开发人员专业技术丛书

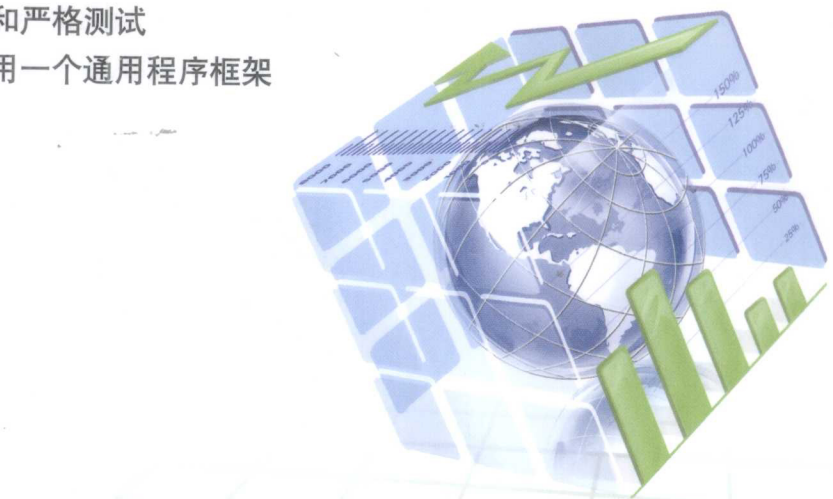
HZ BOOKS
华章科技

# OpenGL编程精粹

## Essential OpenGL Programming

杨柏林 陈根浪 徐静 编著

- 内容精粹：涉及OpenGL 3.1应用最广的概念、原理、技巧
- 实例精彩：实用有趣，交互性强，易于扩展
- 代码精炼：取自实际项目，有流程图和UML图，经过完善和严格测试
- 学习精专：采用一个通用程序框架



附光盘

机械工业出版社
China Machine Press

开发人员专业技术丛书

# OpenGL编程精粹

## Essential OpenGL Programming

杨柏林 陈根浪 徐静 编著

机械工业出版社
China Machine Press

本书讲述如何使用 OpenGL 进行编程。从实际应用的角度出发，全书以 OpenGL 在实际应用中频繁出现的技术重点和难点为讲解内容，完全以对实例的精心讲解贯穿全书，并在各个实例中穿插 OpenGL 和 3D 图形学的相关原理和概念，舍弃 OpenGL 中与实际 3D 图形应用开发关联不大的琐碎知识细节，以一种全新的方式引导读者快速掌握实际开发中所必须掌握的最重要、最实用的概念、原理和编程技巧，事半功倍地进入相关开发领域。本书中的实例代码都是从大量实际应用中精心筛选出来的，并经过适当的修改、完善和严格测试。

本书的适用对象包括欲进入游戏开发、影视特效、仿真系统、虚拟现实与增强现实、图形图像处理、移动图形应用等领域的初、中级程序员和高校与科研机构的相关研究人员；进行毕业设计、课程设计的学生；游戏专业/软件学院/游戏学院学员；对图形编程有兴趣的业余爱好者。本书还可作为高校图形学、游戏程序设计课程的辅助参考资料。



**图书在版编目（CIP）数据**

OpenGL 编程精粹/杨柏林等编著. —北京：机械工业出版社，2010.9
（开发人员专业技术丛书）

ISBN 978-7-111-31576-6

Ⅰ.O…　Ⅱ.杨…　Ⅲ.图形软件，OpenGL－程序设计　Ⅳ.TP391.41

中国版本图书馆 CIP 数据核字（2010）第 160720 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：李东震
北京京师印务有限公司印刷
2010 年 9 月第 1 版第 1 次印刷
186mm×240mm · 23.5 印张
标准书号：ISBN 978-7-111-31576-6
　　ISBN 978-7-89451-647-3（光盘）
定价：59.00 元（附光盘）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

# 序

目前，基于 OpenGL 开发的 3D 应用程序已经被广泛应用到科学计算可视化、军事仿真、体育仿真、虚拟漫游、增强现实等领域。近年来，已有多种 OpenGL 编程的书籍出版。相比这些书籍，除了相关图形学的基本理论介绍之外，本书更加突出书中实例的原创性、代表性、丰富性、实用性和技术的前沿性，力图全面系统、深入浅出，并充分考虑各类读者对内容的可接受性，具体为：

