文章编号：2095-4980(2015)02-0302-06

空间态势场景中 OpenGL 视点控制策略与实现

王浩然，梁彦刚，陈 磊

(国防科学技术大学 航天科学与工程学院，湖南 长沙 410073)

摘 要： OpenGL 视点控制是决定空间态势可视化效果的关键技术之一。本文首先介绍 OpenGL 视点坐标系统相关原理；其次，提出了基于球坐标系的视点控制算法，考虑到 OpenGL 视点设置函数提供的函数接口为笛卡尔坐标系下视点，实现了笛卡尔坐标系下视点与球坐标系下视点间的相关转换；然后推导了球坐标下的视点平移、旋转和缩放；给出了该视点控制方法在空间态势场景中的应用。

关键词： 空间态势；笛卡尔坐标；球坐标；坐标变换；视点变换

中图分类号： TN911.73；TP391.9 **文献标识码：** A **doi：** 10.11805/TKYDA201502.0302

OpenGL viewpoint control strategy and implementation in space situation scene

WANG Haoran，LIANG Yangang，CHEN Lei

(College of Aerospace Science and Engineering，National University of Defense Technology，Changsha Hunan 410073，China)

Abstract： The OpenGL viewpoint control is one of the key techniques to determine the space situational visualization. The principle of OpenGL viewpoint coordinate system is introduced firstly. Secondly, a viewpoint control algorithm based on the spherical coordinates is put forward, which realizes the viewpoint transform between the spherical coordinates and Cartesian Coordinates since the view function interface is based on the Cartesian coordinates system. Then, the viewpoint translation, rotation and scaling in spherical coordinates are derived. The application of the viewpoint control method in the space situation scene are introduced.

Key words： space situation；Cartesian coordinates；spherical coordinates；coordinates transform；viewpoint transform

运用轨道动力学相关知识可求解空间目标在三维空间的几何位置关系，但计算产生的大量轨道数据比较抽象，不易让人们理解空间目标的运行态势。运用计算机仿真技术将大量轨道数据在三维可视化空间进行直观表现，对空间目标运行态势的理解可起到重要辅助作用。目前进行图形视景仿真的开发平台有很多，如开放性图形库 OpenGL，Multigen 公司的 Vega 以及开源著称的 OpenSceneGraph。一方面由于 OpenGL 软件的移植性好，部署简单；另一方面，系统许多功能都需要从底层开发，而商业开发平台的大多数功能模块并非软件系统所需，所以空间态势可视化采用 OpenGL 设计开发。OpenGL 视点控制可方便人们从不同角度了解空间目标运行态势，同时也是决定空间态势可视化效果的关键技术，OpenGL 作为底层图形渲染库，并不提供相应的视点平移、旋转和缩放算法。本文主要研究空间态势可视化中 OpenGL 视点控制策略及其在可视化系统中运用情况。

1 OpenGL 视点坐标定义

OpenGL 函数通常针对二维或三维物体顶点，OpenGL 内将所有顶点看作是有 4 个坐标值的三维齐次顶点。每一列向量 $(x,y,z,w)^{\mathrm{T}}$(正体上标 T 表示转置)中，如果至少一个元素不为 0，则其表示一个齐次顶点[1-2]。三维欧几里德空间点 $(x,y,z)^{\mathrm{T}}$ 用齐次顶点表示为 $(x,y,z,1.0)^{\mathrm{T}}$，二维欧几里德空间点 $(x,y)^{\mathrm{T}}$ 用齐次顶点表示为 $(x,y,0,1)^{\mathrm{T}}$。

OpenGL 采用 4×4 的矩阵计算和存储视点坐标变换，其中左上角 3×3 矩阵表示视点的旋转变换。OpenGL 中

收稿日期： 2013-11-26；**修回日期：** 2015-01-13

基金项目： 国家自然科学基金资助项目空间环境对碎片碰撞预警和轨道演化的影响分析(41240031)

