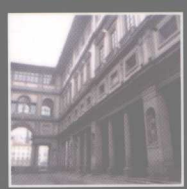
KH北京科海
25年 · IT技术出版专家

碁峰
www.gotop.com.tw

全球著名游戏公司
绘图引擎设计师倾力奉献
探讨最前沿的3D绘图技术，与游戏产业的技术发展保持同步

# 3D 绘图程序设计

## 使用 Direct3D 10/9 和 OpenGL 2.0



- **由浅入深**：从基本几何图形开始，渐进到传统的Fixed Function Pipeline（固定功能流水线）绘图，再深入最新的Shader程序设计
- **前沿技术**：讲解能真正应用于游戏产业的绘图技术，帮助您打造绚丽的游戏特效，所述开发技术与国际接轨
- **多种标准**：覆盖最新的Direct3D与OpenGL实时3D绘图技术
- **行业专家**：作者具有**7年**的游戏开发经验，开发过**7款**著名的3D游戏，目前在洛杉矶Activision Blizzard开发Xbox 360与PS3绘图引擎

彭国伦　编著

ICD
本书范例代码

科学出版社
北京科海电子出版社
www.khp.com.cn

# 3D绘图程序设计

## 使用Direct3D 10/9和OpenGL 2.0

彭国伦 编著

全球著名的游戏公司

Activision Blizzard（暴雪）

绘图引擎设计师倾力奉献

探讨最前沿的3D绘图技术，和游戏产业的

技术发展保持同步

科学出版社

北京科海电子出版社

www.khp.com.cn

## 内 容 提 要

本书系统、全面地介绍了 Direct3D 9、Direct3D 10 以及 OpenGL 2.0 三维图形程序设计方法。全书内容分成 3 大部分，第 1～10 章介绍传统的固定绘图流程和基本 3D 绘图概念，包括坐标转换、动画与交互、打光、贴图、混合与纹理、动态贴图、Stencil Buffer 和特效处理等内容。第 11～18 章为比较高级的 Shader 程序编写，包括 HLSL 和 GLSL 的使用、Shader 特效和调试 Debug 等内容。第 19~20 章是补充教学，介绍绘图引擎、Xbox360、PS3、GPGPU 和线性代数等基础知识，对刚入门或已经具有 3D 程序设计经验的读者都会有所帮助。配套光盘提供了书中所有 示例程序的可执行文件、工程文件和完整的源代码，以方便读者编译、调试示例程序。

本书主要面向对 3D 图形程序设计感兴趣的编程人员、游戏开发人员，以及可视化仿真工程技术人员，也可作为高等院校相关专业和培训机构的 3D 程序设计用书。

**图书在版编目（CIP）数据**

3D 绘图程序设计：使用 Direct3D 10/9 和 OpenGL 2.0/
彭国伦编著. —北京：科学出版社，2009
ISBN 978-7-03-024291-4

Ⅰ. 3… Ⅱ. 彭… Ⅲ. 图形软件，Direct3D 10/9、
OpenGL 2.0 Ⅳ. TP391.41

中国版本图书馆 CIP 数据核字（2009）第 041567 号

责任编辑：王海霞 / 责任校对：刘雪连
责任印制：科 海 / 封面设计：洪文婕

科学出版社出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

北京市鑫山源印刷有限公司印刷

科学出版社发行 各地新华书店经销

*

2009 年 5 月 第 一 版 开本：16 开
2009 年 5 月第一次印刷 印张：46.5
印数：0 001~3 000 字数：1 131 000

**定价：89.00 元（含 1CD 价格）**

# 序

由于近几年硬件的不断进步，个人计算机的 3D 绘图发生了非常大的变化。绘图芯片 GPU 早期只支持 2D 图形显示，到 20 世纪 90 年代末期进化到具有 3D 加速功能。21 世纪后，绘图芯片还提供了可编程的 Shader，让程序设计员可以自由控制 3D 画面的成像过程，3D 程序设计也因此进入了新的时代。