1）抓住最新前沿技术：全面介绍 OpenGL 的最新版本中所涉及的与实际应用关联度大的技术细节，涵盖 GPU 编程技术、移动平台上的 OpenGL ES 图形处理技术。

2）面向实践应用：全书简化理论知识的介绍，主体上讲解实例的设计与实现。全书以 OpenGL 在实际应用中频繁出现的技术重点和难点为讲解内容，完全以对实例的精心讲解贯穿全书，以一种全新的方式引导读者快速掌握实际开发中所必须掌握的最重要、最实用的概念、原理和编程技巧，事半功倍地进入相关开发领域。

3）良好的代码风格：书中代码注以序号，结合具体的代码语句讲解相关原理和概念，全书代码保持一种良好的风格，尤其注意清楚明了的注释并配有清晰的流程图和操作步骤抓图等。

4）实例原创、实用：力求通过翔实有趣、操作性强、有实用价值的例子帮助读者领会复杂、枯燥的原理与概念。本书中的实例代码都是从大量实际应用中精心筛选出来的，并经过适当的修改、完善和严格测试，它们不但有较大的教学和研究价值，而且可复用性高，很多都能拿到读者遇到的实际项目中“即插即用”。

5）低门槛、易上手：精心开发一个通用的程序框架，书中每个实例都采用该框架，读者只需理解其运行流程，不必深陷复杂、枯燥的细节，就可看懂每个实例。

本书的适用对象包括欲进入游戏开发、影视特效、仿真系统、虚拟现实与增强现实、图形图像处理、移动图形应用等领域的初、中级程序员和高校与科研机构的相关研究人员；进行毕业设计、课程设计的学生；游戏专业/软件学院/游戏学院学员；其他对图形编程有兴趣的业余爱好者。本书还可作为高校图形学、游戏程序设计课程的辅助参考资料。

编　者

2010 年 4 月

# 前 言

OpenGL（Open Graphics Library）即“开放的图形程序接口”，它于1992年由SGI公司开发，得到广大硬件和软件厂商的支持和参与，并成为该领域的工业标准。由于OpenGL是一个与硬件无关的软件接口，同时可以在不同的平台如Windows 95、Windows NT、UNIX、Linux、MacOS、OS/2之间进行移植。因此，支持OpenGL的软件具有很好的移植性，可以获得非常广泛的应用。随着OpenGL版本升级，功能不断增强，基于OpenGL开发的应用程序已经从传统的简单三维模型绘制扩展到复杂场景绘制，并被广泛应用到科学计算可视化、军事仿真、体育仿真、虚拟漫游、增强现实、商务虚拟展示等领域。

与已有的市面上多种关于OpenGL的书籍相比，本书在内容安排上有3个特点：

1）从最基础的平台和程序框架开始，并介绍了相关图形学基础知识，使读者能够快速进入本书内容；

2）提供的OpenGL编程技术内容翔实，范围从OpenGL的基本绘制技术延伸到场景优化和高级特性，并扩充了目前的研究热点；

3）针对每部分内容，都有典型例子来说明，从而使读者能够快速理解所讲授的内容。

本书共分28章。第1、2章，概述了OpenGL图形开发的基本方法和基本程序框架，通过这两章的学习读者能够实现一个最简单的OpenGL程序。第3章～第9章，主要介绍了进行三维绘制所需要的基本要素，包括基本图形绘制、颜色、光照和材质、显示列表、位图和图像、纹理映射以及混合等方面。第10章～第21章详述了对三维场景的优化方法及其相关高级特性，主要涉及字体、反走样、雾、各种几何处理、多种高级纹理映射和光照技术、阴影和拾取。第22章～26章详述了在各种3D系统开发中用到的典型技术，包括动画、摄影机漫游、天空和地形绘制以及粒子系统。第27章和第28章分别介绍了目前得到广泛应用的OpenGL的着色语言和在移动设备上使用的移动图形库—OpenGL ES。

本书第4，6，9，11，12，13，16和28章由浙江大学宁波理工学院陈根浪老师编写；其余部分由浙江工商大学杨柏林副教授编写；全书的示例程序由徐静女士调试实现。在编写过程中，作者参考了国内外有关OpenGL的书刊和文献资料，并得到了《OpenGL游戏编程》原作者之一的王琬先生的大力支持和协助，特此表示感谢。

