第一部分　Direct3D 保留模式简介

DirectX 6 为你提供了编制下一代计算机游戏和多媒体程序的策略、技术和工具。本书是 DirectX 6 程序员参考手册。

微软公司的 DirectX 程序员参考手册为你提供了一组 API，可以用来设计高性能、实时的应用软件。 微软公司开发的 DirectX 用于 Windows 操作系统，其性能可以匹敌、甚至超过 MS-DOS 平台上的游戏。本程序员参考手册是用来为 Windows 下的游戏开发提供一个强大标准，而且文档良好的开发环境。

微软公司开发 DirectX 的主要目的之一就是促进 Windows 平台下的游戏开发。在 DirectX 开发出来之前，大多数游戏都是基于 MS-DOS 平台之上的。这些游戏的开发者必须为支持不同的卡而费神。而有了 DirectX 之后，游戏开发者既可以享受与设备无关的方便，又可以直接操纵硬件。开发 DirectX 的主要目的就是提供便于移植的 MS-DOS 特点，达到或者超过基于 MS-DOS 平台的程序的性能，并且消除个人电脑上面硬件革新带来的障碍。

此外，微软开发 DirectX 还能够为基于 Windows 的程序，针对当前及未来的硬件提供高性能、实时的直接操作。DirectX 在程序和硬件之间提供一个始终如一的接口，减少安装和调试硬件的复杂度，并且可以最大限度地发挥硬件的特点。 使用 DirectX 提供的接口，软件开发者可以充分发挥硬件的特点而不用关心其细节。

本部分主要包括以下内容:

- 第一章　保留模式
- 第二章　D3D 保留模式入门
- 第三章　Direct3D 保留模式体系结构
- 第四章　Direct3D 保留模式导览

第一部分　Direct3D 保留模式简介

DirectX 6 为你提供了编制下一代计算机游戏和多媒体程序的策略、技术和工具。本书是 DirectX 6 程序员参考手册。

微软公司的 DirectX 程序员参考手册为你提供了一组 API，可以用来设计高性能、实时的应用软件。微软公司开发的 DirectX 用于 Windows 操作系统，其性能可以匹敌、甚至超过 MS-DOS 平台上的游戏。本程序员参考手册是用来为 Windows 下的游戏开发提供一个强大标准，而且文档良好的开发环境。

微软公司开发 DirectX 的主要目的之一就是促进 Windows 平台下的游戏开发。在 DirectX 开发出来之前，大多数游戏都是基于 MS-DOS 平台之上的，这些游戏的开发者必须为支持不同的卡而费神，而有了 DirectX 之后，游戏开发者就可以享受与设备无关的方便，又可以直接编程硬件，开发 DirectX 的主要目的就是提供了获得的 MS-DOS 特点，达到或者超过基于 MS-DOS 平台的程序的性能；并且流畅个人电脑上画面作革新带来的降低。

此外，微软开发 DirectX 还能够为基于 Windows 的程序、针对当前及未来的硬件提供高性能、实时的直接操作。DirectX 在程序和硬件之间提供一个简洁和一致的接口，减少安装和调试硬件的复杂度，并且可以最大限度地发挥硬件的特点。使用 DirectX 提供的接口，软件开发者可以充分发挥硬件的特点而不用关心其细节。

本部分主要包括以下内容：

- 第一章　保留模式
- 第二章　D3D 保留模式入门
- 第三章　Direct3D 保留模式体系结构
- 第四章　Direct3D 保留模式导览

第一章 保留模式

这一章我们描述的是 Microsoft 为个人计算机上的实时三维图像提供的一种解决方案 Direct3D 保留模式。如果你需要在你的 PC 机上创造一个三维环境并且要实时的操作和处理它的话，那么你应该使用 Microsoft 公司的 Direct3D 保留模式的应用程序接口（API）来编程。

Direct3D 是和 DirectDraw 紧紧集成在一起的，一个 DirectDraw 对象中封装了 DirectDraw 和 Direct3D 两种状态。我们的应用程序可以用 IDirectDraw::QueryInterface 调用的方法来得到一个 DirectDraw 对象的 IDirect3D 接口。

