HZ BOOKS

OpenGL® Reference Manual

Third Edition

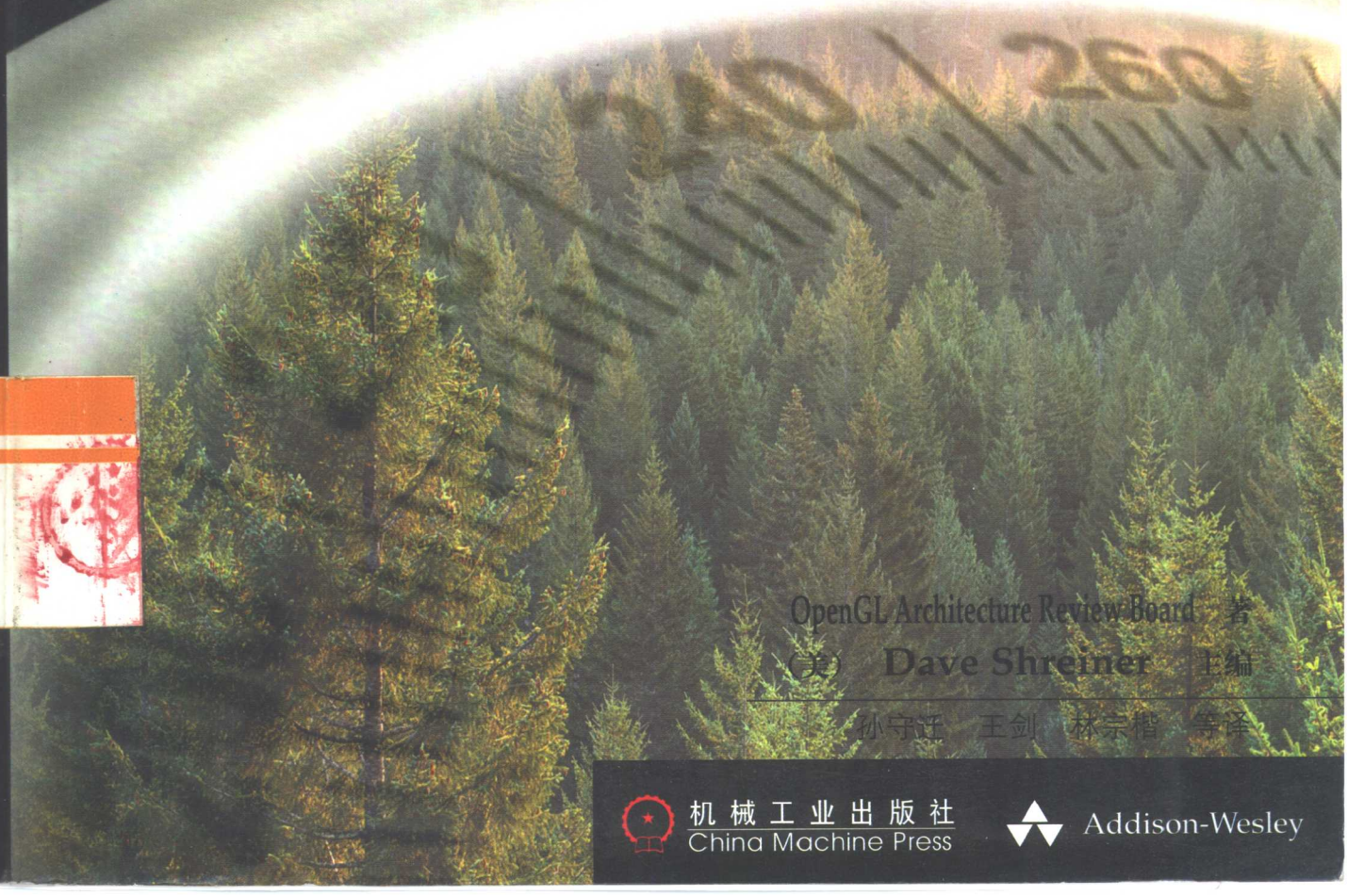
The Official Reference Document to OpenGL, Version 1.2

软件开发技术丛书

OpenGL®

参考手册

(第3版)



OpenGL Architecture Review Board 著

(美) Dave Shreiner 主编

孙守迁 王剑 林宗楷 等译

机械工业出版社 China Machine Press

Addison-Wesley

软件开发技术丛书

OpenGL® 参考手册

第3版

OpenGL Architecture Review Board 著
（美） Dave Shreiner 主编
孙守迁 王 剑 林宗楷 等译
李岳梅 罗仕鉴 审校

机械工业出版社
China Machine Press

本书经OpenGL结构评审委员会正式批准出版，全书全面权威地介绍了OpenGL的所有函数，OpenGL实用库（GLU 1.3）中的最新例程和对X窗口系统的OpenGL扩展（GLX 1.3）中的新增功能。此外，本书还对OpenGL的基本概念、所定义的常量及相关命令、对ARB扩展的描述等内容进行了专门介绍。

全书内容全面、结构严谨、叙述清晰，是一本有关OpenGL所有函数及命令的参考大全。对于专门从事三维图形开发的人员及高等院校师生，本书是一本必备的参考手册。

本书版权登记号：图字：01-2000-0306

图书在版编目（CIP）数据

OpenGL® 参考手册 / OpenGL结构评审委员会著；孙守迁等译. -3版. -北京：机械工业出版社/辽宁教育出版社，2001.1

(软件开发技术丛书)

书名原文：OpenGL Reference Manual, Third Edition: The Official Reference Document to OpenGL, Version 1.2

ISBN

I. O… II. ①O… ② 孙… III. 三维-动画-图形软件，OpenGL-技术手册 IV.TP391.41-62

中国版本图书馆CIP数据核字(2000)第52753号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：姚 蕾