由于时间仓促和水平有限，书中疏漏之处在所难免，希望读者提出宝贵意见，以便再版时修正。如有任何问题，请发邮件至programming. game@ gmail. com。

编 者

2010年4月

# 目　录

# 第1章　OpenGL图形开发快速入门

欢迎来到3D图形编程的世界，你将会发现这是一个非常有趣而且富有挑战的领域。

当你看到游戏中精彩绝伦而且富有震撼效果的场面时，是否有过这样的疑问——它究竟是怎样完成的？它到底使用了哪些技术？……你可能有太多的疑问，请不要着急，本书将会通过一个个具体的实例来讲解使用OpenGL进行图形开发的方法，使你逐步地了解3D图形开发中的一些实用技术，为你今后在这个领域的学习和研究抛砖引玉。

真正开始之前，我们需要对OpenGL的基础知识以及3D图形学理论有初步的认识，它们有助于你更好地理解后续章节的知识。

## 1.1　OpenGL基础知识

OpenGL的英文全称为Open Graphics Library，即*开放式图形库*。它为程序开发人员提供了一个图形硬件接口，是一个功能强大、调用方便的底层3D图形函数库。OpenGL适用于从普通PC到大型图形工作站等各种计算机，并可与各种主流操作系统兼容，从而成为占据主导地位的跨平台专业3D图形应用开发包，进而也成为该领域的行业标准。

### 1.1.1　OpenGL的历史

OpenGL最初是美国SGI公司为其图形工作站开发的、独立于窗口操作系统和硬件环境的图形开发环境。其目的是将用户从具体的硬件考虑中解放出来，完全不用理解这些系统的结构和指令系统，只要按照规定的格式编写应用程序就可以在任何支持该语言的硬件平台上执行。它源于IRIS GL，在跨平台移植过程中发展成为OpenGL。

SGI在1992年7月发布了OpenGL的1.0版本，之后它便成为行业标准，并由成立于1992年的OpenGL ARB（Architecture Review Board）评审委员会负责其后续版本的制定。该委员会由主流的图形开发厂商组成，主要包括有SGI、Intel、IBM、nVIDIA、ATi、Microsoft、Apple等公司。它历经了几个重要的版本：1.1、1.2、1.3、1.4、1.5、2.1、3.0，目前最新版本为4.0。

2010年3月10日，Khronos的OpenGL ARB发布了OpenGL 4.0，把GLSL 4.0升级到了OpenGL着色语言（Shading Language），使开发者能够使用最新一代的GPU加速，并提高编程的灵活性。OpenGL 4.0增强了与OpenCL之间的通用性，提升密集型视觉计算性能。OpenGL 4.0还可向下兼容OpenGL 3.2以后的标准，使开发者能够根据具体需要选择最新API或保留现有OpenGL代码。

OpenGL 4.0特别为程序开发人员设计了大量全新特性，包括：

- 两个新的着色阶段，让GPU接手几何嵌饰（tessellation）工作，不再由CPU完成。
- 每采样片段着色器与可编程片段着色器输入位置，提高渲染质量和反锯齿效果。
- 数据绘图由OpenGL或者OpenCL之类的外部API负责生成，无需CPU干预。

- 着色子例程（subroutine），显著提升编程弹性。
- 通过新增的对象类型采样对象（sampler object）实现纹理状态和纹理数据的分离。
- 64 位双精度浮点着色器操作和输入/输出，提升渲染精度和质量。
- 性能上的改善，包括实例化几何着色器、实例化阵列和新的计时器序列（timer query）。

### 1.1.2 OpenGL 的特点

OpenGL 作为一个性能卓越的图形应用编程接口（API），适用于广泛的计算机环境，并已成为目前的三维图形开发标准，是从事三维图形开发工作的技术人员所必须掌握的开发工具。OpenGL 的应用领域十分广泛，如军事、电视广播、CAD/CAM/CAE、娱乐、艺术造型、医疗影像、虚拟现实等。它具有以下特点。