如果你以前使用过三维图像编程的话，那么在 Direct3D 保留模式下的许多概念对你来说将是很熟悉的。然而，如果你是一个三维图像编程的初学者的话，你应该十分注意“Direct3D 保留模式结构”，并且你应该认真阅读“入门（Getting Started）”。不管你是一个新的三维编程者或是初学者你都应该认真阅读 SDK 中包含的例子程序，这些例子分析了在一个实时应用程序中怎样使保留模式正确工作。

这一章是对三维编程的简介，它描述了“三维图形解决方案”和在三维空间中怎样来操作和处理点所必需的一些技术背景，这不是对“Direct3D 保留模式”编程的介绍。如要看“Direct3D 保留模式编程”，那么请参见“Direct3D 保留模式导读”。

第二章 D3D 保留模式入门

下面这一节我们来描述一下在编写一个三维图像程序之前你应该理解的一些技术上的概念。你会学习到我们对坐标系及其变换的一个简略的讨论，这不是对其结构广泛详细的讨论，比如像建立一个模型，灯光和参数的设置。若要看关心这些更详细的信息，那么请参见“Direct3D 保留技术结构”。

如果你是一个经验丰富的三维图像编程人员，那么你只需要对下面章节中关于 Direct3D 保留模式所特有的信息加以留意即可。

2.1 3D 坐标系统

在三维图象中有二种不同的直角坐标系：左手直角坐标系和右手直角坐标系。在这两种坐标系中，x 的正轴都指向右手，y 轴的正方向为正上方。你会记得 z 轴的正向是由以下法则得到的：用大拇指外的四个手指指向 x 轴正向，再环绕到 y 轴，那么你的大拇指指向的方向的方向便是 z 轴的正方向。在左手直角坐标系中，z 轴的正方向为远离我们，向右手坐标系则刚好相反为指向我们。

这一节我们讨论 Direct3D 直角坐标系和你的应用程序会用到的坐标类型：

- Direct3D 坐标系统
- u 和 v 坐标

2.1.1 Direct3D 直角坐标系统

Direct3D 使用左手系直角坐标系统，这意味着 z 轴正方向为远离观察者，如图 2-1 分析所示。

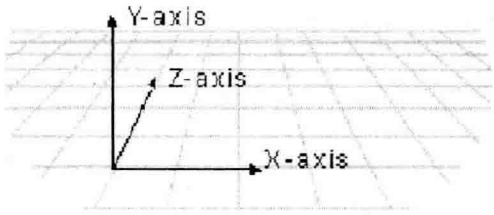

图 2-1

在左手直角坐标系中，物体的旋转是沿指向观察者的顺时针方向进行的。

如果你需要在右手直角坐标系中工作。比如，当你在输入一个基于右手的应用程序时，你可以按以下方法进行简单的变换，便可以把变换后的数据传送给 Direct3D 的直角坐标系

统。

- 交换 3 个坐标值；使得系统沿指向观察者的顺时针方向运行，也就是如果输入右手坐标系为 V0，V1，V2，那么我们以 V0，V2，V1 顺序把 3 个分量传递给 Direct3D。
- 在 z 轴上进行-1 的向量尺度变换，为了达到 z 轴上-1 的向量尺度变换，我们把“D3DMATRIX 结构”中的_13，_23、_33 和_43 单元的符号反号即可。

2.1.2 U 和 V 坐标

Direct3D 也使用纹理坐标，这些坐标（u 和 v）用于我们把纹理映射到一个对象上去。v 向量描述用于描绘纹理的方向，它是沿着 z 轴的方向的，u 向量（向上的向量）一般是沿着 y 轴的正向，其源点起源于 Direct3D 坐标的[0，0，0]点，要更详细的了解 u 和 v 坐标系，请参见“IDirect3DRMWrap 接口”。

2.2 三维变换

在三维图象的应用程序中，我们可以用几何变换完成以下功能：

- 表达出一个物体相对于另一个物体的位置
- 旋转、截取（剪切）和缩放对象
- 变换观察的位置，方向和透视（物体的相对坐标）

你可以用一个 4×4 矩阵把一个点变换为另一个点，在下面的例子中，我们用一个 4×4 矩阵把点（x，y，z）变换为一个新的点（x′，y′，z′）：

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

你对（x，y，z）和矩阵 M 进行如下运算你可得到点（x′，y′，z′）：

$$X' = (M_{11}XX) + (M_{21}XY) + (M_{31}XZ) + (M_{41}X1)$$
$$Y' = (M12XX) + (M22XY) + (M32XZ) + (M42X1)$$
$$Z' = (M13XX) + (M23XY) + (M33XZ) + (M43X1)$$

