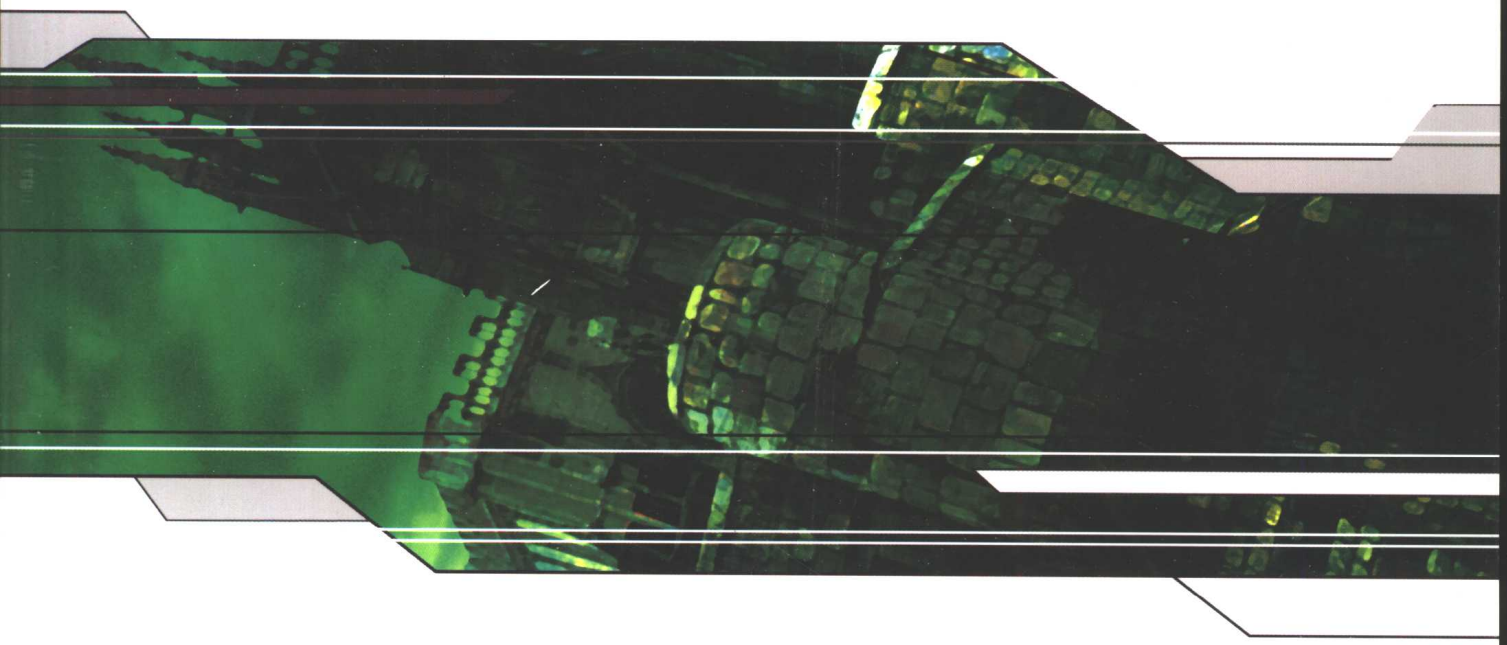
游戏开发与编程系列

3D 游戏编程

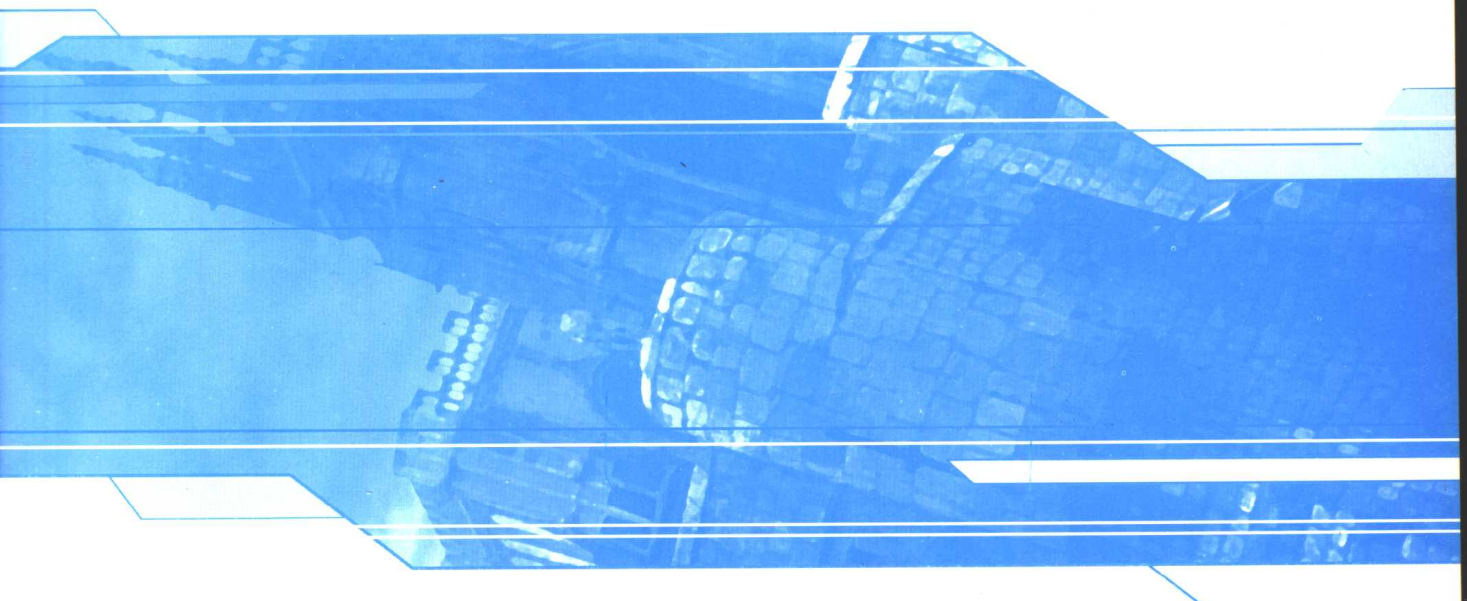


北京希望电子出版社　总策划
杨 青　杨 磊　编

中国科学技术出版社
CHINA SCIENCE AND TECHNOLOGY PRESS
北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

游戏开发与编程系列

3D 游戏编程



北京希望电子出版社　总策划
杨 青　杨 磊　编

中国科学技术出版社
CHINA SCIENCE AND TECHNOLOGY PRESS

北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

图书在版编目（CIP）数据

3D 游戏编程/杨青，杨磊编，　—北京：中国科学技术出版社，2004.4

ISBN 7-5046-3752-1

Ⅰ.3…　Ⅱ.①杨…　②杨…　Ⅲ. ①多媒体—软件工具，Direct X　②游戏—应用程序—程序设计　Ⅳ. ①TP311.56②G899

中国版本图书馆 CIP 数据核字（2004）第 017596 号

书　　名：3D游戏编程
文本著作者：杨青　杨磊
责任编辑：周艳　许慧
出版、发行者：中国科学技术出版社　北京希望电子出版社
地　　址：北京市海淀区中关村南大街 16 号　100081
北京市海淀区知春路甲 63 号卫星大厦三层　100080