- **图形质量好、性能高**。无论是三维动画、CAD，还是视觉模拟、可视化计算等，都利用了 OpenGL 高图形质量、高性能的特点。这个特点使得程序开发者在广播、CAD/CAM/CAE、娱乐、医学图像和虚拟现实等领域中创造和显示出难以想象的 2D 和 3D 图形。
- **行业标准**。OpenGL ARB 作为独立的联合委员会，制定规范文档（Specification）。随着业内厂商的支持，OpenGL 成为唯一真正开放的、独立于供应商的、跨平台的标准。
- **稳定性**。OpenGL 能够在各种平台上执行，而且 OpenGL 高版本兼容低版本，保证了已经开发的应用程序不会失效。
- **可移植性和可靠性**。利用 OpenGL 技术开发的应用图形软件与硬件无关，只要硬件支持 OpenGL API 标准就行了，也就是说，OpenGL 应用程序可以运行在支持 OpenGL API 标准的任何硬件上。但是，硬件是不断变化的，OpenGL 如何保持可移植性呢？OpenGL 扩展（OpenGL Extension）正是为这一目的而设计的。厂商只要提供 OpenGL 扩展，就可以轻松实现硬件特有的功能。利用 OpenGL 扩展，OpenGL 实现者（OpenGL Implementer）也可以添加新的处理算法。
- **可扩展性**。OpenGL 是低级的图形 API，它具有充分的可扩展性。许多 OpenGL 开发商在 OpenGL 核心技术规范的基础上，增强了许多图形绘制功能，从而使 OpenGL 能紧跟最新硬件发展和计算机图形绘制算法的发展。对于硬件特性的升级可以体现在 OpenGL 扩展机制以及 OpenGL API 中，一个成功的 OpenGL 扩展会被融入在未来的 OpenGL 版本之中。通过这种方法，程序开发者和硬件厂商能够在正常的产品周期中组合出新的产品。
- **可适应性**。基于 OpenGL API 的图形应用程序可以运行在许多系统上，包括各种用户电子设备、PC、工作站以及超级计算机。由此，OpenGL 应用程序可以适应开发人员选择的各种目标平台。
- **易用性**。OpenGL 具有良好的结构、直观的设计和逻辑命令。与其他图形程序包相比，OpenGL 只有很少的代码，因此执行速度快。另外，OpenGL 封装了有关基本硬件的信息，使得开发者无须针对具体的硬件特征进行设计。

## 1.2 3D 图形学理论入门

前面我们介绍了有关 OpenGL 的基础知识，作为一名 3D 图形开发人员，了解一些 3D 图形学

的理论会对以后的学习有一定的帮助作用，虽然它不是 OpenGL 初学人员的必需条件。同时你还要知道我们介绍的东西只是 3D 图形学的其中一小部分，你可以通过其他相关资料了解 3D 图形学的深入内容。

### 1.2.1 点

点是图形中最基本的几何对象。没有了点，电脑就不知道应该将物体放置到什么位置，也不知道如何让物体移动。一般地，利用直角坐标系来确定物体在屏幕上的位置，直角坐标系包含一个 $x$ 轴和一个 $y$ 轴，一个点就可以用两个轴上的坐标表示，如 $(x,y)$。其中原点的坐标为 (0,0)，位于两个坐标轴相交的地方。从原点出发，向右是 $x$ 轴的正方向，向左是 $x$ 轴的反方向；同样，向上是 $y$ 轴的正方向，向下是 $y$ 轴的反方向，如图 1-1 所示。

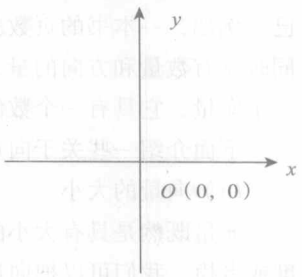


图 1-1 直角坐标系

有了点，我们就可以用坐标来确定物体的位置如 (0,0)，表示物体起始位于原点位置。由于在编程中我们经常要计算两个点之间的距离，下面给出两点间的距离公式。

假设在 2D 空间中有两个点 $P_1$ 和 $P_2$，其坐标分别为 $(X_1,Y_1)$ 和 $(X_2,Y_2)$，那么它们之间的距离为：

$$P_1P_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

上面讲的是在 2D 空间中的点，同样我们可以建立 3D 空间内的坐标系，方法很简单，只需在原来 $x$ 轴和 $y$ 轴的基础上添加一个 $z$ 轴即可。需要注意的是，在这里有两种法则：左手法则和右手法则，从而建立起来的坐标系称为左手坐标系和右手坐标系。在这两种坐标系中，正 $x$ 轴指向右面，正 $y$ 轴指向上面。通过沿正 $x$ 轴方向到正 $y$ 轴方向握拳，大拇指的指向就是相应坐标系的正 $z$ 轴的指向，如图 1-2 所示。