山东省高青县印刷厂 印刷 · 新华书店北京发行所发行

2001年1月第1版第1次印刷

787mm × 1092 mm 1/16 · 31.25印张

印数：0 001-5 000册

定价：58.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

译 者 序

近年来，随着计算机技术的进步，我们跨入了一个三维时代，各种扣人心弦的三维游戏、能数字化地显示天气变化的气象服务、震撼人心的3D数字化特殊效果，无不使我们体验到三维世界的全新感觉。可视化、计算机动画、虚拟现实是当今图形学领域的三大热门话题，它们的技术核心都是三维图形。

1992年7月，SGI公司首次发布了作为三维图形编程接口的OpenGL。目前它已成为国际上通用的开放式三维图形标准。一方面，OpenGL规范由ARB（OpenGL Architecture Review Board，OpenGL结构评审委员会）负责管理，充分保证了它的独立性、开放性、前瞻性和跨平台性。它可被集成到Unix、Windows NT 4.0、Windows 98、X窗口等窗口系统中。另一方面，Compaq 、IBM、 Intel、Microsoft等在计算机界具有主导作用的公司纷纷采用OpenGL图形的国际标准。各种游戏加速卡、专用加速部件都能不同程度地提高OpenGL程序的运行性能。这些都推动了OpenGL的发展，并迅速成为三维图形的国际标准。再者，SGI公司不断推出以OpenGL为基础的高级开发工具，以满足对图形工具性能日益增长的需求。这一切使得OpenGL成为最流行的三维图形开发工具。目前它已被广泛应用于CAD/CAM/CAE、地质、航空、医学图像处理、广告、艺术造型、电影后期制作等领域。

OpenGL由大量功能强大的图形函数组成，它集成了所有曲面造型、图形变换、光照、材质、纹理、像素操作、融合、反选择、雾化等复杂的计算机图形学算法。开发人员可以利用这些函数对整个三维图形轻松进行渲染，从而达到数字化现实生活景象的目的。

本书是OpenGL 参考手册的第3版，对OpenGL的函数进行了详细而简洁的说明，是程序员利用OpenGL进行程序开发的不可缺少的工具书。本书第1章是OpenGL入门，第2章对命令和例程进行了简介，第3章集中介绍了各种命令和例程，第4章介绍了定义的常量和相关命令，第5章是OpenGL参考说明，第6章是GLU的参考说明，第7章是GLX的参考说明。

本书在翻译过程中得到了国家863计划项目（863-511-942-016）的支持。参加翻译的人员还有王火亮、许宇荣、杨勤、杨颖、余牛、周贵仔、蒋丽、李岳梅，在此感谢他们的辛勤劳动。

由于计算机软件行业的飞速发展，加之时间仓促，翻译中难免会有不妥之处，如能得到您的及时指正将不胜感激。

我们的电子邮箱地址为caid@cs.zju.edu.cn.

译 者

2000年9月于求是园

前　言

OpenGL是一个图形硬件的软件接口（“GL”即Graphics Library）。这一接口包含了数百个函数，图形程序员可以利用这些函数指定设计高品质的三维彩色图像所需的对象和操作。这些函数中有许多实际上是其他函数的简单变形，因此，实际上它仅包含大约180个左右完全不同的函数。

OpenGL 实用库（OpenGL Utility Library，GLU）和对X窗口系统的OpenGL扩展（OpenGL Extension to the X Window System，GLX）为OpenGL提供了有用的支持特性和完整的OpenGL核心函数集。本书详细介绍了这些函数的功能。书中各章内容如下：

• 第1章　OpenGL简介

在概念上对OpenGL作了概述。它通过一个高层的模块图来阐述OpenGL所执行的所有主要处理阶段。

• 第2章　命令和例程概述

较详细地阐述了OpenGL对输入数据的处理过程（用顶点形式来指定一个几何体或用像素形式来定义一幅图像时），并告诉你如何用OpenGL函数来控制这个过程。此外，在本章中还对GLU和GLX函数作了讨论。

• 第3章　命令和例程一览

根据OpenGL命令所完成的功能列举说明了这些命令组。一旦了解了这些命令的功能，你就可以利用这些完整的函数原型作为快速参考。

• 第4章　定义的常量及相关命令

列举了在OpenGL中定义的常量和使用这些常量的命令。

• 第5章　OpenGL参考说明

本书的主体部分，它包括各组相关的OpenGL命令的描述。带参数的命令和与之一起描述的其他命令仅在数据类型方面有所不同。每个函数的参考说明介绍了参数、命令的作用和使用这些命令时可能发生的错误。

此外，本章还包含了有关OpenGL的 ARB扩展——多重纹理和绘图子集的参考说明。需要说明的是并非所有的OpenGL的环境都支持ARB扩展。

• 第6章　GLU参考说明

本章包含了所有的GLU命令的参考说明。

• 第7章　GLX参考说明

本章包含了所有的GLX命令的参考说明。

0.1　阅读此书前的预备知识

本书是OpenGL Architecture Review Board，Mason Woo、Jackie Neider、Tom Davis 和 Dave Shreinre编著的《OpenGL编程指南（第3版）》（Reading，MA：Addison-Wesley，1999）