游戏产业是这些新科技的主要动力。早期的 3D 游戏在技术上的主要差别是程序优化，看谁的产品可以用更快的速度来画出复杂的场景。现在的游戏绘图，除了显示速度外，还可以在画面效果上一分高低。懂得如何使用 Shader 的程序员可以做出别人无法达到的效果。本书从传统的固定流程 Fixed Function Pipeline 绘图开始介绍，然后逐步进入最新的 Shader 程序设计。

目前的 3D 绘图程序接口以 Direct3D 和 OpenGL 这两套标准为主。到 2008 年底，Direct3D 的版本已经更新到 10.1，OpenGL 则更新到 3.0 。如果一开始就使用纯 Shader 环境的 Direct3D10 进行讲解会比较困难，所以本书还是先从 Direct3D9 和 OpenGL 1.1 开始，慢慢带入 Direct3D10 和 OpenGL 2.0/3.0 的使用。

程序设计方面以概念介绍为主，使用哪个平台、哪个链接库，甚至哪种编程语言都不是重点。以 Direct3D 和 OpenGL 为例，虽然它们是两套不同的标准，但是它们的功能和用法几乎完全相同，甚至其大部分指令都是一一对应的。

如果读者只打算学习 Direct3D 或者 OpenGL 的其中之一，就可以跳过不需要的部分，这样不会遗漏太多东西。若读者只想学习 Direct3D，建议还是把 OpenGL 部分浏览一次，反之亦然。读者会发现，所谓的跨平台支持事实上并不是一件太难的事情。本书的范例程序大部分都同时提供 Direct3D9，Direct3D10 以及 OpenGL 这 3 个版本供读者参考。

笔者目前在美国从事 Xbox360 和 PS3 游戏绘图程序开发，所以本书的大部分观点都是从游戏开发者的角度来看 3D 绘图。游戏产业是一种科技与娱乐相结合的产业，因此从事这个产业的人基本上都很愿意分享制作技术。每年的 GDC 和 SIGGRAPH 都会有很多来自全世界各地的人，他们聚集在一起发表并交换最新技术，只可惜这些数据都是英文的。由于产业规模比较小，再加上业界比较封闭，因此少人有收集和发表中文方面的信息。

最后要感谢一些朋友的帮忙，本书才得以完成。游韵慈提供了插图；许美云、洪毓伟、连廷佑等人校稿；Neversoft 和 EA 的 Henry，Ili，Johnny，Patrick 在 3ds Max 和 Photoshop 使用上的指导；北京科海电子出版社在排版和出版方面的协助。

彭國倫

realtimegraphics@gmail.com

http：//www.cmlab.csie.ntu.edu.tw/～perng/3D

# 前 言

读者只要具有基本的 C++ 程序设计能力，就可以理解本书内容。编译范例程序需要使用 Visual C++ 和 DirectX SDK，读者可以自行上网下载 Visual C++ Express 与 DirectX SDK。3D 绘图难免要用到一些数学知识，第 20 章把书中所使用的线性代数知识进行了详细整理。本书所使用的英文名词都是根据它们在 3D 绘图中的功能来翻译的，而不是按照字面来翻译。书中偶尔会提到 3ds Max，但它并不是必备工具。

计划学习 Shader 的读者，建议配置支持 Direct3D9 Shader Model 2.0 以上的显示芯片，例如 NVIDIA 5200 和 ATI 9500 以上的芯片，这样就可以执行本书 80% 以上的范例程序。如果打算运行所有范例程序的话，则需要支持 Direct3D10 Shader Model 4.0 的显示芯片，例如 NVIDIA 8x00 和 ATI HD2x00 以上的芯片，并安装 Windows Vista 操作系统。不打算学习 Shader 的读者，使用普通配置的计算机应该就足够了。NVIDIA 和 ATI 都提供了 Shader 开发工具，有兴趣的读者可以自行安装。