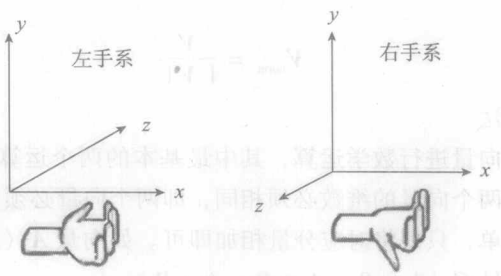


图 1-2 左手系和右手系

从上图可以看出来，左手系和右手系的主要区别就是 $z$ 轴正向的朝向不同，一个朝向屏幕里面，一个朝向屏幕外面，因此左、右手坐标系可以相互转换，最简单的方法是只翻转一个轴的符号，注意如果同时翻转两个轴的符号，结果和不翻转是一样的。

左、右手坐标系没有好坏之分，在不同的研究领域和不同的背景下，人们会选择不同的坐标系。由于大多数程序开发人员都采用右手系，而且 OpenGL 也采用这种坐标系，所以本书将采用

右手坐标系。

同理，3D 场景中的点用 $x$、$y$、$z$ 这 3 个坐标值来表示，如（0，2，3）。

同样，3D 空间内两点 $P_1$（$X_1$，$Y_1$，$Z_1$）和 $P_2$（$X_2$，$Y_2$，$Z_2$）的距离公式为：

$$P_1P_2 = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$$

### 1.2.2 向量

在介绍向量之前，我们首先要说一下标量。标量是一个表示数量大小的值，它仅仅是数值而已。例如，一本书的页数就是一个标量，两个城市之间的距离也是一个标量。向量被定义为一个同时具有数量和方向的量，也就是说，向量等于标量加上一个方向。例如，汽车的行驶速度就是一个向量，它具有一个数值表示大小（叫做速率），同时它还有一个方向用于表示行驶的方向。

下面介绍一些关于向量的性质和运算。

（1）向量的大小

向量既然是具有大小的，我们有时会需要计算该数量，其实计算向量的大小就是求向量的长度或者模。我们可以把向量 $\boldsymbol{N}$ 的长度记作：$|\boldsymbol{N}|$。在 3D 空间中该数值的计算如下：

$$|\boldsymbol{N}| = \sqrt{N_x^2 + N_y^2 + N_N^2}$$

其中，$N_x$、$N_y$ 和 $N_z$ 是向量 $\boldsymbol{N}$ 的 $x$、$y$ 和 $z$ 分量。

（2）向量的归一化

我们知道向量具有方向，它的方向描述了向量在空间中的指向。对于许多向量，我们只关心它的方向而不关心其大小，如某一个平面的法线方向是什么，在这样的情况下，使用单位向量非常方便。单位向量就是大小为 1 的向量，单位向量经常也被称做标准化向量。

求一个向量的单位向量的过程，叫做向量的归一化。它的运算法则为：

对于一个非零向量 $\boldsymbol{V}$，可以用该向量除以它的大小（或模）即可得到该向量的归一化向量，如下所示：

$$\boldsymbol{V}_{\text{norm}} = \frac{\boldsymbol{V}}{|\boldsymbol{V}|}$$

（3）向量的加法和减法

可以像对标量一样对向量进行数学运算，其中最基本的两个运算是向量加法和向量减法。需要注意的是，参加运算的两个向量的维数必须相同，即两个向量必须都是 2 维、3 维或 4 维等。向量加法的运算规则非常简单，只需将对应分量相加即可。如向量 $\boldsymbol{A}$（$A_x$，$A_y$，$A_z$）和向量 $\boldsymbol{B}$（$B_x$，$B_y$，$B_z$）相加，其和为向量 $\boldsymbol{C}$（$A_x + B_x$，$A_y + B_y$，$A_z + B_z$）。

同样，向量的减法只需将对应的分量相减即可。如向量 $\boldsymbol{A}$（$A_x$，$A_y$，$A_z$）和向量 $\boldsymbol{B}$（$B_x$，$B_y$，$B_z$）相减，得到向量 $\boldsymbol{C}$ 为（$A_x - B_x$，$A_y - B_y$，$A_z - B_z$）。