的姊妹篇。阅读这两本书的前提是你已经懂得如何用C语言编程。

两本书的不同之处主要在于：《OpenGL编程指南》一书着重于介绍如何运用OpenGL，而本书的重点则是OpenGL的工作方式。当然要想彻底地了解OpenGL，这两方面的知识都是必需的。这两本书的另一个不同点是本书的大多数内容都是按字母次序编排的，这样编排的前提是假定你已经知道你所不明白的地方而仅仅想查找某个特定命令的用法。而《OpenGL编程指南》一书的编排则更像一本指南：它首先解释了OpenGL的简单概念，然后再导出更复杂的概念。虽然你不必通过阅读《OpenGL编程指南》一书来理解本书对命令的解释，但如果你已经读过它，你将会对这些命令有更深刻的理解。

如果你对计算机图形学还不太了解，那么请先从《OpenGL编程指南》一书入手学习，并同时参考下面这些书：

- James D. Foley、 Andries van Dam、Steven K. Feiner和John F. Hughes著，《计算机图形学：原理及应用》（Computer Graphics:Principles and Practice)。(Reading， MA：Addision-Wesley)。该书是一本计算机图形学的百科全书，它包含了丰富的信息量，但最好在你对这门学科有一定的实践经验之后再读它。
- Andrew S. Glassner 著，《3D计算机图形学：艺术家与设计师的用户指南》(3D Computer Graphics: A User's Guide for Artists and Designers)。(New York：Design Press)。这是一本非技术性的、综合介绍计算机图形学的书，它着重于所能获得的视觉效果而非如何获取这些效果的具体技巧。
- Olin Lathrop著，《计算机图形学的工作原理》（The Way Computer Graphics Work）。(New York：John Wiley and Sons，Inc.)。这本书概括性地介绍了计算机图形学，主要面向初级和中级计算机用户。它介绍了理解计算机图形学所必需的一般概念。

0.2 字体约定

本书使用如下的字体约定：

黑体字（**Bold**）——命令和例行程序名；

斜体字（*Italics*）——变量名、自变量名、参数名、空间维数和文件名；

正体字（Regular）——枚举类型和定义的常量；

等宽字体（`Monospace font`）——示例代码。

值得注意的是本书所使用的命令名称都是缩写形式。许多OpenGL命令只是其他命令的变种。简言之，这里只使用函数的基本名称。如果此命令上加有星号（*），则说明它所代表的实际的命令名称可能比显示的命令名称要多。如，**glVertex***代表所有指定顶点的命令变种所构成的命令。

多数命令的区别仅在于它们所带的自变量的数据类型。有些命令则在相关自变量的数目、这些自变量是否被指定为向量以及是否需在列表中单独指定等方面存在着区别。例如，你使用**glVertex2f**命令时必须以浮点数形式提供x和y的坐标；而使用**glVertex3sv** 命令时你需为x，y，z提供一个包含三个短整型值的数组。

0.3 致谢

本手册的初版是许多人共同努力的结果。Silicon Graphics的Kurt Akeley，SABL Productions

的Sally Browning以及Silicon Graphics的Kevin P. Smith为第1版提供了大量的资料，另外还有Jackie Neider和Mark Segal(他们均来自Silicon Graphics)。Mark和Kurt合著《The OpenGL Graphics System: A Specification》，Kevin著《OpenGL Graphics System Utility Library》，Phil Karlton 著《OpenGL Graphics with the X Window System》为本书作者提供了文献来源。Phil Karlton和Kipp Hickman帮助在Silicon Graphics定义并创建了OpenGL，此外还有Gain Technology, Inc.的Raymond Drewry、Digital Equipment Corp.的Fred Fisher、Kubota Pacific Computer, Inc.的Randi Rost等人也为本书的编写提供了帮助。OpenGL 结构评审委员会的成员Murray Cantor以及International Business Machines的Linas Vepstas、Digital Equipment Corporation的Paula Womack和Jeff Lane、Intel的Murali Sundaresan，还有Microsoft的Chuck Whitmer也提供了很多帮助。Thad Beier同Seth Katz以及Silicon Graphics的Inventor小组一起制作了封面图形。Silicon Graphics的Kay Maitz、Evans Technical Communications的Arthur Evans以及Susan Blau提供了产品援助，Tanya Kucak对本手册进行了编辑。当然，如果没有OpenGL，也就不会有本书的存在，所以要感谢Silicon Graphics的OpenGL小组所有成员，感谢他们的辛勤工作。他们是：Momi Akeley、Allen Akin、Chris Frazier、Bill Glazier、Paul Ho、Simon Hui、Lesley Kalmin、Pierre Tardif、Jim Winget，尤其是Wei Yen。另外，还有上面提到的Kurt、Phil、Mark、Kipp以及Kevin。当然还有许多其他的Slicon Graphics成员也为改进OpenGL的定义和功能做出了很多贡献，在这里也一并感谢他们。

Kempf的Renate Kempf及其同事、Silicon Graphics的Chris Frazier为《OpenGL Reference Manual for OpenGL， Version1.1》添加了所有OpenGL1.1 Specification中的新功能，并编辑审查了其他所有参考说明书。下列人员对该书进行了仔细的复审，他们是Allen Akin、David Blythe、Craig Dunwoody、Chris Frazier以及Slicon Graphics的Paula Womack、OpenGL 结构评审委员会中的成员，包括Silicon Graphics的Kurt Akeley、HP的Dave Arns、E&S的Bill Armstrong、Intergraph的Dale Kirkland和IBM的Bimal Poddar。Silicon Graphics的Simon Hui复审了GLX参考说明，John Spitzer复审了已校对的图形插页。

在本书中，SGI的Dave Shreiner添加了OpenGL 1.2和GLX 1.3的大部分新的功能，并在David Yu的帮助下重新修订了图面。Norman Chin重新修订了GLU 1.3的参考说明。下列人员认真地进行了手册复审这一艰巨的工作，他们是：Ron Bielaski、Steve Cunningham、Jeffery Galinovsky、Eric Haines、Mark Kilgard、Dale Kirkland、Seth Livingston、Bimal Poddar、David Nishimoto、Mike Schmitt、Scott Thompson、David Yu以及SGI的OpenGL小组的成员Craig Dunwoody、Jaya Kanajan、George Kyraizis、Jon Leech和Ken Nicholson。