矩阵采用列优先和左乘原则进行矩阵变换，即，$\boldsymbol{v}'=\boldsymbol{T}\cdot\boldsymbol{R}\cdot\boldsymbol{v}$ 表示向量 $\boldsymbol{v}$ 先作旋转变换 $\boldsymbol{R}$ 再作平移变换 $\boldsymbol{T}$ 后得到向量 $\boldsymbol{v}$。OpenGL 默认视点定义为照相机位于原点位置，方向朝向 z 轴负方向，向上方向为 y 轴的正方向，并且视点朝向与视点向上方向要保持垂直关系，如图 1 所示。

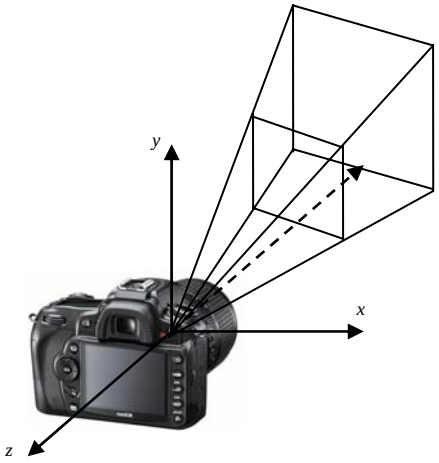

Fig.1 Default camera position
图 1 默认相机位置图

2　笛卡尔坐标视点与球坐标视点转换

OpenGL 中以笛卡尔直角坐标系作为视点输入参数，但在笛卡尔直角坐标系下进行视点旋转、平移和缩放变换相对比较复杂，而在球坐标系下进行视点变换就相对比较简单，本文视点控制方法基于球坐标系实现。由于 OpenGL 视点输入参数和视点变换使用不同坐标系，需实现笛卡尔坐标视点与球坐标视点之间的相关转换。

2.1 笛卡尔坐标视点转换为球坐标视点

令当前视点位置为 $\boldsymbol{Eye}=(EyeX,EyeY,EyeZ)^{\mathrm{T}}$，相机所瞄准的一个参考点的位置为 $\boldsymbol{Center}=(CenterX,CenterY,CenterZ)^{\mathrm{T}}$，视点向上方向为 $\boldsymbol{Up}=(UpX,UpY,UpZ)^{\mathrm{T}}$，则由默认视点坐标系到当前视点坐标系的坐标旋转矩阵 $\boldsymbol{M}$ 可表示为：

$$\boldsymbol{M}=\begin{bmatrix} SX & UpX & -FX & 0 \\ SY & UpY & -FY & 0 \\ SZ & UpZ & -FZ & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

其中，$\boldsymbol{Front}=(FX,FY,FZ)^{\mathrm{T}}$ 为视点中心与视点位置的差，$\boldsymbol{Front}=\boldsymbol{Center}-\boldsymbol{Eye}$，$\boldsymbol{Side}=(SX,SY,SZ)^{\mathrm{T}}$ 为视点朝向向量与视点向上方向向量的叉积的单位化向量：$\boldsymbol{Side}'=\boldsymbol{Front}\otimes\boldsymbol{Up}$，$\boldsymbol{Side}=\dfrac{\boldsymbol{Side}'}{|\boldsymbol{Side}'|}$。

由坐标旋转矩阵 $\boldsymbol{M}$ 可计算出绕 3 个坐标轴旋转(旋转顺序为 321)欧拉角 h,p,r。$Distance=|\boldsymbol{Front}|$ 表示视点与视点中心的距离。参考点的位置 $\boldsymbol{Center}$、欧拉角 h,p,r 和 $Distance$ 的 7 个量就可以确定视点 9 个量的位置关系。欧拉角表示由以 $\boldsymbol{Center}$ 为原点的默认坐标系到当前坐标系的旋转角。

2.2 球坐标视点转换为笛卡尔坐标视点

向量 $\boldsymbol{v}=(x,y,z)^{\mathrm{T}}$ 绕 z 轴旋转 θ 得到向量 $\boldsymbol{v}'=(x',y',z')^{\mathrm{T}}$，$\boldsymbol{v}$ 到 $\boldsymbol{v}'$ 的转化关系[2]如下：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}=\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}\cdot\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\Rightarrow \boldsymbol{v}'=\boldsymbol{M}_z(\theta)\cdot\boldsymbol{v} \tag{1}$$

