文章编号: 1672－6952(2008)04－0086－05

Direct3D 场景中 3D 模型的处理方法

杜园园, 侯彤璞*, 郭艳霞
(辽宁石油化工大学计算机与通信工程学院, 辽宁抚顺 113001)

摘　要: 介绍了在三维虚拟场景中 Direct3D 调用 3DSMAX 制作完成的模型(实体)的方法, 即如何通过 D3D 来运用 3D 作品。提出了 3D 实体的概念, 研究了基于坐标综合变换的模型加载过程; 重点介绍了向量的坐标变换以及.x 文件加载过程中确定 3D 模型的空间位置的方法; 最后用程序再现了一个真实的场景。由于在 Direcrt3D 中对 3D 模型的处理介绍得比较少, 因此, 此研究有一定的应用价值。

关键词: 3DSMAX; 3D 模型加载; 坐标变换; Direct3D
中图分类号: TP183　　文献标识码: A

Disposal Method of 3D Models Used in Direct3D Scene

DU Yuan－yuan, HOU Tong－pu*, GUO Yan－xia
(*School of Computer and Communication Engineering, Liaoning University of Petroleum & Chemical Technology, Fushun Liaoning* 113001, *P. R. China*)

Received 4 *September* 2008; *revised* 25 *October* 2008; *accepted* 30 *October* 2008

Abstract: The method of Direct3D call the model (entities), which is created by 3DSMAX in a virtual three－dimensional (3D) scene was introduced, that is, how to use 3D works by D3D. The concept of 3D entities was proposed and the loading process of model which is based on an comprehensive transformation of coordinate was studied. The transformation of vector－coordinate and method to determine the 3Dmodel's space location during process of loading 3D model were specially introduced. Finally, procedures were used to reproduce a real－life scene. The 3D model's handling is seldom introduced in Direcrt3D, so this study has some value of application.

Key words: 3DSMAX; 3D module loading; Coordinate transformation; Direct3D

*Corresponding author. Tel.: ＋86－413－6865132; fax: ＋86－413－6860766; e－mail: htp1917@163.com

OpenGL 即开放性图形开发库, OpenGL 是国际上公认的 3D 图形工业标准, 它不仅加速了 3D 应用程序的开发, 而且使应用程序可移植性更好。但是在开发交互式 3D 应用程序方面, OpenGL 存在明显的不足: OpenGL 与窗口系统无关, 不提供任何交互手段, 必须有程序员自己编写所有的交互功能; OpenGL 编程接口(API)是低级的 C 函数, 不提供可复用(Reuse)的对象库或者应用程序框架, 开发效率不高; 缺乏对文字绘制的很好支持[1]。

而 DirectX 确克服了这些问题, 它是一种图形应用程序接口, 简单地说它是一个辅助软件, 一个提高系统性能的加速软件, 由微软开发的[2]。

Direct 3D是微软公司DirectX SDK 集成开发包中的重要部分, 是基于微软的通用对象模式COM (Component Object Model)的 3D 图形 API[3], 通过这些 COM 对象, 开发者可以在无视设备差异的情况下访问底层的硬件, 可以在隐藏特定硬件细节的情况下让程序员去访问硬件设备, 微软称之为"硬件无关性"。因为 Direct3D 既具有硬件设备无关性又支持硬件加速, 因此成为三维图形程序设计的首选工具[4]。

开发任何三维程序, 都要涉及到非常复杂的三维物体数据模型, 如果直接用 Cbuildvertex 类来构造这些模型, 显然是不合实际的, 这些复杂的模型通常都是用专业的三维建模软件来制作, 例如 3DSMax 和 Maya 等软件[5]。这些软件制作了模型之后再用一些工具或插件将制作好的模型文件转换成在 Direct3D 可以直接使用的.x 文件格式。在.x 文件中存储了三维模型的顶点坐标 颜色 法向量和

收稿日期: 2008－09－02
作者简介: 杜园园(1982－), 女, 河北保定市, 在读硕士。
*通讯联系人

纹理坐标等信息[6]。每一个.x 文件就是一个实体。在三维建模过程中我们不可能把整个虚拟场景都建成一个.x 文件，这样对以后的场景管理会带来麻烦，因此我们用 3DSMax 把场景中的模型根据需要建模，然后再重新组织成一个整体的虚拟场景。独立的模型会给场景管理带来极大的方便，而在场景的布置中模型的位置确定是必不可少的，本文重点分析了模型位置的确定在编程中是如何实现的，而在 Direct3D 中没有直接提供一个整体的处理方法。