本书分成 3 部分，第 1～10 章介绍了传统的固定流程绘图和基本 3D 绘图概念；第 11～18 章是比较高级的 Shader 程序编写，想要深入研究的读者才需要了解这部分内容；第 19～20 章是补充内容，该部分无论对刚入门的读者，还是已经具有 3D 程序设计经验的读者，或多或少都有帮助。

并不是每一位读者学习 3D 绘图都是为了做出炫丽的游戏画面，很多读者学习 3D 绘图只是为了把数据进行图形化显示。每章的开始都会指出哪些部分是应该要了解，不该跳过的；哪些部分属于高级，可以有选择性地阅读。

本书通过范例程序来说明 Direct3D 和 OpenGL 指令的使用方法，并不会在正文中详细说明每个指令及它们所使用的参数。DirectX SDK 说明文件中可以找到 Direct3D 所有指令列表及说明，网络上也有在线的所有 OpenGL 指令说明。虽然大部分指令通过范例就可以了解，但还是建议读者随时参考这些说明文件。

本书所附的光盘中有全部的范例程序源代码以及它们所使用的模型和贴图。有些范例程序在运行时要按照正确的相对路径，才能加载 Shader 和模型数据。读者在复制或者解压范例程序时，要保留相同的目录结构。大部分范例程序都会被编译成使用 Direct3D10 和没有使用 Direct3D10 这两个版本，使用 Windows Vista 和配置 Direct3D10 等级的显示

适配器的读者，才能运行 *_dx10.exe 的范例程序；字尾没有 dx10 的执行文件只支持 Direct3D9 和 OpenGL，无法使用 Direct3D10 的读者可以选用这个版本的执行文件。

范例程序所使用的模型和贴图都是从网络上下载的。有些是原来就可以免费使用，其他则都征求过原作者的同意才拿来使用，书上会注明它们的出处。读者如果想要再次传播这些数据，也请记得获得原作者同意。

笔者的网页上提供了本书范例程序的下载地址，上面还会有一些不定期的程序错误修正，从网络下载的代码可能和书中的有一些差别。下面列出的是前面提到的网页链接。

本书网页：

http：//www.cmlab.csie.ntu.edu.tw/~perng/3D

DirectX SDK 下载地址：

http：//www.microsoft.com/downloads

OpenGL 指令说明：

http：//www.opengl.org/sdk/docs/man/

NVIDIA FX Composer 和 CG 的下载地址：

http：//developer.nvidia.com

Render Monkey 的下载地址：

http：//ati.amd.com/developer/rendermonkey/downloads.html

太阳系行星图主要来自以下两个网页：

http：//www.nasa.gov/

http：//planetpixelemporium.com/

环境贴图 Cubemap 和高动态范围 HDR 图片来自：

http：//www.debevec.org/

模型和其他贴图主要来自：

http：//www.turbosquid.com/

彭國倫

realtimegraphics@gmail.com

http：//www.cmlab.csie.ntu.edu.tw/~perng/3D

# 目 录

# 第 1 章

# 计算机绘图简介

本章将先对绘图芯片的成像过程作简单介绍，还会解释一些常见的中英文名词。本章没有任何范例程序，正式的程序教学将从第 2 章开始。

## 1.1 图像的数字化

首先介绍数字化的概念，计算机里存储的所有数据都是数字化后的一堆数字。就以文字为例，中文英文都一样，它会把每个字母都给定一个数字代码来存储。在计算机绘图里，则是把颜色的信息转换成数字，转换的原则是使用一个 0～1 之间的数值来显示不同色光的强度。先来看一个黑白图片的例子，如图 1.1 所示。比较暗的部分就是光比较弱的部分，它的数值比较接近于 0；比较亮的部分光比较强，它的数值也就会比较接近于 1。

图 1.1　黑白图像，偏亮的地方数值会接近于 1，偏暗的地方数值会接近于 0

所有不同颜色的光线，都可以把它们分解出红、绿、蓝三种基本色光，也就是所谓的三原色。把三种基本色光使用不同的强度混合起来，就可以调配出不同的颜色。彩色图像的数字化，就是把图片颜色分解成三种基本色光后，再分别记录三种色光的强度。