在 3D 图象中，最通常的变换有移动（translate），旋转（Rotate）和缩放（Scaling 也叫比例化）即放大或缩小，这些都是通过矩阵变换来实现的，你可以把这些功能的矩阵复合起来，构成一个新的矩阵，使我们能够只进行一次矩阵运算便产生几种功能的复合。比如，

你可以建立一个矩阵，通过这个矩阵变换对一系列的点同时进行平移和旋转变换。

矩阵以行的次序来确定，例如，下面矩阵可以由一个向量来表示：

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

这个矩阵的向量如下：

```
D3DMARTZX Scale=[
        D3DVAL（S）,    0,          0,          0,
        0,          D3DVAl（s）,  D3DVAl（t）,  0,
        0,          0,          D3DVAl(s),    D3DVAl(v),
        0,          0 ,         0,            D3DVAl（1）,
],
```

这一节描述 Direct3D 中，你的应用程序可以使用的 3D 变换：

- 平移（translate）
- 旋转（rotate）
- 缩放（scaling）

本书的其它部分也讨论了三维变换，在提供给保留模式的视图，变换的那一节中，我们可以看到对变换的整体的一般性的讨论。对帧变换的讨论，我们可以在“变换（Transformations）”中去学习。尽管，这些小节讨论的都是保留模式的应用程序接口（Retainted Mode，API），但是其变换的体系结构和数学运算同时适用于保留模式和立即模式。

2.2.1 平 移

以下的变换把点（x、y、z）移到一个新的点（x′，y′，z′）：

$$[x' \quad y' \quad z' \quad 1]=[x \quad y \quad z \quad 1]\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_X & T_Y & T_Z & 1 \end{bmatrix}$$

2.2.2 旋 转

本节的讨论的变换都是基于左手直角坐标系的，所以有可能和你在某些右地方看到的变换矩阵有一些不同。

下面的变换使点（x、y、z）沿 x 轴旋转，产生一个新的点（x′，y′，z′）：

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

下面的变换使点（x，y，z）沿 y 轴旋转，生成（x′，y′，z′）

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

下面的变换重点沿 z 轴旋转。

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

注意：在这些矩阵中，希腊字母θ代表旋转的角度，它的单位为“弧度”，当从旋转轴正向看的时候，旋转角度θ是以顺时针来计的。

2.2.3 缩 放

下列尺度变换把一个点（x、y、z）在 x、y、z 三个轴分别以任意比例缩放以形成一个新的点（x′，y′，z′）：

$$[x' \quad y' \quad z' \quad 1]=[x \quad y \quad z \quad 1]\cdot\begin{bmatrix}S_X & 0 & 0 & 0\\ 0 & S_Y & 0 & 0\\ 0 & 0 & S_Z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

2.3 多 边 形

Direct3D 中的一个三维物体对象是由许多网格（mesh）构成的，一个网格便是一套面的集合，而每个面可以由一个简单的多边形描述。多边形的基本类型是三角形尽管在保留模式应用程序中，可以指定多于 3 个顶点的多边形。但系统在视同显示之前要把这些多边形转换为多个三角形，再以三角形作为基本处理对象，而立即模式应用程序则必须为三角

形。

这一节描述你的应用程序怎样使用 Direct3D 的多边形：

- 几何要求（Geometry Requirement）
- 面和顶点的规范化（Face and Vertex Normals）
- 阴影模式（Shade Modes）
- 三角形内插（Triangle interpolants）

2.3.1 几何要求

三角形是最理想的多边形是因为它始终是凸多边形，并且它始终在一个平面上。这是视图显示一个多边形要求的两个条件，一个多边形是凸的是指任意两个顶点的连线都在多边形内，如图 2-1 所示。