网址: www.bhp.com.cn　E-mail:lwm@bhp.com.cn　yb@bhp.com.cn
电话: 010-62520290, 62528991, 62630301, 62524940, 62521921, 62521724（发行）　010-82675588-318,62532258,62562329（门市）
010-82675588-501,82675588-201（编辑部）
经　　销：各地新华书店、软件连锁店
排　　版：希望图书输出中心　张月岭
印 刷 者：北京媛明印刷厂
开本 / 规格：787 毫米×1092 毫米　1/16　21.5 印张　498 千字
版次 / 印次：2004 年 6 月第 1 版　2004 年 6 月第 1 次印刷
印　　数：0001～5000 册
书　　号：ISBN 7-5046-3752-1/TP・238
定　　价：35.00 元

内 容 简 介

本书是介绍如何使用 DirectX 进行游戏编程的书籍。

本书共由 8 章内容构成。其中，第 1 章：3D 入门；第 2 章：渲染和显示，主要介绍了 3D 物体的组成、渲染 3D 场景以及 3D 转换。第 3 章：纹理，讲述了如何使用纹理。第 4 章：灯光、材质和模型，教读者如何使用材质和灯光，在三维场景中建立 Direct3D 支持的灯光效果。第 5 章：雾、雨、雪，通过增加这些自然现象，增强了游戏的可玩性。第 6 章：加入声音。第 7 章：处理输入，可以使用键盘、鼠标、游戏杆、方向盘等等输入设备来控制游戏运行。第 8 章：网络游戏实现。