（4）向量与标量的乘法运算

虽然标量和向量不能相加，但它们能相乘，结果将得到一个向量，它与原向量平行，但长度不同或方向相反。

标量与向量的乘法非常简便，将向量的每个分量都与标量相乘即可。而且标量与向量乘的顺

序并不重要，但经常把标量写在左边。例如：用标量值 2 乘以向量 $N$，其结果为一个向量，其方向与向量 $N$ 的方向一致，但其长度变为原来的 2 倍。相似地，用标量值 -1 乘以该向量 $N$，其结果向量的大小与 $N$ 相同，但方向变为 $N$ 的相反方向。

（5）点乘运算

向量和向量相乘可以有两种不同的类型，在这里我们首先介绍点乘（也称做内积）。

术语“点乘”来自于记法中 $a \cdot b$ 的点号，与标量与向量的乘法一样，向量点乘的优先级高于加法和减法。标量乘法和标量与向量的乘法经常可以省略乘号，但在向量点乘中不能省略点乘号。

向量点乘就是对应分量乘积的和，其结果是一个标量，在 2D 和 3D 中其运算公式为：

$$\boldsymbol{A} \cdot \boldsymbol{B} = A_x B_x + A_y B_Y \quad (\boldsymbol{A}, \boldsymbol{B} \text{ 都为二维向量})$$

$$\boldsymbol{A} \cdot \boldsymbol{B} = A_x B_x + A_y B_y + A_z B_z \quad (\boldsymbol{A}, \boldsymbol{B} \text{ 都为三维向量})$$

一般来说，点乘的结果描述了两个向量的“相似”程度，结果越大两个向量越相近。点乘等于向量大小与向量夹角的余弦值的积：

$$\boldsymbol{A} \cdot \boldsymbol{B} = |A| \ |B| \cos\theta$$

由上面公式即可计算两个向量之间的夹角：

$$\cos\theta = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{|A| \ |B|}$$

然后，只要再进行以下反余弦函数计算就能求得角度值了。如果向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的长度都为 1，那么公式可以简化为：

$$\cos\theta = \boldsymbol{A} \cdot \boldsymbol{B}$$

这就明显简化了运算，根据此时的点乘结果可以得出一些有用的性质，如下：

$\boldsymbol{A} \cdot \boldsymbol{B} = 0$，则说明 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 之间的夹角 $\theta$ 等于 90°；

$\boldsymbol{A} \cdot \boldsymbol{B} > 0$，则说明 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 之间的夹角 $\theta$ 小于 90°；

$\boldsymbol{A} \cdot \boldsymbol{B} < 0$，则说明 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 之间的夹角 $\theta$ 大于 90°。

（6）叉乘运算

另一个向量乘法称作叉乘或叉积，仅可应用于 3D 场景，和点乘不一样的是，向量叉乘得到的是一个向量。叉乘运算在 3D 图形学中应用于许多领域，如碰撞检测、光照和物理计算等。给定两个 3D 向量 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，其叉积向量的大小为 $|\boldsymbol{A}| \ |\boldsymbol{B}| \sin\theta$，其方向垂直于向量 $\boldsymbol{A}$ 和向量 $\boldsymbol{B}$，如图 1-3 所示。

有两个垂直于 $\boldsymbol{A}$、$\boldsymbol{B}$ 所在平面的方向：向上和向下。你可以用右手定则来判断一下叉乘得到的值是哪个方向。把你的手放在 $\boldsymbol{A}$、$\boldsymbol{B}$ 的交点处，食指指向 $\boldsymbol{A}$，弯曲其他手指指向 $\boldsymbol{B}$，这时候拇指所指的方向即为 $\boldsymbol{A} \times \boldsymbol{B}$。

需要注意的是叉乘不满足交换率，即 $\boldsymbol{A} \times \boldsymbol{B} \neq \boldsymbol{B} \times \boldsymbol{A}$，实际上 $\boldsymbol{A} \times \boldsymbol{B} = -(\boldsymbol{B} \times \boldsymbol{A})$。