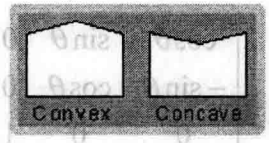

图 2-1

三角形的三个顶点确定一个平面，当我们再增加一个顶点时，我们能很容易地产生一个非平面的多边形，如图 2-2 所示。

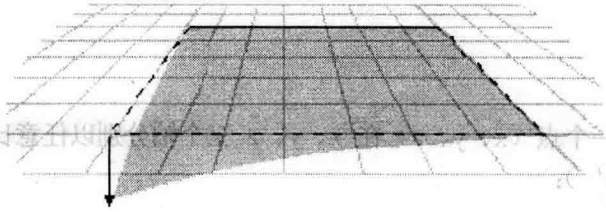

图 2-2

2.3.2 面和顶点的规范化

每一个网格有一个直的面“规范化向量（normal vector）”。规范化向量的方向由顶点定义的次序和直角坐标系（左手或右手为基础）来确定。如果规范化向量的再是朝向观察者的，那么那面的方向便是它的前面。在 Direct3D 中只有前面的那一定是可见的，在前面的面中，顶点是以顺时针方向定义的，如图 2-3 所示。

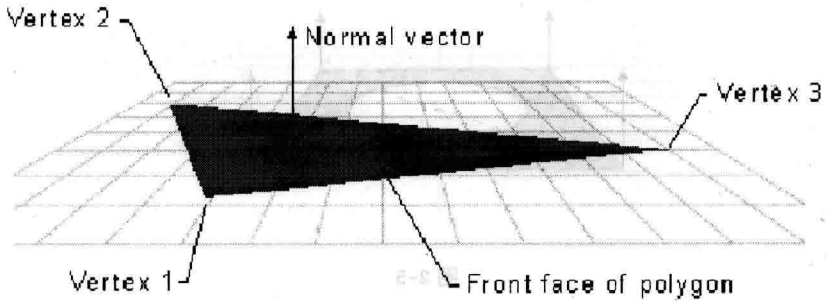

图 2-3

Direct3D 应用程序不用指定规范面，系统需要时会自动的计算它们。系统在“平面阴影模式”中会用到面规范化，而在“phong 和 Gouraud 阴影模式”中，以及控制线和角的效果时，系统使用顶点规范化，如图 2-4 所示。

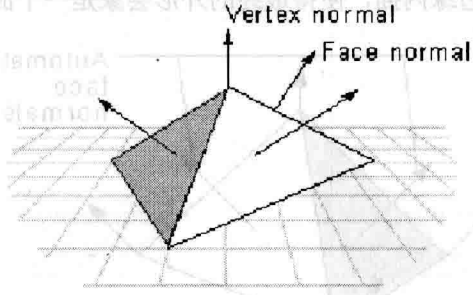

图 2-4

2.3.3 阴影模式

在平面阴影模式中，系统把一个顶点的颜色复制到所有画上，而在 Gouraud 阴影模式和 phong 阴影模式中，顶点规范化用于使一个多边形物体对原有一个圆之间的外观。在 Gouard 阴影集式中，顶点的颜色和饱和度根据它们等比的距离来内插，而在 phong 阴影模式中，系统对面上的每一个像素都进行相应阴影值进行计算。

注意：phong 阴影模式现在还不能支持。

许多应用程序中使用 Gouraud 阴影模式；因为它使物体对象看上去圆润，并且其计算效率高。对于 Gouraud 阴影模式会遗失一些细节面 phong 阴影模式则不会，然而 Gouraud 和 phong 阴影模式会产生非常不同的效果。在图 2-5 中我们会看到在一个面上包含了一个亮点的例子。

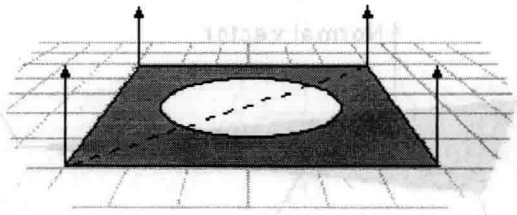

图 2-5

在这种情况下，phong 阴影模式会对每一个像素进行计算，它会显示出一个亮点来，而在 Gouraud 阴影模式中，由于使用的是顶点的内插，所以会使亮点整体在面上消失，这时这个面在透式图显示时会使亮点不存在。

在平面阴影模式中，下图 2-6 的“金字塔”（棱柱体）在两个面相邻的棱边处，会有一个尖的边缘，系统会自动产生面规范化，使得图形圆润。然而，在 Gouraud 阴影模式和 phong 阴影模式中，阴影值通过边缘内插，使得最终的外形会象是一个曲面。