本书内容清晰、编排合理，为了让读者最好的掌握主要知识点，每章都配有小结和对应的例子。

本书面向广大游戏编程爱好者，也可以作为大专院校相关专业师生的参考书。书中部分实例代码请从http://www.b-xr.com免费下载。

前　言

记得我第一次玩游戏时才上小学，那时一放学回家就和哥哥一起玩魂斗罗，在后来的QUAKE、星际争霸……直到现在的“魔兽争霸”等等，游戏的快乐深深吸引着我。慢慢的，玩游戏已经不能让我满足，想写出自己的游戏成了我的目标。可是，一直想深入学习电脑游戏编程的我，却苦于不能找到好的游戏编程参考书籍，我也看过 Andre LaMothe 写的《Windows 游戏编程大师技巧》一书，学到了不少游戏方面的知识，但是遗憾的是没有讲目前最吸引人的三维游戏编程。于是，我选择学习 DirectX 来实现三维游戏。从一步步学习的过程中，产生了写作此书的念头，于是坚持了下来，才有了本书。

本书是介绍关于如何使用 DirectX 进行游戏编程的书籍，关于电脑游戏的基础原理，可以参考其他更好的书籍。本书从头到尾都是使用实际的程序例子来引导读者进行一步一步的深入学习。在我调试这些程序的过程中，虽然多少次碰到问题，但也充满了乐趣。希望读者也和我一样，当看到第一个三维程序运行出来时，会十分的惊喜。

虽然游戏产业在中国起步比较晚，但是发展非常迅速。我也看到国内游戏不断推陈出新，日益发展。希望任何想学习使用 DirectX 进行游戏编程的读者都能够从本书中获得可以使用的知识。

我要感谢杨磊与我一起完成这本书稿，也感谢希望出版社的陆卫民、赵文博、周艳让此书得以顺利出版。最后，对支持我进行写作的小垚说一声，谢谢。

杨青

目　录

第 1 章　3D 入门

玩了那么多的三维游戏，你是否也想制作自己的三维游戏？如果你希望挑战自己，那么就跟着我们的学习旅程，进入三维游戏制作的世界吧。

1.1　进入 3D 世界

在计算机的 3D 世界中，可以表现出真实世界中所有的事物。虽然，目前受计算机硬件水平和软件开发水平的限制，计算机中的三维世界不如现实世界那样逼真，但是，随着技术的进步，总有一天，会在计算机中表现出真实的虚拟世界。

真实世界中，建筑、风景都具有固定的物理位置，而我们可以四处行走，改变所处的空间。在计算机表示的世界中，需要掌握相应的规则，才能理解计算机中的三维世界。

1.1.1　三维世界的表示

如何才能建立三维世界？回想一下，在二维世界中，比如平面图画，我们是如何做的。使用线条、图形来表示二维世界。记得《清明上河图》描述的一幅繁荣的市井景象。不过，不管多么复杂的二维世界，都是由图形组成；图形又是由线条组成；线条由很多的点组成。点？世界是由点组成的！在三维世界里，这个真理同样成立。

在计算机的三维世界中，如果要显示一个物体，首先关心的就是这个物体怎样由点来构成。然后用这些点来构成多边形，由多边形来构成立体的几何形体。只要你能确定了点的位置、数量、颜色，就可以形成任何你需要的物体。比如迷宫、宫殿、长城，还有外太空。

不过在上面的描述中，忽略了最重要的前提，如何在三维世界中表示点的位置呢？和平面图形类似，三维世界中的点可以在坐标系统中惟一地确定下来。

1.1.2　三维坐标系统

在现实世界中，我们可以用纬度、经度确定某一位置。不过更多的时候，是以每个固定目标作参考，描述其他的目标位置。我们经常说“从这向东 100 米就是邮局”，或者“向前 1 公里，然后右拐就是超级市场”等等。在游戏世界中，我们也必须使用一种手段来表示物体的位置，目前使用的就是三维坐标系统。假设你对二维坐标系统没有了解的话，你就必须自己补一补数学课程了。在二维坐标系统中，使用 x，y 坐标来表示一点在坐标系统中的位置。比如显示画面的电脑屏幕，可以表示成一个二维坐标系统，在屏幕上的位置画一个点，就是对坐标值（x，y）的某一点进行赋值。在三维坐标系统中，引入了 z 坐标来表示物体距离的远近。图 1-1 和图 1-2 分别描述了以下两种三维坐标系统：左手坐标系和右手坐标系。

在左手坐标系中，使用左手来确定三维坐标系。如图所示，左手四指弯曲，拇指四指

垂直，那么四指弯曲的方向就是 x 坐标轴（正方向）绕向 y 坐标轴（正方向）的方向；拇指所指的方向是 z 坐标轴的正方向。而使用右手确定的坐标系也如此：四指弯曲的方向就是 x 坐标轴（正方向）绕向 y 坐标轴（正方向）的方向；拇指所指的方向是 z 坐标轴的正方向。从以上两个图可以看出，两种坐标系不同的地方就是 z 坐标轴的方向。左手坐标系中 z 轴离我们而去，右手坐标系迎我们而来。在游戏中，我们使用哪种坐标系呢？