尤其感谢Jon Leech，是他编辑了OpenGL 1.2.1、GLU 1.3和GLX 1.3的说明书，以及使OpenGL保持活力和生机的OpenGL ARB。同时也要感谢Laura Cooper和Dany Galgani为本手册的编写所提供的产品支持。

原书书号：ISBN 0-201-65765-1

原出版社网址:www.aw.com/cseng/

目　录

第1章 OpenGL简介

OpenGL是图形硬件的一个软件接口。它的主要作用是将二维或三维的对象绘入一个帧缓冲区中。对象被描述为一系列的顶点（用来定义几何对象）或像素（用来定义图像）。OpenGL对数据进行几个步骤的处理从而将其转换成像素，这些像素将在帧缓冲区中形成最终需要的图形。

本章将全面地介绍OpenGL的工作原理，包括以下两个主要部分：

- **OpenGL基础** 主要解释基本的OpenGL概念，例如什么是几何图元以及OpenGL如何实行客户端-服务器端的执行模式。
- **基本OpenGL操作** 通过一个高层的模块图来说明OpenGL在帧缓冲区中处理数据并生成相应图像的过程。

1.1 OpenGL基础

本节主要解释一些OpenGL固有的命令。

1.1.1 OpenGL图元及命令

OpenGL通过几个可选模式来绘制“图元”——点、线段或多边形。你可以对各种模式独立进行控制；也就是说，一个模式的设置并不影响其他模式的设置（尽管模式间的相互作用将影响帧缓冲区中的最后结果）。OpenGL的程序通过调用函数来指定图元、设置模式并描述其他操作。

在OpenGL中，图元由单个或多个顶点组来定义。一个顶点可以是一个点、一条线的端点或一个多边形的角。数据（由顶点坐标、颜色、法线、纹理坐标和边界标志所组成）与顶点是相对应的，并且每个顶点和与它相关的数据独立，按照次序，采用同样的方法进行操作。这里仅有一种情况例外，那就是当一组顶点必须被“剪切”从而使得某一特定的图元刚好在某一指定的区域内，则顶点数据可能会被修改并产生新的顶点。其剪切的类型由该组顶点所代表的图元决定。

虽然有些命令在生效前可能会有一段不确定的延时，但是OpenGL的所有命令都是依照其被接收的次序来执行的。也就是说，每个图元在被绘制完成之前，其后面的命令将不会有效。同时，这也意味着使用状态查询命令时它所返回的数据将只包含所有以前发布并已执行完毕的OpenGL命令。

1.1.2 OpenGL是一种过程语言

OpenGL从根本上说是一种过程语言而非描述性的语言：OpenGL提供了直接控制二维和三维几何体的基本操作。它包含了转换矩阵、光照方程系数、反走样方法和像素校正算子的描述。然而，OpenGL并不能直接描述或建模复杂的几何对象。

你所发布的OpenGL命令指定了怎样产生一个特定的结果（即接下来所应该采取的操作）而不是指定确切的结果。正是这种过程特性使我们能够了解OpenGL是如何工作的——只有明白了它的操作顺序才能对如何使用它有更深的理解。

1.1.3 OpenGL的执行模式

OpenGL使用了一种客户端-服务器端的模式来解释命令。应用程序（客户端）所发布的命令将通过OpenGL（服务器端）来编译和处理。服务器的操作既可以同客户端在同一台计算机上进行，又可以分别属于不同的机器。因此，从这个意义上讲，OpenGL是网络透明的。一个服务器可以维护数个GL上下文，每个上下文被封装在一个GL状态里。服务器可以同时包含几个GL上下文，每个上下文都被封装在一个GL状态里。每个客户端都可以连接到这些上下文中的任何一个。所需要的网络协议可以是扩充过的已有协议（如X Window系统）或是一个完全独立的协议。OpenGL并没有提供命令用来获取用户的输入。

窗口系统分配给帧缓冲区的资源将最终控制OpenGL命令对帧缓冲区的影响。窗口系统将决定OpenGL帧缓冲区的哪些部分可在给定的时间内被访问并将这些部分的结构传送给OpenGL。因此OpenGL中不存在配置帧缓冲区及初始化OpenGL的命令。帧缓冲区的配置是在OpenGL外，由与其相关联的窗口系统来完成的，而OpenGL的初始化则是当窗口系统为OpenGL绘图分配一个窗口时完成的。（GLX——OpenGL界面的X扩展，提供了这些功能。有关细节请参阅2.4节“对X窗口系统的OpenGL扩展”。）

1.2 基本OpenGL操作

图1-1是抽象的高层的模块图，它展示了OpenGL处理数据的过程。如图所示，命令由左边进入，然后经过了一个可被看作处理流程的处理过程。其中有一些命令指定了所要绘制的几何对象，而另一些命令则用于控制不同阶段中对象的处理方法。