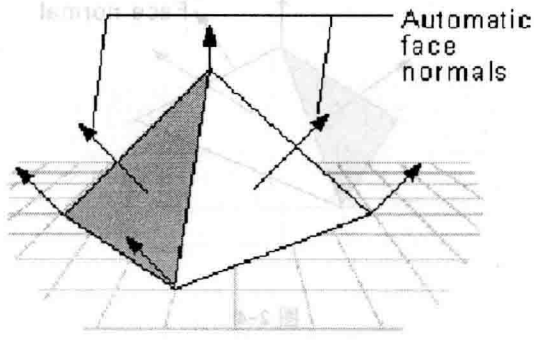

图 2-6

如果你想使用 Gouraud 阴影模式或 phong 阴影模式来显示曲面，并且想使某些物体对象有尖的边缘，那么你的应用程序需要在面效的边缘（要求有尖的边缘的边）复制顶点规范化。如图 2-7 所示。

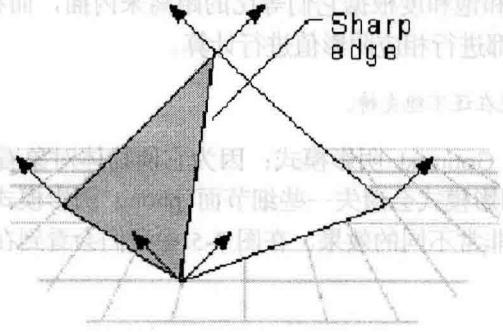

图 2-7

另外，如果想使一个物体同时具有曲的和平的平面，那么 Gouraud 阴影模式比平面阴影模式有实际意义。一个在平面阴影模式下的面是一个单色画，但 Gouraud 阴影模式允许光线正确透射到面上，如果是接近源的点这种效果会特别明显，在许多 Direct3D 应用程序中，Gouraud 阴影模式是很好的一种阴影模式。

2.3.4 三角形的内插

系统在透视图中显示一个面时，它会通过对三角形进行，三角形顶点间内插特征来处理一个三角形，所谓内插就是指一种渐变的特性，试想在两个点之间进行颜色的内插一个点为红，另一个点为点蓝色，那么，在此两个点间线上的颜色为红到蓝的渐变。以下为三角形的内插属性：

- 颜色（Color）
- 镜像（Specular）
- 雾（Fog）
- 阿尔法（Alpha）

所有的三角内插属性都由当前阴影模式改变。

平面阴影模式（Flat） 没有内插进行，它是用第一个顶点的颜色来填充整个三角形的面。

Gouraud 阴影模式 在所有 3 个顶点之间使用线性内插。

Phong 阴影模式 每个面上的像素的参数重新计算，并使用当前灯光，但 phong 阴影模式当前还不被支持。

根据不同的颜色模式我们应区别对待颜色内插和镜像内插。当在 RGB 彩色模式中（D3DCOLOR_RGB）。系统使用红色、绿色和蓝色三个基本色分别内插，然后得到其红色、绿色和蓝色的基本包以构成其颜色，而在单色模式（D3DCOLOR_MONO）下时，系统仅在顶点颜色中的蓝色分量进行内插。

例如，如果顶点 1 的颜色中红色分量为 0、8 的顶点之为 0、4，当我们在 Gouraud 阴影模式和 RGB 彩色模式下时，系统会使用内插来为其两顶点间中点处分配一个红色分量为 0、6 的值，以完成其颜色内插。

颜色的 alpha 分量被分开来单独内插是因为这样可以使设备驱动程序，可以有两种透明的实现方式——使用表面混合（texture blending）或使用打点（stippling）。