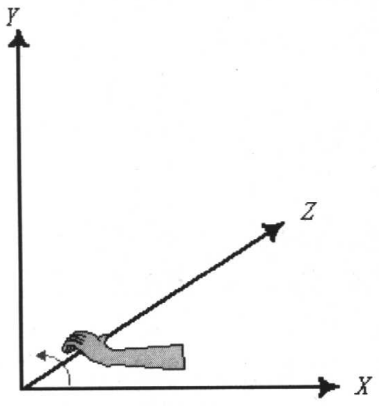


图 1-1　左手坐标系

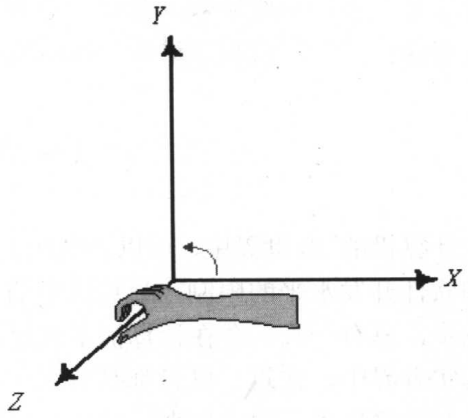


图 1-2　右手坐标系

如果我们面对游戏世界，只会看到我们前面的世界，除非在游戏中扮演有四只眼睛的外星人。景物除了上下左右的相对位置，还有离眼睛的距离决定了看到的最终景物。使用左手坐标系统，可以保证永远使用正的 z 值来表示距离，再远也没有关系。这也比较符合日常的习惯，最重要的是，可以方便程序的处理，毕竟使用负值来表示距离是多余的麻烦。所以在 Direct3D 中，使用了左手坐标系统来表示整个世界的位置。

有了坐标系统，就可以描述三维世界的位置了。要描述某一位置有辆汽车，就必须说“在坐标（x，y，z）处有一辆汽车”，计算机是处理数字的，只有数字才是计算机最了解的语言。

1.1.3　描绘模式

记得我们在前面的例子中形成的三维物体吗？由很多的面构成的。但是真实世界中的大多物体都是表面比较平滑的，如何才能形成真实世界中的平滑表面呢？其实这不用我们来担心，三维编程工具都提供了这样的功能。我们可以直接告诉编程工具说：“给我绘制光滑的表面”，或者像前面的例子“给我绘制平坦的表面”。Direct3D 也提供了这两种描绘模式：光滑描绘模式和平坦描绘模式。使用平坦描绘模式时，由点组成的多边形每一个的表面都是平坦的，组合在一起就是有棱有角的三维物体表面。而使用平滑描绘模式时，会自动计算相邻多边形的相交角度，进行平滑处理，最终形成十分平滑的表面，这种方式就接近真实的世界所展示的物体了。

1.1.4　坐标系转换

为什么要进行坐标系转换？什么是坐标系转换？

在三维游戏中，三维世界中总有物体需要运动吧，至少游戏主角的扮演者希望在三维

世界中任意遨游，最好能够上天入地，或者航行太空。在计算机中的三维世界中体验无法实现的愿望，是三维游戏最吸引人的地方了。好了，让我们的世界动起来吧。想一想，如何做。回忆一下，三维物体都是由基本的点来构成的。我们在三维坐标系中确定了组成物体的每个点的位置，那么，物体的位置也就决定了。要改变物体的位置，比如要把物体向 x 轴的正方向移动 10 个单位的距离，可以这样做：把组成这个物体的所有点的 x 坐标值都加上 10。如果原来有一个点的坐标是（a，b，c），那么这点新的位置坐标就是（a+10，b，c）。计算每个点的新坐标，然后使用前面学过的两种描绘模式之一重新描绘所有的点，就可以在新的位置形成物体。不过这产生了一个问题，如果我们自己去处理每个点，而点的数目很大，比如几十万个（不要吃惊，一个逼真的场景需要的点数目很容易就会达到这个值），我们如何照料得过来。是的，这是一个比较棘手的问题。学过矩阵吗？现在就可以派上用场了。如果没有学过，也没有关系，往下看吧。

平时的数学运算通常是对单个数进行计算的，比如 1+1 =2。但是我们对坐标进行计算时，必须计算坐标的每一个分量，就像这样：（$x1$，$y1$，$z1$）+（$x2$，$y2$，$z2$）= ？。参与计算的数包含了几个分量，这种计算方式我们称为向量计算。