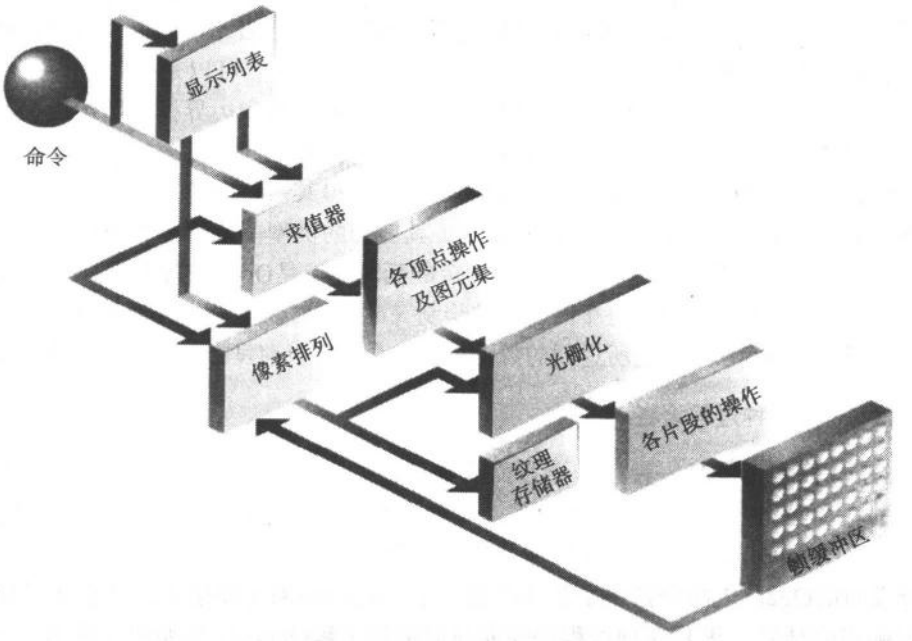


图1-1 OpenGL数据处理过程

当命令进入流程时，你可以选用两种方法对它们进行处理：一种是通过流程立即执行这些命令；另一种是将其中一些命令组织到一个“显示列表”，过一段时间再执行它们。

流程中的“求值器”阶段通过将输入值赋给多项式命令提供了一种非常有效的方法来生成几何曲线和曲面的近似值。接下来的“各顶点操作及图元集”阶段主要是处理OpenGL的几何图元——点、线段和多边形。所有这些图元均由顶点来描述。顶点可以被转换和照亮。接下来图元被剪切到视口，为下一阶段做好了准备。

“光栅化”生成了一系列的帧缓冲区地址以及相应的用于描述点、线段或多边形的二维值。这些生成的“片断”将被送到最后一个阶段——“各片断的操作”，这一阶段是数据以像素形式存入“帧缓冲区”之前的最后操作。这些操作包括根据帧缓冲区中原有的深度值（用于深度缓存操作）与输入值而有条件地更新帧缓冲区的操作，还包括对输入的像素颜色与已存储的颜色所进行的融合操作，对像素值所进行的屏蔽操作及其他的逻辑操作。

数据是以像素形式而非顶点形式输入的。这些数据可以用来描述一个用于纹理映射的图像，它将跳过第一阶段（如前面所述），而通过“像素操作”阶段作为像素来处理。这一阶段的处理将导致两种结果：其一是被存入“纹理存储器”，以备光栅化阶段所用；其二是直接被光栅化。后者所形成的片断将被存入帧缓冲区，就好象它们是由几何数据生成的一样。

一个OpenGL应用程序可以获得OpenGL状态的所有元素，其中包括纹理存储器的内容，甚至还包括帧缓冲区的内容。

第2章 命令和例程概述

许多OpenGL命令直接影响诸如点、线、多边形以及位图等OpenGL对象的绘制。而另一些命令，例如那些用于反走样或纹理操作的命令，主要用来控制图像如何生成。还有一些命令则关注帧缓冲区的操作。

本章主要介绍所有OpenGL命令是如何协同工作来建立OpenGL处理流程的。同时也对OpenGL实用库（GLU）和对X窗口系统的OpenGL扩展（GLX）中的命令作了概述。

本章包括以下几个主要部分：

- **OpenGL处理流程**：在第1章的基础上讲解特定的OpenGL命令如何控制数据的处理。
- **其他OpenGL命令**：讨论几个前一章中没有提及的OpenGL命令集。
- **OpenGL实用库**：介绍了已有的GLU例程。
- **对X窗口系统的OpenGL扩展**：介绍GLX中有用的例程。

2.1 OpenGL处理流程

第1章介绍了OpenGL如何工作，本章将进一步讨论各阶段中数据处理的实际情况并且将各阶段与其用到的命令结合起来。图2-1是一幅较为详细的OpenGL处理流程图。

从图中我们可以看到其中有三组箭头穿过了大多数的阶段。这三组箭头分别代表了顶点和与其相关的两个主要的数据类型——颜色值和纹理坐标。值得注意的是顶点首先组合成图元，然后是片断，最后成为帧缓冲区中的像素。这一过程将在下面章节中作详细介绍。

一个OpenGL命令的效果将很大程度地依赖于某特定模式是否有效。例如，与光照有关的命令只有当你启动了光照功能才能有效地生成一个适当的光照对象。如果要启动一个特定的模式，请调用**glEnable( )**命令，并且要提供一个适当的常量来确定该模式（如GL_LIGHTING）。下面章节中并没有介绍特定的模式，但在函数**glEnable()**的使用说明中提供了一个完整的列表用来说明它可启动的模式。调用函数**glDisable()**可以关闭一个模式。

2.1.1 顶点

本节介绍与在图2-1中与各顶点操作有关的OpenGL命令。它包含了有关顶点数组的各种信息。

1. 输入数据

你必须为OpenGL流程提供几种输入数据类型。

- 顶点——顶点用来描述所需要的几何对象的形状。你可以通过在函数对**glBegin()/glEnd()**之间调用函数**glVertex*()**来指定顶点，并用这些顶点建立点、线或多边形。你也可以用函数**glRect*()**来直接绘制一个完整的矩形。
- 边界标志——在缺省情况下，多边形的所有边都是边界边。用函数**glEdgeFlag*()**可以显式