1　三维实体建模

1.1　实体

由于在三位虚拟场景中涉及到的事物比较多，为了给以后的管理带来方便，将任何事物都统一成能够一起管理的东西，以达到简化和方便程序设计的目的。在以后的应用中所有的模型简称为实体。

1.2　实体的创建

首先定制环境，在 Windows XP 下安装 DirectX 9.0SDK 开发工具，并确认在 D3D 的插件工具里包含一个 eonv3ds 转换器。D3D 中一个 3D 物体——包含了顶点信息、材质、贴图、动画帧的 3D 成品被叫做 mesh 物体。conv3ds 就是用来把 3D 作品转换为标准 mesh 物体文件（*.x）的工具。首先打开 3DSMax（或 Maya 等其他 3D 软件，确保能导出 D3D 能用的文件），根据需要制作我们想要的模型。通过 conv3ds 命令由 3ds 文件转换而来，因此必须首先得到 3ds 文件，然后通过 3DSMax 中 export 命令导出。在菜单 file 中用 export 或 export select（只导出所选物体）将其导出为 *.3ds 文件。然后用命令行方式运行 conv3ds 把它转为 *.x。这样就有了一个可用的3D作品。默认的 *.x 都是二进制的，这样写就生成文本式的 *.x：conv3ds －x *.x，用写字板形式或其他形式打开，你会直接看到 mesh 物体的构成。.X 文件的头信息：在.x 文件的开头一般有一个简单的文件信息：xof 表明这是一个.x 文件；0302 表示其版本号，txt 表示其存储的文件格式是文本格式. 其存储格式还可以是二进制方式，用 binary 来表示；0302 是 32 位浮点数。如果是 64 位就可以用 0604 表示了；模板定义：定义的每个模板都以关键字 template 开头。然后是模板名。每个定义的模板都有一个 GUID 以识别该模板的唯一身份，模板是可以嵌套的。模板一般分三种类型：封闭式模板，开放式模板，限制式模板；数据对象：定义好模板之后再实例化，以存放实际数据。先是模板名 然后是实例名 实例名不是必须的 可以省略，然后就是存放实际数据[7]。文件的加载就是对数据的识别从而显示出真实的场景。

2　坐标转换

将 3ds 文件转换成 *.x 文件并不能直接实现真实场景的再现，必须把实体放到场景中属于自己的位置。需要解决的就是实体的位置、大小和方向。要想确定模型的空间位置，只需确定模型的中心坐标。在世界坐标系中，每一个坐标都可以看成一个标准向量。由于模型是一个整体，中心坐标的变化会引起整体变化。

当用 Direct3D 编程时，使用 4×4 矩阵来进行矩阵变换。使用 4×4 矩阵是因为这样的大小能表现需要的所有变换，比如平移、投影、反射等[8]。若要使某一向量 $p=(x\ y\ z\ 1)$ 平移，旋转，放缩可以通过下面的变化实现：通过与 $T(p)$ 矩阵相乘把向量 $(x\ y\ z\ 1)$ 沿 x 轴移动 p_x 个单位，沿 y 轴移动 p_y 个单位，沿 z 轴移动 p_z 个单位；通过与 $S(q)$ 矩阵相乘把向量沿 x 轴缩放 q_x 个单位，沿 y 轴缩放 q_y 个单位，沿 z 轴缩放 q_z 个单位。

$$T(p)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\p_x&p_y&p_z&1\end{bmatrix}$$

$$S(q)=\begin{bmatrix}q_x&0&0&0\\0&q_y&0&0\\0&0&q_z&0\\0&0&0&1\end{bmatrix}$$

通过与下面的矩阵相乘把一个向量围绕 x，y 和 z 轴旋转 θ 弧度。注意：当俯视绕轴原点时，角度是指顺时针方向的角度。

$$X(\theta)=\begin{bmatrix}1&0&0&0\\0&\cos\theta&\sin\theta&0\\0&-\sin\theta&\cos\theta&0\\0&0&0&1\end{bmatrix}$$

$$Y(\theta)=\begin{bmatrix}\cos\theta&0&-\sin\theta&0\\0&1&0&0\\\sin\theta&0&\cos\theta&0\\0&0&0&1\end{bmatrix}$$

$$Z(\theta)=\begin{bmatrix}\cos\theta&\sin\theta&0&0\\-\sin\theta&\cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$