一个应用程序可以使用 dwShadeCaps 中的 D3DPRIMCAPS 结构来确定当前，设备驱动程序所支持的是哪种类型的 alpha 内插。

2.4 三角形的条块和三角形扇面

你可以使用三角形条块和三扇形展开来指定一个面，这样你可以不用提供每个三角形的三角顶点，而只需提供整个面的顶点便可以。例如，图 2-8 指示出了只需七个顶点便确定

了以下的三角形条块。

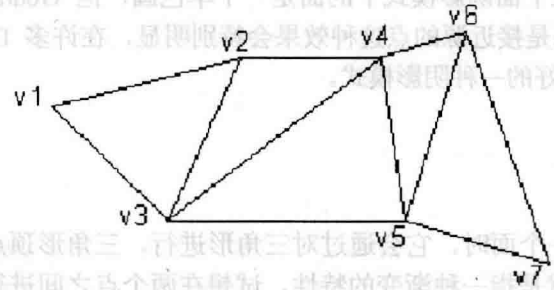

图 2-8

系统使用顶点 V1，V2 和 V3 来画第一个三角形，用 V2，V4 和 V3 画第二个三角形，用 V3，V4 和 V5 来画第三个三角形，用 V4，V6 和 V5 来画第四个三角形。以此类推，注意第二个和第四个三角形的顶点是无序的，这要求确认每个三角形是以顺时针画出的。

一个三角形的扇面与三角形条块相似，只是其所有三角形共有一个顶点。

如图 2-9 所示，系统用顶点 V1，V2 和 V3 来画第一个三角形，V3，V4 和 V1 来画第二个三角形，V1，V4 和 V5 来画第三个三角形，以此类推。

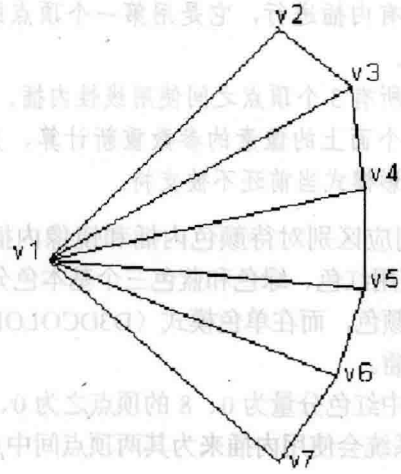

图 2-9

你可以使用 Wflag5 成员中的 D3DTRIANGLE 结构来指定建立三角形条块和三角形扇面的标志。

2.5 向量、顶点和四变量组

在 Direct3D 中，顶点描述位置和方向，每一个顶点原始地由以下来描述：一个向量描述其位置，一个标准向量（规范化向量）指出其方向，网面坐标和颜色（在保留模式中，

D3DRWVERTEX 结构中包含有这些值）。

四变量组（Quaternions）在[x、y、z]值的基础上增加了一个值来定义一个向量，四变量组是作为矩阵方法（用于三维旋转的）的替换方法。一个四变量组代表一个三维空间的轴和沿这个轴的一次旋转。例如，一个四变量组可以代表（1、1、2）这个轴和旋转 1 弧度这一组合。四变量组带有非常有用的信息，但它们真正的有用之处在于你对它们的两种操作——合成和内插。（Composition and interpolation）。

对四变量组进行合成运算就好象结合它们（四变量）一样，两个四变量组的合成式如下：

$$Q = q_1 \bullet q_2$$

两个四变量组的合成运算在几何中的运算是指“先将几何图形沿轴 2 旋转的 r_2 弧度。再将它沿轴 1 旋转 r_1 弧度”。这时，Q 代表沿一个单轴旋转一个角度，这等效于选沿 q2 旋转一次再沿 q1 旋转一次。

使用四变量组的内插，一个应用程序能够计算从一条轴指向另一条轴的平滑和可信的路径。因此，在 q1 和 q2 之间内插提供了一种从一个方向向另一个方向运动的简易方法。

当你同时使用合成和内插时，它给你提供了一种处理几何图形看似复杂的简便方法。例如，想象你有一个几何图形想要沿某一给定方向旋转。你知道你想把它沿轴 2 旋转 r_2 弧度。但是你不知道最终的四变量组，通过合成，你可以把这两次旋转（对几何图形的）组合成一个和两次旋转效果一样的四变量组。然后你可以在旋转开始到组成的单一合成四变量组之间使用内插来获得两次旋转的平滑过渡。

Direct3D 保留模式包含了一些函数来与四变量组一起工作。例如，函数 D3DRMQuaternionFromRotation 为向量（确定旋转的轴）定义了一个旋转的值，并返回由 D3DRMQUATERNION 结构定义的四变量组中的一个值。另外，D3DRMQuaternionMutiply 函数合成一个四变量组，而 D3DRMQuaternionSlerp 提供两个四变量组的线性内插。