解释概念时，通常会以 0～1 的数值范围来描述色光的强度。在硬件实现上，普遍使

用 8 bits 精确度，也就是 1 Byte 的整数来存储一个基本色光，而不会使用 32 bits 的浮点数来存储 0～1 之间的数值，所以在内存中实际上看到的是 0～255 的整数。但是，使用 0～1 的数值范围来作概念解释并没有什么不妥，因为硬件在实现与图像相关的计算时，仍然是把它们视为 0～1 的数字来思考，把 0～255 的整数对应成 0～1 的数值。

计算机屏幕上的像素数量是有限的，常见的分辨率有 1024*768、1280*1024 等。现在，几乎大家都有数码相机，大家也都知道像素越高，画面会越精细。身为程序设计人员还应该要了解的是，不同分辨率所需要使用的内存量。以常见的 1024*768 分辨率来说，如果每个像素各使用 1 Byte 来存储三种色光的话，就需要用 1024*768*3 Bytes = 2.25 MB。硬件实现上经常需要把每个像素都放在内存中刚好是以 4 为倍数的地址才能运行，所以虽然只需要记录红绿蓝三种色光，还是会强迫每个像素使用 4 Bytes 的内存，实际的内存使用量会变成 1024*768*4 Bytes = 3 MB。多出来的那个 Byte 并不一定就会被白白浪费，很多时候可以把它拿来存储透明值 Alpha。Alpha 是个很有用的东西，后面有一章会专门介绍 Alpha。

现在的绘图芯片会提供很多不同的精确度和格式来存储颜色，例如，在 3D 绘图刚开始萌芽的时期，经常使用 2 Bytes，也就是 16 bits 来存储一种颜色，红绿蓝三种色光各使用 5 bits，或者是多分配 1 个 bit 给绿色使用，16 bits 的模式比较节省内存空间。现在的硬件还可以使用 32 bits 的浮点数来存储 1 种基本色光，在这种模式下，1 种颜色 RGB 最少需要用 4*3 = 12 Bytes；如果再包含 Alpha 值，就会变成 4*4 = 16 Bytes。这时候，1024*768 分辨率的画面就需要用 12 MB 的内存。

早期彩色计算机刚开始流行的时候，经常使用色盘来存储颜色。一个色盘有 256 个位置，可以放 256 种不同的颜色；画面内存只要记录 1 Byte 的索引数值，代表要取出色盘里的第几号颜色来用就可以了。这个方法的缺点是，同一个画面中只能同时出现 256 种不同的颜色。色盘模式在现代的个人计算机里几乎已经完全被淘汰了，而且色盘并不太适合用来做 3D 绘图，这个模式下很难对两种不同颜色进行内插计算，因此本书不会演示这种显示模式。

## 1.2 3D 画面的产生

照片是一张 2D 的平面，它所显示的是四周环境投影到相机底片上的结果。计算机绘图所做的也是类似的事情，它们的差别在于计算机绘图所使用的场景是虚拟的，镜头本身也是个虚拟的镜头。

简单地说，3D 虚拟环境就是用一堆几何图形来描述现实世界的物体，通常所选用的几何图形种类是三角形，如图 1.2 所示。就以一个箱子来看，首先要确定箱子 8 个顶点的位置，再描述这 8 个顶点所组成的 6 个平面，再用 12 个三角形去排列这 6 个平面。画球的时候，则要用到许多个小三角形，把它们沿着球的表面排放来近似表示出球形。要画人物的话，同样也要使用一堆小三角形，把它们放在人体表面来堆砌出人形。把

