式。

许多 OpenGL 函数的参数值是从小的、离散的整数集合中选取的。为了避免使用“魔数”（magic number），OpenGL 为这些值定义了宏（“魔数”是编程时应该避免的。例如，假定一个程序不断将数字 10 作为一个循环的计数器来使用，人们就会产生“10 就是计数器”的错觉。如果程序中还包括了与计数器无关的其他 10，就会使读者产生混淆。——译者注）。这些宏是在包容文件 gl.h 和 glu.h 中指定的。前缀 GL_和 GLU_定义了宏来源于哪一个包容文件，比如 GL_LINES 和 GL_LIGHT0。

OpenGL 使用基本的 C 数据类型：float，double，int，char 等等。但是，为了让一个实现能够重新定义它的基本类型，包容文件 gl.h 定义了基本类型 GLint，GLfloat，GLdouble 以及 GLbyte。虽然这些类型通常就是您所希望的，但它们是在 OpenGL 函数定义中使用的。例如以下函数：

```
void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
```

它使用 3 个 GLfloat 定义了三维空间中的一个顶点。事实上，gl.h 中为每种类型都添加了一行定义，形如：

```
typedef GLfloat float
```

所以，这些值正是我们所希望的。有些时候，您还会看到一些不太熟悉的类型，比如 GLclampf（用于表示范围在 0.0～1.0 之间的浮点数）以及 GLsizei（用于表示当前实现所支持的最大非负整数）。

许多函数都要求允许数组指针作为自己的参数，OpenGL 为它们准备了备用函数。这些函数的实参列表之前有一个 v。所以，我们可以像下面这样使用 glVertex3fv()函数：

```
GLfloat point[3] = {1.0, -2.5, 0.5};
glVertex3fv(point);
```

当我们描述函数时，会使用大括号和单词 TYPE 来指定一个函数的多种形式。例如，可以像下面这样列出 glVertex*()的所有形式：

```
glVertex{234}{sifd}(TYPE coords, ....);
glVertex{234}{sifd}v(TYPE *coords);
```

换言之，我们必须在调用函数时指定一个维度（2，3 或 4）以及一个数据类型（short[s]，