一个点经过怎样的计算才能向右移动 10 个单位。我们假设新的点坐标为（nx，ny，nz），原来的坐标为（x，y，z）。计算式为：（nx，ny，nz）= （x，y，z）* M。式中 M 表示矩阵。在三维坐标系中，我们必须使用 4×4 的矩阵来进行计算。只要确定了矩阵，然后让 3D 支持程序去计算，就行了。这样，我们就不必为每一个点去操心，只要得到我们需要的矩阵，然后告诉 Direct3D，让他去完成物体位置的改变。

下面的运算把一个点的坐标（x，y，z）变为（x'，y'，z'），如图 1-3 所示。

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

图 1-3 矩阵运算

如果把矩阵运算展开为普通的多项式运算，上面的运算等同于图 1-4 所示。

$$\begin{aligned} x' &= (x \times M_{11}) + (y \times M_{21}) + (z \times M_{31}) + (1 \times M_{41}) \\ y' &= (x \times M_{12}) + (y \times M_{22}) + (z \times M_{32}) + (1 \times M_{42}) \\ z' &= (x \times M_{13}) + (y \times M_{23}) + (z \times M_{33}) + (1 \times M_{43}) \end{aligned}$$

图 1-4 矩阵运算展开

上面说的让物体沿某一个坐标轴运动的问题，实际就是寻找一个矩阵，通过运算实现坐标的平移变换。平移变换的矩阵运算如图 1-5 所示。

$$[x'y'z'1]=[x\ y\ z\ 1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ T_x & T_y & T_z & 1\end{bmatrix}$$

图 1-5　平移矩阵

矩阵中的 *Tx* 表示在 *x* 轴方向上的位移大小，如果为正值，表示向 *x* 轴正方向进行位移；如果为负值，表示向 x 轴负方向进行位移。*Ty* 表示在 *y* 轴方向上的位移，*Tz* 表示在 *z* 轴上的位移。

三维旋转变换可以使物体在三维空间中产生旋转。旋转可以是沿着 *x* 轴，沿着 *y* 轴，或是沿着 *z* 轴进行。沿着 *x* 轴旋转的矩阵运算如图 1-6 所示。

$$[x'y'z'1]=[x\ y\ z\ 1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & \sin\theta & 0\\ 0 & -\sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

图 1-6　绕 x 轴旋转

矩阵中的 θ 表示旋转的角度，可以取正负值，表示旋转的方向。

沿 *y* 轴旋转的矩阵运算如图 1-7 所示。

$$[x'y'z'1]=[x\ y\ z\ 1]\begin{bmatrix}\cos\theta & 0 & -\sin\theta & 0\\ 0 & 1 & 0 & 0\\ \sin\theta & & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

图 1-7　绕 *y* 轴旋转

沿 *z* 轴旋转的矩阵运算如图 1-8 所示。

$$[x'y'z'1]=[x\ y\ z\ 1]\begin{bmatrix}\cos\theta & \sin\theta & 0 & 0\\ -\sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

图 1-8　绕 *z* 轴旋转

三维缩放变换可以使物体在三维空间中产生大小的缩放效果，其矩阵运算如图 1-9 所

示。

$$[x'y'z'1]=[x\ y\ z\ 1]\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

图 1-9 缩放矩阵

矩阵中的 *Sx*，*Sy*，*Sz* 分别表示三维物体在 *x* 轴，*y* 轴，*z* 轴方向上的缩放倍数。*S* 值大于 1 表示放大物体，*S* 值小于 1 表示缩小物体，等于 1，既不放大也不缩小。

上面就是三维物体在三维空间中进行运动的数学基础，如果不会进行矩阵运算，没有关系，否则我们要计算机干什么。我们只要清楚什么运动对应什么矩阵，然后，要让物体运动，就简单了。告诉计算机一个确定的矩阵，一切就完成了。比如，想让三维世界旋转，产生“天旋地转”的效果，那么只要把旋转转换中的矩阵告诉计算机就行了，是不是很简单？

1.1.5 观察坐标系

前面知道了如何在计算机中表示三维世界，如何变换三维物体的坐标，但是一直忽略了一个重要的问题：如何去“看”计算机中的三维世界呢？

在真实世界中，我们通过自己的眼睛去看这个世界。当我们来到每一个陌生的地方，可以用眼睛观察周围的环境。我们可以走动，改变看到的环境；可以转动头部，观察不同的方向；当看到一些不能理解的事物时，有时会弯下腰，水平着看，或者干脆把头向着地，倒着看这个世界。不过随着我们渐渐长大，这样的好奇也越来越少了。在计算机的三维世界中，我们也必须学会观察。好吧，就让我们拥有另一双眼睛吧。