地设置边界标志。

- 当前光栅位置——当前光栅位置用来确定绘制像素和位图时的光栅坐标。它由函数**glRasterPos*()**指定。

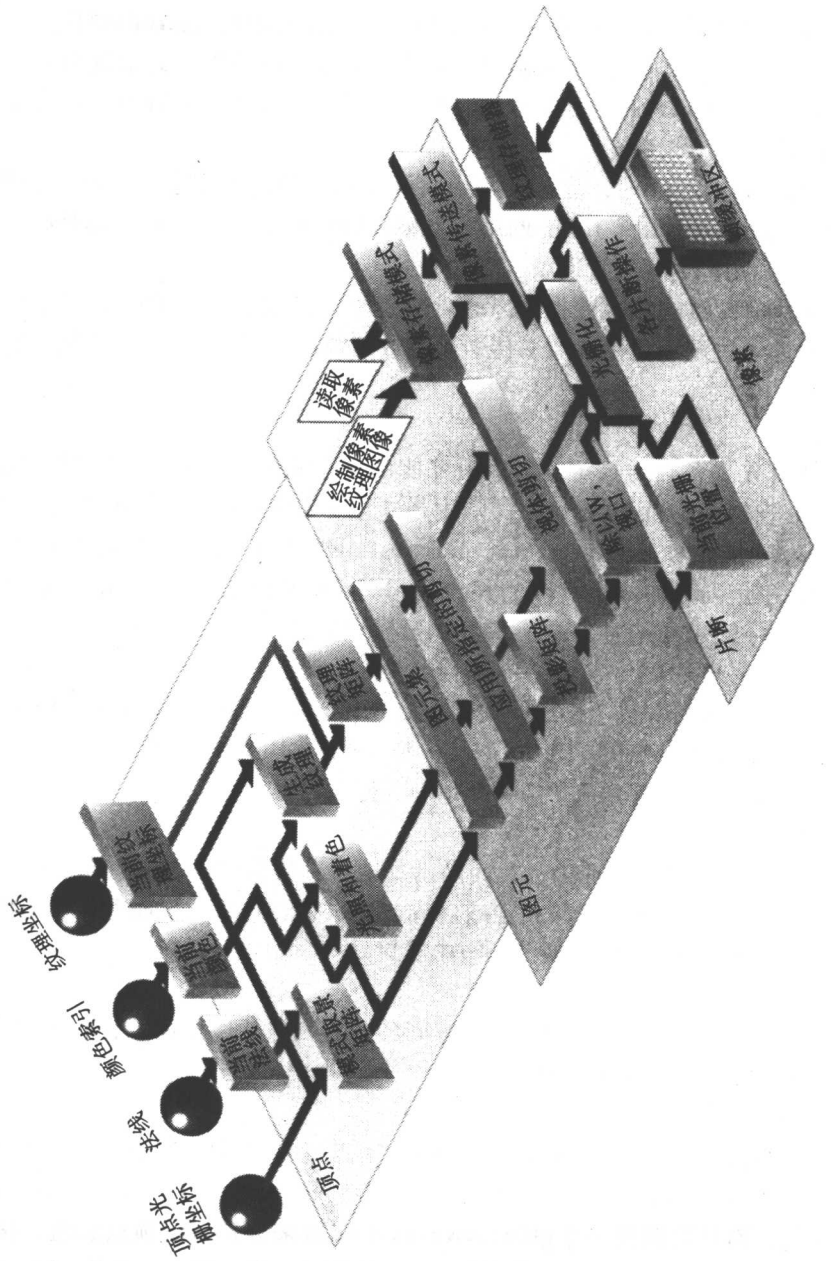


图2-1 OpenGL处理流程的各个阶段

- 当前法线——每个法向量都与一个特定的顶点相对应，它用来确定顶点处的表面在三维空间中的方向。它同时又影响该顶点所接收的光照的多少。函数**glNormal***()用来指定一个法向量。
- 当前颜色——一个顶点的颜色用来确定光照对象最终的颜色。在RGBA模式下，可以通过函数**glColor***()来指定颜色；在颜色索引模式下则需使用函数**glIndex***()。
- 当前纹理坐标——纹理坐标用来确定在纹理映射表中的位置，此位置与一个对象的某个顶点相关联。它们可以由函数**glTexCoord***()指定。当系统支持ARB多重纹理扩展时用函数**glMultiTexCoord*ARB()**。

当调用函数**glVertex*( )**时，生成的顶点将继承当前的边界标志、法线、颜色和纹理坐标。因此，必须在函数**glVertex*( )**之前调用函数**glEdgeFlag*()**、**glNormal*()**、**glColor*()**和**glTexCoord*( )**来影响所生成的新顶点。

上面所列的所有顶点的输入数据可以通过使用 “顶点数组”来指定，在下面的叙述中会有详细介绍。它允许通过调用单个函数来传送顶点数组数据。某些OpenGL机制可能用该方法来指定顶点会更有效。

2. 矩阵转换

顶点和法线首先要各自通过矩阵转换后才能用于在帧缓冲区中生成图像。顶点通过模式取景矩阵和投影矩阵转换，而发光的法线则由模式取景矩阵转换。你可以使用诸如**glMatrixMode()**、**glMultMatrix*()**、**glRotate*()**、**glTranslate*()**和**glScale*()**等函数来组成所需的转换。或者，也可直接用函数**glLoadMatrix*()**和**glLoadIdentity()**指定矩阵。用函数**glPush- Matrix()**和**glPopMatrix()**可以在各自的堆栈中存储和恢复模式取景矩阵和投影矩阵。