由于叉乘可以计算出垂直于两个原向量的一个向量，所以经常用来计算面的法向量，因为任意两个 3D 向量都可以确定一个平面。

在实际运用中，经常会使用到矩阵运算（这将在后面介绍），即两个向量叉乘的另一个公式：

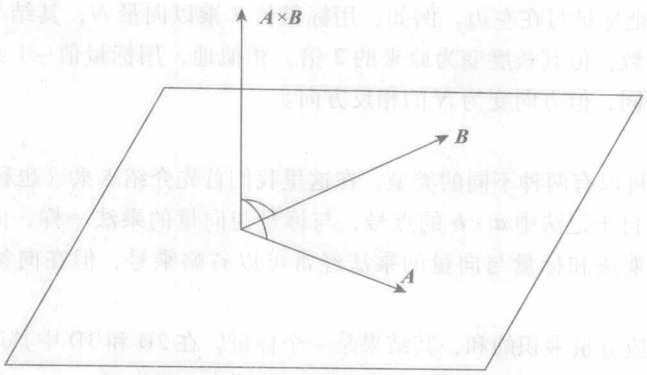



图 1-3 向量叉乘 $\boldsymbol{A}\times\boldsymbol{B}$

$$\boldsymbol{A}\times\boldsymbol{B}=(A_yB_z-A_zB_y,\ A_zB_x-A_xB_z,\ A_xB_y-A_yB_x)$$

其中，$A_yB_z-A_zB_y$、$A_zB_x-A_xB_Z$、$A_xB_y-A_yB_x$ 分别为 $\boldsymbol{A}$、$\boldsymbol{B}$ 叉乘向量的 $x$、$y$ 和 $z$ 分量。

### 1.2.3 矩阵

矩阵是3D数学的重要基础，对于坐标系的转换和物体的变换都要用到矩阵，因此在这里有必要讨论一下矩阵的性质和运算。

在线性代数中，矩阵就是以行和列的形式组织的矩形数字块。我们可以使用 $m\times n$ 来定义矩阵的尺寸，它表示该矩阵 $m$ 行和 $n$ 列。我们可以根据行和列的值引用矩阵中指定位置的元素值，通常用第 $i$ 行第 $j$ 列来表示。对于 $3\times3$ 矩阵 $\boldsymbol{M}$，可以表示如下：

$$\boldsymbol{M}=\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

其中，$m_{00}$为矩阵的第一行第一列元素值。

下面介绍几个特殊矩阵。

（1）方阵和对角矩阵

行数和列数相同的矩阵称作方阵，方阵中的对角线元素就是方阵中行号和列号相同的元素。比如上述矩阵 $\boldsymbol{M}$ 的对角线元素为 $m_{00}$、$m_{11}$和 $m_{22}$。那么其他元素为非对角元素，如果所有非对角元素都为0，那么称这种矩阵为对角矩阵。例如：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

（2）单位矩阵

单位矩阵是一种特殊的对角矩阵。$n$ 维单位矩阵记作 $\boldsymbol{I}_n$，它是 $n\times n$ 的矩阵，其对角线元素为1，其他元素为0。例如，$3\times3$ 单位矩阵为：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

单位矩阵非常特殊，如果用一个矩阵和一个相应的单位矩阵相乘，其结果还是原来的矩阵，稍后我们会讲解矩阵的乘法运算。

下面介绍有关矩阵的运算。

（1）矩阵的加法和减法

矩阵的加法和减法运算非常简单，只需将对应位置上的元素进行相加或相减，其结果作为结果矩阵在该位置上的值。但是，首先要确定参加运算的矩阵具有相同的阶数，否则运算将无法进行。

（2）矩阵的乘法

矩阵的乘法有三种情况，下面分别进行说明。

1）标量和矩阵相乘。

标量 $k$ 和矩阵 $\boldsymbol{M}$ 相乘，其结果是一个和 $\boldsymbol{M}$ 维数相同的矩阵，其元素值为 $k$ 乘以 $\boldsymbol{M}$ 中的每个元素，其公式如下：

$$k\boldsymbol{M} = k\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} km_{00} & km_{01} & km_{02} \\ km_{10} & km_{11} & km_{12} \\ km_{20} & km_{21} & km_{22} \end{bmatrix}$$

2）向量和矩阵相乘。

因为向量可以看作是一行或一列的矩阵，因此向量和矩阵也是可以进行相乘运算的，在这里只有两种组合是允许的，即行向量左乘矩阵时，结果是行向量；列向量右乘矩阵时，结果是列向量。其中行向量的列数要和矩阵的行数相等，列向量的行数要和行向量的列数相等。另外两种组合是不允许的，即不能用行向量右乘矩阵，也不能用列向量左乘矩阵。其运算规则为：

$$\begin{bmatrix} x & y & z \end{bmatrix}\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix} = \begin{bmatrix} xm_{00} + ym_{10} + zm_{20} & xm_{01} + ym_{11} + zm_{21} & xm_{02} + ym_{12} + zm_{22} \end{bmatrix}$$