在三维坐标中观察世界时，需要确定以下几个问题：

- 在哪个位置观察：需要使用坐标来表示这个观察点的准确位置，这个坐标是在计算机中的世界坐标系中的值。
- 向哪个方向观察。
- 对于观察者来说，代表上方的方向。如同我们头的上部是相对于我们上方，当我们头向下看世界时，就会看到倒立的世界。在计算机三维世界中也是同样道理，我们得指定我们的上方是哪个方向，才能看到正确的三维世界。

当上面几个问题确定以后，实际上就可以建立一个以观察者为圆心的三维坐标系，就把它称作观察坐标系吧。把原来世界坐标系中的所有物体的坐标转换为观察坐标系中的坐标以后，就可以知道每一个三维物体相对于我们的位置了。

图 1-10 显示了世界坐标系和观察坐标系的关系。

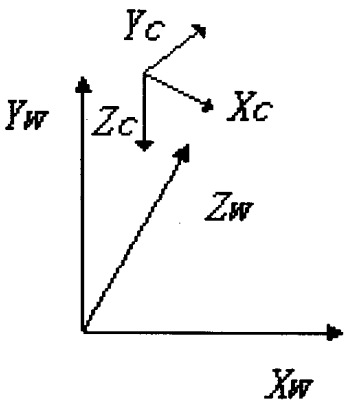


图 1-10　观察坐标系

图 1-10 中 *Xw*，*Yw*，*Zw* 指出的坐标系为世界坐标系，*Xc*，*Yc*，*Zc* 指出的坐标系为观察坐标系。可以想像观察坐标系为一个摄像机确定的世界，最终看到的世界就是这个摄像机能够观察到的世界。那么，想要观察世界坐标的某些景物，就必须把摄像机在世界坐标系中四处移动，确定摄像机的观察位置，同时把握摄像机镜头的方向，这样才能在三维世界任意观察。

为什么要了解这些？当然有用了。虽然不用我们来计算三维物体在观察坐标系中的位置，但是总得告诉计算机我们的摄像机在什么位置了，镜头向着什么方向，我们的上方在哪个方向。计算机只要这 3 个线索，就可以建立起观察坐标系，帮我们建立进行坐标转换了。还记得坐标怎么转换吗？矩阵计算，计算机需要知道转换矩阵。观察坐标转换矩阵如图 1-11 所示。

$$\begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ -(u\cdot c) & -(v\cdot c) & -(n\cdot c) & 1 \end{bmatrix}$$

图 1-11　观察坐标系转换矩阵

矩阵中的 *u* 表示观察坐标系上方的方向；*v* 表示观察坐标系右方的方向；*n* 表示观察方向；*c* 表示观察坐标系原点在世界坐标系中的坐标。

不过这个矩阵不用我们记了，计算机会从我们确定的观察位置、观察方向、观察者上方 3 个信息确定出计算矩阵，自动计算的。想自己算？不会吧。记住，简单的事情我们自己做，复杂的事情让计算机完成吧。

知道了观察坐标系，就可以知道三维世界物体相对于我们的位置了。不过我们玩游戏时都是在电脑屏幕上看到的三维世界，如何使用摄像机把观察到的世界显示到屏幕上，就需要进入下面的学习。

1.1.6　三维透视转换

在显示器屏幕上显示的都是平面图形，不管他们是否具有三维的效果。计算机的三维世界最终需要显示在平面的屏幕上，下面就要讨论如何进行显示。在真实世界中，我们可以看到空间中的任意景物，就是因为从景物发出或者反射的光线能够进入我们的眼睛，从而形成了图像。在计算机的三维世界中，要看到景物，必须通过前面说过的摄像机。可以想像摄像机是我们观察计算机中三维世界的窗口，要想看到景物，景物发出或者反射的光线必须能够通过摄像机的镜头。三维透视就是如何把三维世界的景物进行透视转换，形成可以在电脑屏幕上显示的平面图形。

可以想像三维透视转换（在 Direct X 中，英文称为 Projection Transformation）就是控制三维世界中的摄像机的过程，控制摄像机镜头光圈、焦距，从而得到想看到的世界。可以指定摄像机的可视范围，来决定最终形成的平面图像，如图 1-12 所示。