3. 光照和着色

除了指定颜色和法向量外，你还可以用函数**glLight*()**和**glLightModel*()**指定所需的光照环境，用函数**glMaterial*()**指定所需的材料属性。用于控制光照计算的相关命令有：**glShadeModel()**、**glFrontFace()**和**glColorMaterial()**。

4. 生成纹理坐标

OpenGL并不明确地提供纹理坐标，而是用其他顶点数据的函数来生成它们。这项工作由函数**glTexGen*( )**完成。当纹理坐标被指定或生成之后，它们将通过纹理矩阵实现转换。控制这些矩阵所使用的命令与前面“矩阵转换”一节中所提及的命令是一样的。

5. 图元集

一旦所有的计算执行完，这些顶点——连同各顶点相应的边界标志、颜色和纹理信息——将被组合成图元（包括点、线段和多边形）。

6. 顶点数组

你只需调用有限的几个命令就可以通过顶点数组来指定几何图元。当你调用函数**glDrawArrays()**来绘制图元时，你不需要再通过调用一个个的OpenGL函数来传送每个单独的顶点、法线或颜色，而只要调用一个**glDrawArrays()**函数来分别指定顶点数组、法线数组和颜色数组就可以用它们来定义一系列要绘制图元（所有同一类型的）。函数**glVertexPointer()**、**glNormalPointer()**、**glColorPointer()**、**glIndexPointer()**、**glTexCoordPointer()**和

glEdgeFlagPointer()用来描述数组的组织结构和存储单元的位置。函数**glEnableClientState()**和**glDisableClientState()**用来指定将访问哪个顶点数组中的顶点坐标和属性。

一个顶点的所有当前有效的数据都可以通过在函数对glBegin()/glEnd()之间调用函数glArrayElement()来指定。此外，函数glDrawElements()和glDrawRangeElements()可以随机访问顶点数组。

7. 图元

在流程的下一个阶段中，图元将被转化成像素片断。该过程有以下几个步骤：适当地剪切图元；对颜色和纹理作必要的相关调整；将有关坐标转换成窗口坐标；最后，将剪切好的图元通过光栅化处理而转换成像素片断。

8. 剪切

当点、线段和多边形需要剪切时，OpenGL对它们的处理稍有不同。对于点而言，要么维持其原始状态（当它包含在剪切体积内时），要么被丢弃（当它处于剪切体积外时）。对于线段和多边形则不尽相同。如果线段或多边形的一部分处于剪切体积外，则在剪切点的位置上生成一个新的顶点。而对于多边形，这些新的顶点之间还需要重新连一条完整的边。当线段和多边形被剪切时，其边界标志、颜色和纹理信息都被赋给新顶点。

剪切实际上有两个步骤：

1) **由应用所指定的剪切**。图元一旦被组合而成，它们将根据应用的要求，通过函数glClipPlane()指定的剪切平面在*眼坐标*中进行剪切。（任何的OpenGL机制都支持至少六个这样的应用相关的剪切平面）。

2) **视体剪切**。接下来，图元将由投影矩阵转换（成*剪切坐标*）并被相应的视体剪切。你可以通过矩阵转换命令来控制这些矩阵。但更多地，它们由函数glFrustum()和glOrtho()指定。

9. 转换成窗口坐标

在剪切坐标转换成*窗口坐标*前，它们先要被归一化，即通过除以*w*值从而生成*归一化设备坐标*。之后，通过视口转换将这些归一化的坐标变为窗口坐标。你可以用函数**glDepthRange()**和**glViewport()**来控制这些视口——它们将决定显示图像的窗口屏幕区域。

10. 光栅化

光栅化是将一个图元转化为一个二维图像的操作。该图像中的每个点都包含这样一些信息：颜色、深度和纹理数据。点及其相关信息被称为一个“片断”。

当前光栅位置（由函数**glRasterPos*()**指定）在该阶段的像素绘制和位图中有多种用途。三种不同类型的图元的光栅化是各不相同的。另外，像素矩形和位图均需被光栅化。

- **图元**。下面的命令允许你通过选择图元的尺寸及点画模式对图元的光栅化进行控制：**glPointSize()**、**glLineWidth()**、**glLineStipple()**和**glPolygonStipple()**。你也可以通过命令**glCullFace()**、**glFrontFace()**和**glPolygonMode()**来控制正面多边形和背面多边形将如何被光栅化。
- **像素**。有几个命令用于控制像素的存储和传送模式。命令**glPixelStore*()**用来控制像素在客户端存储器中的编码方式，**glPixelTransfer*()**和**glPixelMap*()**控制像素存入帧缓冲区之前的处理方式。另外，当你使用的OpenGL机制支持“ARB绘图子集”（参见2.1.2节）时，

就可以对像素进行其他处理。像素矩形由函数**glDrawPixels()**指定，其光栅化由函数**glPixelZoom()**控制。

- **位图**。位图是由0和1所组成的矩形，它用来指定一个将要生成的特定格式的片断图案。每个这样的片断都有相同的相关数据。位图由函数**glBitmap()**指定。
- **纹理**。当纹理功能启动后，它将把一个指定的纹理图像的一部分映射到每个图元上。要想实现这种映射，你需要使用由片断的纹理坐标所确定的存储单元中的纹理图像的颜色来修改该片断的RGBA颜色。

  你可以用函数**glTexImage1D(), glTexImage2D()**或**glTexImage3D()**来指定一个纹理图像。如果要通过拷贝帧缓冲区中的数据来建立一个纹理图像，请使用函数**glCopyTexImage1D()**或**glCopyTexImage2D()**。你也可以通过**glTexSubImage1D(), glTexSubImage2D()**或 **glTexImage3D()**载入子图像，或通过函数**glCopyTexSubImage1D(), glCopyTexSubImage2D()**或**glCopySubImage3D()**用从帧缓冲区中拷贝来的数据替换部分纹理图像。函数**glTexParameter*()**和**glTexEnv*()**用来控制对纹理值进行解释并应用于一个片断。