综合变换：常常要对一个向量进行一系列的变换　比如 可能先缩放一个向量 然后旋转它 最后

再把它平移到指定的位置。例如：先把向量 $p=[5\ 0\ 0\ 1]$ 在所有轴上缩小为原来的 1/5，然后绕着 y 轴旋转 $\pi/4$，最后把它在 x 轴上移动 1 个单位，在 y 轴上移动 2 个单位，在 z 轴上移动 3 个单位。注意：必须完成缩放，绕轴旋转，最后完成移动。设缩放、旋转、移动的变换矩阵分别是 S、R_y、T 如下：

$$S(1/5,1/5,1/5)=\begin{bmatrix}1/5&0&0&0\\0&1/5&0&0\\0&0&1/5&0\\0&0&0&1\end{bmatrix}$$

$$R_y(\pi/4)=\begin{bmatrix}0.707&0&-0.707&0\\0&1&0&1\\0.707&0&0.707&1\\0&0&0&1\end{bmatrix}$$

$$T(1,2,-3)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\1&2&-3&1\end{bmatrix}$$

分别计算：

$$pS=[1,\ 0,\ 0,\ 1]=p'$$
$$p'R_y=[0.707,\ 0,\ -0.707,\ 1]=p'' \quad (1)$$
$$p''T=[1.707,\ 2,\ -3.707,\ 1]$$

分别采用缩放，旋转，以及平移一系列变换后，结果为：

$$[1.707,\ 2,\ -3.707,\ 1]$$

联合变换：

$$Q=SR_yT;\quad pQ=[1.707,\ 2,\ -3.707,\ 1] \quad (2)$$

同理应用联合变换也会得到相同的结果。

3 实体的位置确定

3.1 单个实体的位置

由于整个虚拟场景中的实体已被分解成不同的.x 文件，因此在写程序时需要把场景中的实体放到合适的位置。一旦构造了某种模型，他们都在自己的坐标系中，但需要把他们放到同一个世界坐标系中[9]。本地空间又叫建模空间(还可叫做自身坐标系)，这是建模的坐标系。本地空间简化了建模的过程。在物体自己的坐标系中建模比在世界坐标系中直接建模更容易。在本地空间中允许构造一个模型而不必考虑物体在世界中相对于其他物体的位置、大小和方向关系。

在本地空间中的物体通过世界变换(world transform)的过程被变换到世界空间中。世界变换通常是用平移、旋转、缩放操作来设置模型在世界中的位置、大小、方向。世界变换就是设置在世界中的各物体彼此的位置、大小和方向关系。世界变换由一个矩阵表示。并且在 Direct3D 中调用 Direct3DDevice9::SetTransform 方法设置它，记住将转换类型设为 D3DTS_WORLD。例如要在世界坐标系中放置一个由.x 文件生成的椅子定位在(-3 2 6)，程序实现如下：

```
D3DXMATRIX thingMat1;
D3DXMatrixTranslation( &thingMat1, -3.0f, 2.0f, 6.0f);
pd3dDevice -> SetTransform(D3DTS_WORLD, &thingMat1);
pd3dDevice -> SetTransform(D3DTS_WORLD, &thingMat1);
g_pMesh -> Render(pd3dDevice);
```

g_pMesh 是指向.x 文件的一个指针，这样就把一个实体放在了世界坐标中。这是个非常简单的实例，没有用到矩阵的旋转和缩放。但是一般很多物体都需要进行这些变换，这个例子展示了世界变换过程。

3.2 实体的大小和方向关系

当实体的位置确定后，根据场景需要来确定实体的大小和方向。例如要把某一实体.x 文件放在(1,1,1)处，并且使其绕 x，y，z 轴分别旋转 90°，扩大为原来的 2 倍，即 x，y，z 坐标分别变为原来的 2 倍。