虚拟环境里的所有多边形进行坐标转换，转换到虚拟镜头的坐标系上，这个动作就是在做投影。

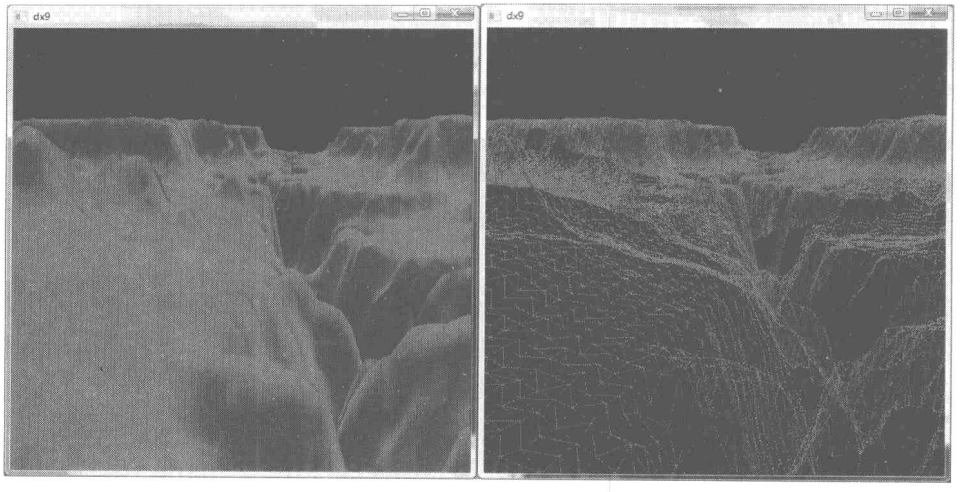



图 1.2　这两张图所使用的是同一个模型，都是使用一堆三角形来近似表示出峡谷的地表。右图只显示三角形的边线，可以明显地看出地表是使用三角形堆砌出来的

现实世界中存在许多不规则形状的物体，不是每样东西都长得像箱子，刚好用 8 个顶点、6 个面就可以搞定。使用面积愈小、数量愈多的三角形，来逼近像人体之类的不规则物体，可以得到更高的质量，不过在绘图时也会花费更多的时间来处理这些三角形。一般的绘图芯片 GPU 常会使用每秒能处理几个三角形为单位来说明它的执行速度，类似于汽车引擎用马力来说明一样。

单纯以每秒能处理几个三角形为单位，其实无法精确地说明绘图芯片的能力，因为三角形可能有大有小，它并不是一个客观的单位。目前，一般还会在规格上说明芯片每秒能处理几个顶点、每秒能够填充多少个像素（称为 Fillrate）、芯片读写内存的速度（也就是内存带宽）等。不过光看这些规格还是无法精确估计出绘图芯片的真正能力，还是要实际使用后才会知道。厂商所列出的硬件规格，都是在理想条件下才会成立，真实世界中不会永远成立。

纯粹使用多边形时，画面上的最终成像结果只有物体的形状而已。想要看到更逼真的物体，可以再使用贴图。简单地说，贴图就是一张有图案的贴纸，把贴纸贴在三角形上，成像的结果就可以更丰富、更逼真。

也许，有的读者会想到一个问题，把物体转换成模型的过程只记录了物体表面的位置，内部怎么好像是空的？这确实是个问题，所以一般的游戏程序都会在移动镜头时做碰撞检测，避免镜头跑进物体空无一物的内部。要记录物体内部并不是做不到，但是不容易做，而且会花费很大量的内存空间。一般会选择忽略物体内部，大部分的情形下也完全不需要显示物体内部。

摄影师在照相时，如果模特儿的后方有一座山，在照片中人物会遮住远山的一角。计算机绘图里，离镜头比较近的三角形，有可能会遮住离镜头比较远的三角形。在真实世界

中，因为人物背后的光线会很自然地被人体遮住，所以镜头里面看不到人物背后的景色。计算机绘图中没有任何现象是自然产生的，所有效果都是要靠软件跟硬件配合才能实现出来。处理前后不同物体会发生重叠的问题，被称为隐藏面消除。