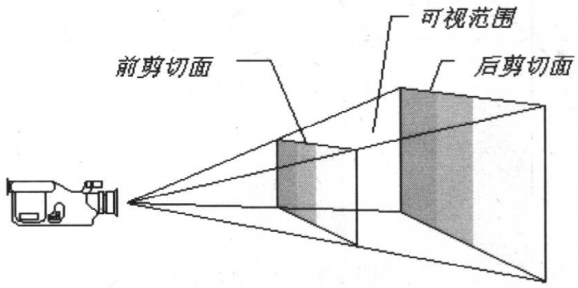


图 1-12　可视范围

在图 1-12 中，表示了摄像机如何观察三维世界。我们可以指定一个后视平面和前视平面，这两个平面都表示成距摄像机的距离，前视平面到摄像机的距离小于后视平面到摄像机的距离。另外，还需要指定摄像机的视角，视角包括水平视角和垂直视角。水平视角是摄像机在水平方向上可以观察到三维景物的角度；垂直视角是摄像机在垂直方向上可以观察到三维景物的角度。有了前视平面、后视平面、水平视角、垂直视角，就可以在三维世界中形成一个几何空间，摄像机可以观察到的三维景物范围就是这个几何空间内的所有景物。

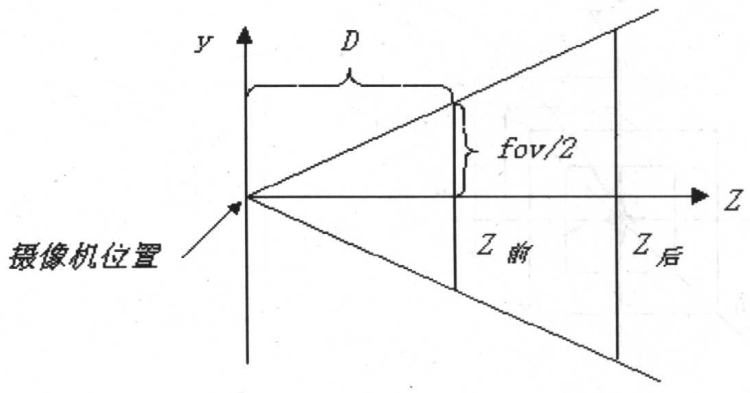


图 1-13　可视范围参数

图 1-13 中显示了前视平面、后视平面、水平视角、垂直视角在坐标系中的关系。坐标系中的 D 表示摄像机到可视空间的距离，Z 表示了前视平面和后视平面到摄像机的距离。Fovl 表示了摄像机的视角距离。

得到了三维世界中的可视空间范围，在 Direct3D 中需要进行透视转换。转换过程包括缩放和透视转换，把三维世界中可视空间范围物体在观察坐标系中的坐标转换成在新坐标系中的坐标。所有可视景物的坐标在新坐标系中的坐标值都在一个空间范围之内，这个空间范围的坐 x 标为-1 到 1，y 坐标为-1 到 1，z 坐标为 0 到 1，如图 1-14 所示。

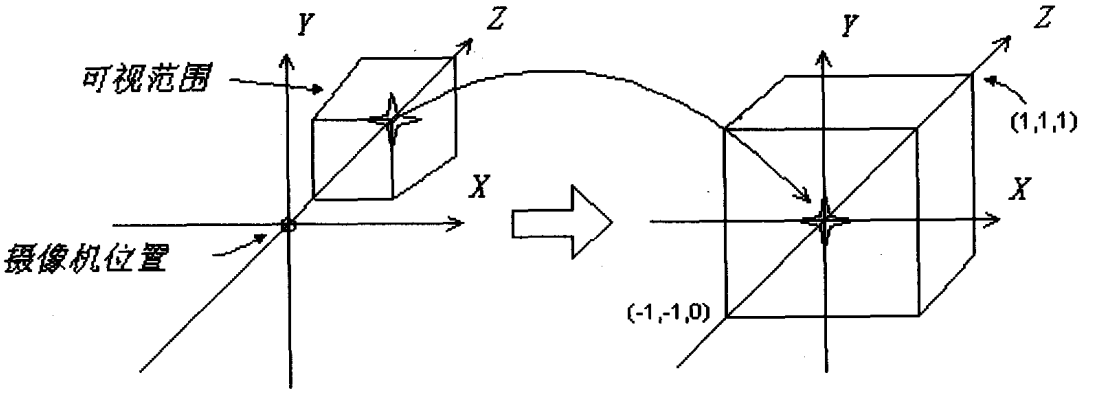


图 1-14　透视转换

为何需要把观察坐标系中景物的坐标进行转换，形成新的坐标呢？其实目的还是为了显示平面图形。在图 1-14 左面部分，显示了观察坐标系，观察坐标系的原点是摄像机。在图 1-14 的右面部分，显示了新的坐标系统。所有可以让摄像机观察到的三维物体都被转换到了一个小范围的、规则的几何空间内。下一步，仅仅需要做的就是选择在显示平面上需要显示的三维景物范围了。