首先定义 D3DXMATRIXA16，用于保存矩阵：

```
D3DXMATRIXA16 matMov, matRotationX, matRotationY, matRotationZ, matRotationUser1, matScale, mat;
```

单位化每个矩阵，这为以后的编程带来了方便。

```
D3DXMatrixIdentity( &matRotationX);
D3DXMatrixIdentity( &matRotationY);
D3DXMatrixIdentity( &matRotationZ);
D3DXMatrixIdentity( &matRotationUser1);
D3DXMatrixIdentity( &mat);
D3DXMatrixIdentity( &matMov);
D3DXMatrixIdentity( &matScale);
```

保存每一个矩阵，代码如下：

```
D3DXMatrixTranslation( &matMov, 1.0f, 1.0f, 1.0f);
D3DXMatrixRotationX( &matRotationX, D3DX_PI/2);
D3DXMatrixRotationY( &matRotationY, D3DX_PI/2);
D3DXMatrixRotationZ( &matRotationZ, D3DX_
```

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
PI/2);
D3DXMatrixScaling( &matScale, 0.2f, 0.2f, 0.2f);
Mat= matMov * matRotationX * matRotationY * matRotationZ * matScale;
IDirect3DDevice9 * pd3dDevice// pd3dDevice 是 DX 设备
pd3dDevice->SetTransform(D3DTS_WORLD, &Mat);
g_pMesh->Render(pd3dDevice);
```

通过以上代码达到了目的，如果想让实体绕某一向量(2.0 3.0 5.0)旋转若干度(90°)，代码实现为：

```
D3DXMatrixRotationAxis ( &matRotationUser1, &D3DXVECTOR3(2.0f, 3.0f, 5.0f), D3DX_PI/2)
pd3dDevice->SetTransform(D3DTS_WORLD, &matRotationUser1);
g_pMesh->Render(pd3dDevice);
```

不难看出，通过矩阵链接给编程带来了很大的方便。

3.3　多实体场景的实现

实体在场景中的具体位置大小方向已经确定，但要实现真实场景的真实再现，只有这些还是不够的。D3D 支持.x 模型文件，.x 文件包括模型数据，如 vertexbuffer，indexbuffer，以及材质等信息。而且材质信息还可能包括纹理等信息。必须从.x 文件中读取网格(Mesh)并且将它们保存至内存，同时设置析构函数，析构函数用来释放网格所占有的内存。渲染模块会告诉你怎样渲染网格(Mesh)，渲染模型即将模型从设备输出出去。在加载过程重创建了两个数组，分别用于保存模型的材质和纹理。然后会有一个循环，其实它的功能就是从文件(由.x 文件给定)中读取纹理并保存到数组中。在析构函数中，必须释放网格和纹理占用的内存，渲染模块用来循环的处理网格的每个子集，并且用适当的纹理和材质渲染它们。

下面是加载一个实体的具体过程：

```
LPDIRECT3D9 g_pD3D = NULL; //Direct3D 对象
LPDIRECT3DDEVICE9 g_pd3dDevice = NULL; //Direct3D 设备对象
LPD3DXMESH g_pMesh = NULL; //网格模型对象
DWORD g_dwNumMaterials = 0; //网格模型材质数量
```

首先从磁盘文件加载网格模型，用 D3DXLoadMeshFromX 函数来完成把网格载入内存的工作。

```
D3DXLoadMeshFromX ( L"airplane.x", D3DXMESH_MANAGED, g_pd3dDevice, NULL, &pD3DXMtrlBuffer, NULL, &g_dwNumMaterials, &g_pMesh );
```

然后从网格模型中提取材质属性和纹理文件名。

```
D3DXMATERIAL * d3dxMaterials = (D3DXMATERIAL *)pD3DXMtrlBuffer->GetBufferPointer();
g_pMeshMaterials = new D3DMATERIAL9[g_dwNumMaterials];
```

依次逐块提取网格文件名。