保留模式应用程序可以使用以下函数来简化与向量和四变量组协同工作的任务。

- D3DRMQuaternionFromRotation
- D3DRMQuaternionMultiply
- D3DRMQuaternionSlerp
- D3DRMVectorAdd
- D3DRMVectorCrossProduct
- D3DRMVectorDotProduct
- D3DRMVectorModulus
- D3DRMVectorNormalize
- D3DRMVectorRandom
- D3DRMVectorReflect
- D3DRMVectorRotate
- D3DRMVectorScale
- D3DRMVectorSubtract

浮点数精度

象其它的 DirectX 结构一样，Direct3D 使用 53 位的浮点数精度，如果你的应用程序需要改变这个精度，那么当计算完成时，你必须把它转换回 53 位的浮点数精度，否则，系统中那些选用缺省值的部分将会停止工作。

第三章 Direct3D 保留模式体系结构

所有对 Direct3D 保留模式的访问都是通过一小套对象来完成的，下面表列出了这些对象和关于他们的简略描述：

对 象	描 述
Direct3DRMAnimation	这个对象定义了一个变换会怎样改变，通常是以一个 Direct3DRMFram 或 Direct3DRMFrame2 对象为参考，你可以使用它来改变其位置，方向和缩放 Direct3DRMVisual，Direct3DRMLight 和 Direct3DRM Viewport 对象。
Direct3DRMAnimationSet	这个对象允许多个 Direct3DRMAnimation 对象集合在一起。
Direct3DRMDevice	这个对象代表视图显示的目标设备。
Direct3DRMDevice2	这个对象代表的和 Direct3DRMDevice 一样，但其对象还具有透明性的增强性控制。
Direct3DRMFace	这个对象代表网格上的一个多边形。
Direct3DRMFrame	这个对象把一个对象放于一个画面上并定义其可见对象的位置和方向。
Direct3DRMFrame2	它是 Direct3DRMFrame 对象的扩展，它可以访问帧轴，捆梆盒子和实物，也支持光线选择。
Direct3DRMInterpolator	这个对象存贮动作，并对起始之间的动态自己计算其相应的值。
Direct3DRMLight	这个对象定义五种光线类型之一，其光线用于照亮屏幕上的可见对象。
Direct3DRMMaterial	这个对象定义一个表面怎样反射光。
Direct3DRMMesh	这个对象由一套多边形的面组成，你可以使用这个对象来对一组面和顶点进行操纵。
Direct3DRMMeshBuilder	这个对象允许你对网格中的单个顶点和面进行处理。
Direct3DRMMeshBuilder2	这个对象同 Direct3DRMMeshBuilder 一样允许你对网格中单个顶点和面进行处理，但比 Direct3DRM MeshBuilder 有更多功能。
Direct3DRMObject	这个对象是所有其它 Direct3D 保留模式的对象必须用到的一个基本类，它具有所有对象所共有的特征。
Direct3DRMPickedArray	这个对象确定一个可视对象在二维空间中的对应物。
Direct3DRMPicked2Array	这个对象确定一个可视对象对一个给定光线横插的对应物。

Direct3DRMProgressiveMesh	这个对象由粗网格和描述网格精度的增加的记录组成，这允许当在远程下载图象时，先出现一个粗糙的图象，然后逐渐增加图像的精确度。
Direct3DRMShadow	这个对象定义了一个阴影。
Direct3DRMTexture	这个对象是一个彩色象 的矩形向量。
Direct3DRMTexture2	这个对象除具有 Direct3DRMTexture 功能外，它还具有如下功能：对象可以从现在执行文件以外的文件调入，矩形向量尔来可以由内存中的图像创造，并且你可以产生一个 MIP map 映象。
Direct3DRMUserVisual	这个对象由应用程序定义用于提供系统中不可见的功能性。
Direct3DRMViewport	这个对象定义怎样在二维空间中显示三维的视图。
Direct3DRMVisual	这个对象是可以在一个场景中显示的某些东西，虚拟的对象不必要是可视的，例如，一个帧可以增加为一个虚拟对象，而一个帧本身不能可视（只有其内容可见）。
Direct3DRMWrap	这个物体计算一个网格或面上的纹理结构。