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xm_{00} & ym_{01} & zm_{02} \\ xm_{10} & ym_{11} & zm_{12} \\ xm_{20} & ym_{21} & zm_{22} \end{bmatrix}$$

3）矩阵和矩阵相乘。

在某些情况下，两个矩阵可以相乘。一个 $m \times n$ 的矩阵 $\boldsymbol{A}$ 能够和一个 $n \times r$ 的矩阵 $\boldsymbol{B}$ 相承，结果为一个 $m \times r$ 的矩阵，需要满足左矩阵 $\boldsymbol{A}$ 的列数要和右矩阵 $\boldsymbol{B}$ 的行数相等。如果不匹配，则 $\boldsymbol{AB}$ 无意义。

矩阵乘法的运算规则为：$m \times n$ 矩阵 $\boldsymbol{A}$ 和 $n \times r$ 矩阵 $\boldsymbol{B}$ 的乘积 $\boldsymbol{AB}$ 为矩阵 $\boldsymbol{C}$，$\boldsymbol{C}$ 的任意元素 $c_{ij}$ 等于 $\boldsymbol{A}$ 的第 $i$ 行向量与 $\boldsymbol{B}$ 的第 $j$ 列向量的点乘结果，即：

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}$$

需要注意的是，矩阵的乘法不满足交换率，即 $\boldsymbol{AB} \neq \boldsymbol{BA}$。

### 1.2.4 变换

在图形程序的开发过程中，我们经常要对物体进行一些几何变换操作，也就是说，将一个几何图形按照一定的规则或规律变换为另一个新的几何图形。在一定程度上可以说，图形的几何变换是计算机图形学系统的核心内容。其中最基本的三种变换是平移、旋转和缩放，下面首先介绍一下在2D空间内的这三种变换，然后将其推广到3D空间中。

#### 1. 二维几何变换

二维图形由点或者直线段组成，其中直线段则由其端点坐标定义。对它进行几何变换可归结为对点或者直线段端点的变换。

（1）平移变换（translation）

对 $XY$ 平面上的点 $P$（$x$，$y$），如果在平行于 $X$ 轴方向上移动 $\mathrm{d}x$ 单位，在平行于 $Y$ 轴的方向上移动 $\mathrm{d}y$ 单位，则可以得到新的点 $P'$（$x'$，$y'$），点 $P$ 和 $P'$ 的关系为：

$$x' = x + \mathrm{d}x, \qquad y' = y + \mathrm{d}y$$

这样的变换称为*平移变换*。

如果点用向量的形式给出，即 $\boldsymbol{P} = [x\ y]$，$\boldsymbol{P}' = [x'y']$，而且 $X$ 和 $Y$ 方向的平移量用行向量的形式表示，即 $\boldsymbol{T}_m = [\mathrm{d}x\quad \mathrm{d}y]$，则平移变换可以写为：

$$[x'y'] = [x\quad y] + [\mathrm{d}x\quad \mathrm{d}y]$$

可以更简单地表示为：$\boldsymbol{P}' = \boldsymbol{P} + \boldsymbol{T}_m$

（2）旋转变换（rotation）

对于 $XY$ 平面上的点 $P$（$x$，$y$），绕原点逆时针旋转某个角度 $\theta$，得到一个新的点为 $P'$（$x'$，$y'$），这种变换称为*旋转变换*。如图1-4所示，根据三角形知识有：

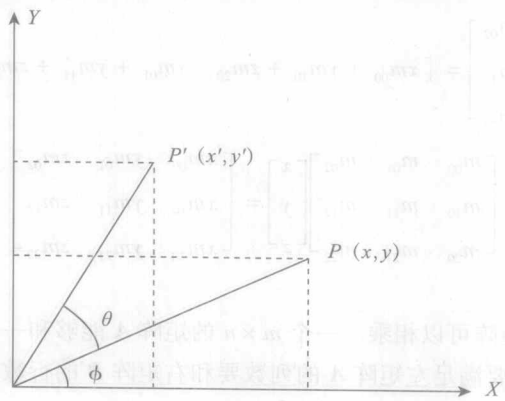



图1-4 旋转变换推导