```
for( DWORD i=0; i<g_dwNumMaterials; i++ )
{
g_pMeshMaterials[i] = d3dxMaterials[i].MatD3D; //材料属性
g_pMeshTextures[i] = NULL;
if( d3dxMaterials[i].pTextureFilename != NULL &&
strlen(d3dxMaterials[i].pTextureFilename)> 0 )
{
WCHAR filename[256]; //获取纹理文件名
RemovePathFromFileName ( d3dxMaterials[i].pTextureFilename, filename);
}
```

当模型材质属性和纹理读取完后开始渲染场景，其渲染过程如下：由于是逐块读取所以也必须逐块渲染网格模型。

```
for( DWORD i=0; i<g_dwNumMaterials; i++ )
{
//设置材料和纹理
g_pd3dDevice->SetMaterial ( &g_pMeshMaterials[i] );
g_pd3dDevice->SetTexture ( 0, g_pMeshTextures[i] );
g_pMesh->DrawSubset(i); //渲染模型
}
```

至此.x 文件加载完毕。通过上面的处理，图像就可以显示出来了。但在程序的最后必须释放释放网格和纹理占用的内存。为了使实体看起来更加真实，必须加上灯光，视图矩阵和投影矩阵。如果缩放了网格，同时也缩放了法线。这可能会使你的对象看起来发暗　解决的办法是激活 D3DRS

NORMALIZENORMALS 渲染状态。

pD3DDevice －＞ SetRenderState（D3DRS _ NORMALIZENORMALS, TRUE);

下面是通过程序实现的真实场景。

图 1 是未经添加实体的空旷房间,

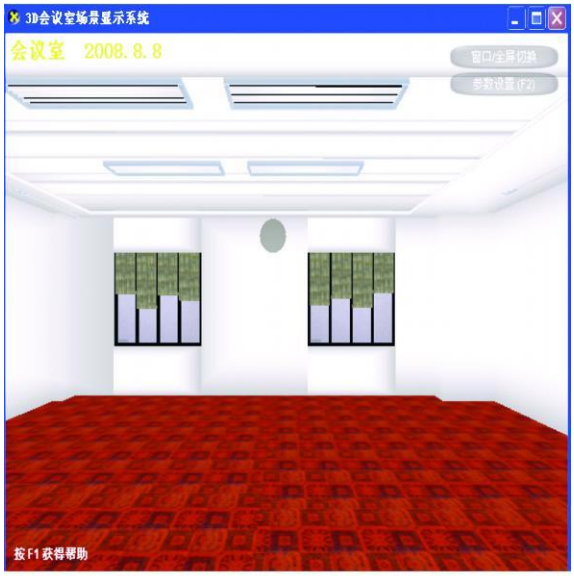


图 1 未经添加实体的空旷房间

图 2 是通过加载.x 文件(会议桌, 椅子, 植物, 沙发), 确定不同实体的位置或相同实体的不同位置, 把它们放在了合理的位置。



图 2 加载. x 文件后的房间

通过 3DSMAX 建造的 3D 模型得到的. x 文件, 经过一系列的转换正确加载后可以得到十分逼真的动画效果, 可以很好的应用于虚拟场景, 例如三维建筑系统, 仓库管理系统等。

参 考 文 献

[1] 于忠德. DirectX 实时渲染技术详解[M]. 重庆: 重庆出版社, 2006.
[2] 童晓然, 吴晟. 用 DirectX 实现地图 3D 展示及模拟行走[J]. 科技广场, 2007(1): 117－119.
[3] 任咏林, 任伟林, 秦勉. 基于 Microsoft DirectX 浅析 COM 组件与多媒体游戏的开发[J]. 北京工商大学学报, 2005, 23(6): 38－42.
[4] Jim Adams. Advanced animation with directX[M]. Thomson: Published by thomsn learning, 2002.
[5] 叶至军, 于忠德. DirectX 实时渲染技术详解[M]. 重庆: 重庆大学出版社, 2006.
[6] 王德才. 精通 DirectX3D 图形与动画程序设计[M]. 北京: 人民邮电出版社, 2007.
[7] 金纯. .x 文件的解读和处理[J]. 福建电脑, 2005(9): 68－69.
[8] 金禾工作室. 3D 游戏程序设计基础[M]. 北京: 北京希望电子出版社, 2006.
[9] 王德才, 张安慧. Direct3D 中的三维坐标变换[J]. 电脑编程技巧与维护, 2007(5): 4－6.

(Ed.: ZW, Z)

辽宁石油化工大学