早期还没有 3D 硬件加速时，隐藏面消除的问题一直没有什么好的解决方法。如果想要把它正确地表达出来，就只能把所有的三角形按照它们距离镜头的远近来排序，即先画距离镜头远的东西，再画距离镜头近的东西，还要让比较晚画上去的对象把原来画面上的图像盖过去，这样就可以实现隐藏面消除的效果。这个方法并不是个有效率的做法，光是排序这件事情就会花掉很多时间，而且在某些情形下，两个三角形可能会相交，无法做出正确的排序。

现在的绘图芯片都支持 Z Buffer 隐藏面消除，它可以在不需要排序的前提下，实现隐藏面消除的功能。Z Buffer 测试事实上并不是什么特别神奇的技巧，它使用的基本是个很简单的暴力算法，不过因为是交给专门的硬件来做，才能够达到高效率。

前面提过计算机绘图的画面分辨率是有限的，每个多边形经过转换后，会占用屏幕上面的某一块空间，绘图芯片 GPU 要对这块空间来着色。使用 Z Buffer 来做隐藏面消除时，GPU 除了需要一块内存空间来存放着色后的画面数据之外，还会使用另一块内存空间来记录画面上每个像素到镜头之间的距离，它就是 Z Buffer。

每当有一个新的三角形转换到屏幕坐标后，在开始着色之前，绘图芯片会计算新像素到镜头之间的距离，把它们和已经存在于画面上同样位置的像素来作比较。如果新的像素离镜头比较远，就可以把它舍弃不用，保留旧的像素。如果新像素离镜头比较近，就可以用它把旧的像素替换掉（如图 1.3 所示）。

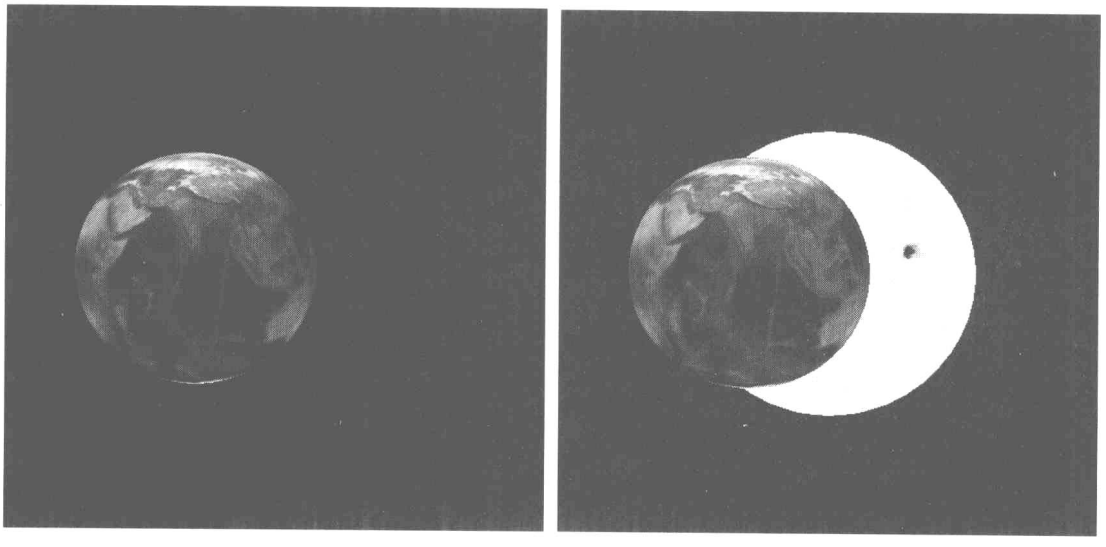


图 1.3　左图先画出距离镜头比较近的地球，右图则是下一步接着画出离镜头比较远的太阳的结果。在画太阳时，跟地球重叠的部分会因为像素离镜头比较远，经过 Z Buffer Test 后会被忽略掉而不画出来

之所以它会被称为 Z Buffer 是因为 X 跟 Y 分别代表屏幕坐标系左右跟上下的位置，记录镜头前后方向的轴，就刚好被分配到 Z 轴上。Z 值有时候也被称为深度值