  要指定具体哪些纹理优先存入纹理内存，需先调用函数**glBindTexture()**生成指定的纹理（纹理对象），然后再调用函数**glPrioritizeTextures()**给它们排序。函数**glDeleteTexture()**可以删除一个纹理对象。
- **颜色总和**。是指在纹理操作之后，将由镜面光照计算而来的颜色片断加到片段上。通过函数**glLightModel()**且将GL_LIGHT_MODEL_COLOR_CONTROL参数赋值为GL_SEPARATE_SPECULAR_COLOR可以指定镜面光照颜色进行分别计算。
- **雾**。OpenGL可以将一种雾颜色和一种已光栅化的片断的纹理颜色用一个融合因子融合在一起。该融合因子由观察点与片断之间的距离决定。用函数**glFog*()**可指定雾颜色和融合因子。
- **多边形偏移**。当你绘制消隐图形或对表面进行贴面时，应该考虑使用函数**glPolygonOffset()** 替换一个片断的深度值。该深度值是在绘制多边形时将一个特定的偏移量加到一个可调节的量上而生成的。该可调节量依赖于多边形深度值的变化量及它的屏幕尺寸。这种替换允许在同一个面上绘制多边形而相互间不产生影响。你可以启动如下所示的三种多边形模式之一来确定将采用怎样的替换方法。这三种多边形模式是GL_POLYGON_OFFSET_FILL，GL_POLYGON_OFFSET_LINE和GL_POLYGON_OFFSET_POINT。

多边形偏移可用于绘制消隐图形、绘制高亮边的实心体及将贴面应用于物体表面。

2.1.2 ARB绘图子集

OpenGL机制可以支持任选的OpenGL ARB绘图子集。该集合是由数个附加的像素处理操作组成的。下面的功能仅当调用函数**glLightString**(GL_EXTENSIONS)的返回值是字符串GL_ARB_imaging时才有效。

ARB图形子集所包含的功能如下所示：

- **颜色表**。颜色查询表提供了一种替换单个像素的方法。颜色表可用**glColorTable**例程指定。如果要指定一个基于帧缓冲区值的颜色表，可以调用函数**glCopyColorTable( )**。函数

glCopySubTable()允许你替换颜色表的一部分。而函数**glCopyColorSubTable()**将使用帧缓冲区中的值替换指定的部分。在指定颜色表时可以对它进行缩放和偏移。用函数**glColorTableParameter*()**来指定缩放和偏移值。

- **卷积滤波器**。卷积是结合附近的像素来计算一个最终的像素值的方法。卷积滤波器可由函数**glConvolutionFilter1D()**、**glConvolutionFilter2D()**或**glSeparableFilter2D()**指定。另外，卷积滤波器也可以用帧缓冲区中的值来指定，这时需要调用函数**glCopyConvolutionFilter1D()** 和**glCopyConvolutionFilter2D()**。卷积滤波器可以通过函数**glConvolutionParameter*()**指定的值对它进行缩放和偏移。经过卷积操作后，所得的像素值也可以进行缩放和偏移，函数**glPixelTransfer*()**用来指定缩放和偏移值。
- **颜色矩阵转换**。矩阵转换可通过颜色矩阵堆栈而应用于像素。函数**glMatrixMode (GL_COLOR)**用来修正当前的颜色矩阵。有关内容请参阅“矩阵转换”。经过颜色矩阵转换之后，像素值可以通过函数**glPixelTransfer*()**指定的值进行缩放和偏移。
- **直方图**。直方图用来确定像素矩形中值的分布情况。函数**glHistogram()**用来指定哪些颜色组件将被计数。用函数**glGetHistogram()**可以返回直方图的计算结果。函数**glReset-Histogram()**可以重新设置直方图表，而调用函数**glGetHistogramParameter*()**将返回用于描述直方图表的值。
- **最值**。每个像素矩形的最小和最大值可以用函数**glMinmax()**来计算。函数**glGetMinmax()**返回通过函数**glMinmax()**指定的颜色组件而计算所得的最小和最大值。调用函数**glReset-Minmax()**可以重新设置内部的最值表，而调用函数**glGetMinmaxParameter*()**将返回用于描述该表的参数值。
- **融合方程**。除了汇总外，像素还可以用融合而非叠加方式进行合并。函数**glBlendEquation()** 用来指定源和目标像素值将如何被合并。
- **常数融合颜色**。除了标准融合函数外，ARB绘图子集允许用常数作为源或目标颜色值的系数。调用函数**glBlendColor()**来指定常数融合颜色，它将与函数**glBlendFunc()**所指定的融合系数一起使用。

2.1.3 片断

当一个通过光栅化而生成的片断能通过一系列的测试时，OpenGL允许通过该片断来修正帧缓冲区中相应的像素。如果该片断没通过测试，则该片断可被用来直接替换帧缓冲区中的值，或者与帧缓冲区中已存在的值合并。具体情况将视特定模式的状态而定。

1. 像素所有权测试

第一项测试用来检验与一个特定的片断相应的帧缓冲区中的像素是否属于当前的OpenGL环境。如果是，则对片断进行下一项测试；否则，将通过窗口系统来决定是丢弃该片断还是要对该片断进行后面的操作。当一个OpenGL窗口不明确时，这一测试允许窗口系统来控制OpenGL的行为。

2. 裁剪测试

裁剪测试将丢弃通过函数**glScissor()**指定的任意屏幕上的校正矩形区域外的片断。