一些对象可以组成向量，叫做“向量对象”。向量对象使对整个向量组的操作得以简化。COM 接口允许你对向量对象进行操作，它包含 GetElement 和 GetSize 方法，这些方法分别得到一个指向向量对象中的元素（对象）的指针和向量的大小。要对向量对象有更多的了解。请见“IDirect3DRM 向量接口”。

3.1 对象和接口

当一个对象支持某一接口时，我们可以通过调用 IObjectName::QueryInterface 来返回一合法的接口指针。因此，你可以调用 IDirect3DRMDevice::QueryInterface 来获取 IDirect3DRMWinDevice 接口，但不是 IDirect3DRMVisual 接口。

下面列出了对象所支持的接口。

Direct3DRMAnimation	Direct3DRMAnimation
Direct3DRMAnimationSet	Direct3DRMAnimationSet
Direct3DRMDevice	Direct3DRMDevice，Direct3DRMWinDevice
Direct3DRMFace	Direct3DRMFace
Direct3DRMFrame	Direct3DRMFrame，Direct3DRMVisual
Direct3DRMFrame2	Direct3DRMFrame2，Direct3DRMVisual
Direct3DRMinterpolator	Direct3DRMInterpolator
Direct3DRMLight	Direct3DRMLight
Direct3DRMMaterial	Direct3DRMMaterial
Direct3DRMMesh	Direct3DRMMesh，Direct3DRMVisual

Direct3DRMProgressiveMesh	Direct3DRMProgressiveMesh，Direct3DRMVisual
Direct3DRMBuilder	Direct3DRMBuilder ，Direct3DRMVisual
Direct3DRMBuilder2	Direct3DRMBuilder2，Direct3DRMVisual
Direct3DRMShadow	Direct3DRMShadow ，Direct3DRMVisual
Direct3DRMTexture	Direct3DRMTexture ，Direct3DRMTexture2 ，Direct3DRMVisual
Direct3DRMUserVisual	Direct3DRMUserVisual ，Direct3DRMVisual
Direct3DRMViewport	Direct3DRMViewport
Direct3DRMWrap	Direct3DRMWrap

下面的例子说明了对一个 Direct3DRMDevice 对象怎样创建两个接口。

用 IDirect3DRM::CreateObject 可以创建一个来被初始化的 Direct3DRMDevice 对象，而使用 IDirect3DRMDevice::InitFromClipper 可以初始化对象，对 IDirect3DRMDevice::QueryInterface 的调用可以为 Direct3DRMDevice 对象创建第二个接口，应用程序将使用 IDirect3DRMWinDevice 接口来处理 WM_PAINT 和 WM_ACTIVATE 消息。

```
d3drmapi->CreateObject(CLSID_CDirect3DRMDevice，NULL，
    IID_IDirect3DRMDevice，(LPVOID FAR*)&dev1);
dev1->InitFromClipper(lpDDClipper，IID_IDirect3DRMDevice，
    r.right，r.bottom);
dev1->QueryInterface(IID_IDirect3DRMWinDevice，(LPVOID*) &dev2);
```

当我们要确定两个接口是否指向一个相同的对象时，每个接口分别通过调用 QueryInterface 并且比较其返回的指针，如果其指针相同，那么这两个接口指向同一个对象。

所有的 Direct3D 保留模式的对象都支持 IDirect3DRMObject 和 IUnknown 接口以及上面列出的接口，向量对象不是从 IDirect3DRMObject 中导出的，向量物体没有类识别等，因为它们没有必要，应用程序并不能通过调用 IDirect3DRM::CreateObject 的方法来产生一个向量对象。然而，向量对象可以通过下表的方法来产生一个向量对象。

IDirect3DRMDeviceArray	IDirect3DRM::GetDevices
IDirect3DRMFaceArray	IDirect3DRMMeshBuilder::GetFaces IDirect3DRMMeshBuilder2::GetFaces
IDirect3DRMFrameArray	IDirect3DRMPickedArray::GetPick， IDirect3DRMPicked2Array::GetPick， IDirect3DRMFrame::GetChildren
IDirect3DRMLightArray	IDirect3DRMFrame::GetLights
IDirect3DRMObjectArray	IDirect3DRMInterpolator::GetAttachedObjects
IDirect3DRMPickedArray	IDirect3DRMViewport::Pick