在显示平面上显示哪些景物，只需要在新形成的坐标系中规则几何体内选择一个子空间就可以了，但是，子空间必须包含在透视转换形成的几何体内，范围绝对不能超出。超出了又如何？我也不知道，可能是“异度空间”吧。

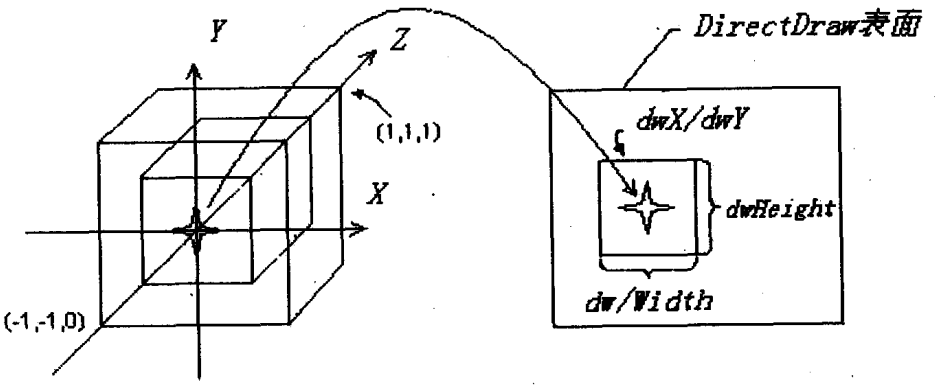


图 1-15　视口转换

图 1-15 中显示了在透视转换形成的几何空间内选择一个显示空间，显示在显示屏幕中的过程。图中举的例子是把景物显示在一个小矩形内，可以想像成我们在窗口模式下观察三维世界。当然，也可以显示到整个屏幕，在后面的编程学习中可以知道具体的过程。

1.1.7 小结

到此为止，我们了解了 Direct3D 中三维世界的表示方法、坐标系统，以及如何把三维世界中的景物显示在平面显示器上的基本概念和原理。或许大家觉得很简单，如果真的有这样的感觉，就对了。本书的原则就是尽量简单，让你掌握制作 3D 游戏的基本方法，简单的东西，绝不把它搞复杂了。不过不用担心，这样简单地学下去并不代表收获也少的。

1.2 建立编程环境

上一节学习了三维世界在计算机中的表示，以及其他必要的三维原理。接下来，要建立我们的游戏开发环境，进入真正的游戏开发，你准备好了吗？

本节中，需要掌握的内容如下：

- DirectX 的介绍
- DirectX 的安装
- 运行 3D 程序例子

1.2.1 DirectX 介绍

现在的电脑软件和电脑书籍实在是太多了，搞得大家面对太多的英文术语而不知其意。我们是要学习如何做三维电脑程序，而且，本书中教给你的是在 Windows 平台下的三维游戏开发。在 Windows 平台下开发程序，我们不得不又提到 Microsoft ，中文叫微软。在 Windows 下开发三维游戏，微软提供一套软件开发包，就是 DirectX 。为何叫做 DirectX？

以前在 DOS 下开发程序时，程序员想做什么都可以，只要是计算机硬件能够支持的功能，都可以通过直接操作硬件的方式来实现，什么限制都没有，那是程序员真正怀恋的时代。在 Windows 平台上，就没有这么自由了。一切都必须严格地通过操作系统提供给我们的接口来实现，我们必须记住太多的函数，通过微软留给我们的接口来实现需要的功能。虽然这样的方式让操作系统变得安全了，但是却让程序的效率变低了，同时程序的复杂度上升了，规模也变得越来越大，更重要的是束缚了程序员的创造力。在游戏领域，充满创造力的程序员当然是追求更高的效率和对硬件的完全的控制权。为了在游戏领域占有一席之地，微软也推出了 DirectX。Direct 是“直接”的意思，X 是代数中常用于表示未知数的字符，在这里可以理解成表示所有的一切。在游戏中对所有的一切硬件直接操作，这便是 DirectX 的含义。

微软的 DirectX 软件开发工具包（SDK）提供了一套应用程序编程接口（APIs），这个编程接口可以提供给你开发高质量、实时的应用程序，当然最主要的就是游戏所需要的各种资源。DirectX 技术为软件开发者提供硬件无关性。微软开发 DirectX，其最主要的目的之一是促进 Windows 操作系统上的游戏和多媒体应用程序的发展。在 DirectX 出现以前，主要的游戏开发平台是 MS-DOS，游戏开发者们为了使他们的程序能够适应各种各样的硬