向量 $\boldsymbol{v}=(x,y,z)^{\mathrm{T}}$ 绕 y 轴旋转 θ 得到向量 $\boldsymbol{v}'=(x',y',z')^{\mathrm{T}}$，$\boldsymbol{v}$ 到 $\boldsymbol{v}'$ 的转化关系[2]如下：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}=\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1.0 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}\cdot\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\Rightarrow \boldsymbol{v}'=\boldsymbol{M}_y(\theta)\cdot\boldsymbol{v} \tag{2}$$

向量 $\boldsymbol{v}=(x,y,z)^{\mathrm{T}}$ 绕 x 轴旋转 θ 得到向量 $\boldsymbol{v}'=(x',y',z')^{\mathrm{T}}$，$\boldsymbol{v}$ 到 $\boldsymbol{v}'$ 的转化关系[2]如下：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}=\begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}\cdot\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\Rightarrow \boldsymbol{v}'=\boldsymbol{M}_x(\theta)\cdot\boldsymbol{v} \tag{3}$$

向量 $\boldsymbol{v}=(x,y,z)^{\mathrm{T}}$ 绕任意向量 $\boldsymbol{p}$ 旋转 θ 得到向量 $\boldsymbol{v}'=(x',y',z')^{\mathrm{T}}$，$\boldsymbol{v}$ 到 $\boldsymbol{v}'$ 的转化关系[2−3]如下：

1) 绕 z 轴旋转 α，绕 y 轴旋转 $-\beta$，即使 x 轴与向量 $\boldsymbol{p}$ 重合，α 为向量 $\boldsymbol{p}$ 在 xy 平面投影与 x 轴正方向夹

角，β 为向量 $\boldsymbol{p}$ 与 xy 平面之间的夹角；

2) 绕 x 轴旋转 θ；

3) 作 1)的逆变换。

$$\boldsymbol{v}' = \boldsymbol{M}_z(-\alpha)\cdot\boldsymbol{M}_y(\beta)\cdot\boldsymbol{M}_x(\theta)\cdot\boldsymbol{M}_y(-\beta)\cdot\boldsymbol{M}_z(\alpha)\cdot\boldsymbol{v} \tag{4}$$

由球坐标表示视点转换为笛卡尔坐标表示视点转换过程如下：

$$\boldsymbol{MRotate} = \boldsymbol{M}_x(r)\cdot\boldsymbol{M}_y(p)\cdot\boldsymbol{M}_z(h) \tag{5}$$

$$\begin{bmatrix} UpX \\ UpY \\ UpZ \\ 1.0 \end{bmatrix} = \boldsymbol{MRotate}\cdot\begin{bmatrix} 0 \\ 1.0 \\ 0 \\ 1.0 \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} FX \\ FY \\ FZ \\ 1.0 \end{bmatrix} = \boldsymbol{MRotate}\cdot\begin{bmatrix} 0 \\ 0 \\ -Distance \\ 1.0 \end{bmatrix} \tag{7}$$

$$\boldsymbol{Eye} = \boldsymbol{Center} - \boldsymbol{Front} \tag{8}$$

所以，$$\begin{bmatrix} EyeX \\ EyeY \\ EyeZ \\ 1.0 \end{bmatrix} = \begin{bmatrix} CenterX \\ CenterY \\ CenterZ \\ 1.0 \end{bmatrix} - \boldsymbol{MRotate}\cdot\begin{bmatrix} 0 \\ 0 \\ -Distance \\ 1.0 \end{bmatrix} \tag{9}$$

3 空间态势场景中的视点变换

为了能多角度、多方位观察空间目标运行态势，需提供友好的人机交互的视点控制方法，空间态势场景中涉及到的视点坐标变换主要有视点平移、视点旋转和视点缩放。

本文采 Visual C++ 6.0 与 OpenGL 搭建基于微软基础类库(Microsoft Foundation Classes，MFC)的空间态势可视化系统框架，使用实用库函数 gluLookAt 进行场景视点设置更新，并通过响应鼠标按键消息实现人机交互的视点控制[4−9]。

3.1 视点平移

视点平移即在保持相机参考点位置、视点向上方向和视点位置相对位置关系不变条件下，整体移动视点。

若屏幕坐标由 $\boldsymbol{P}_0(x_0,y_0)$ 移动到 $\boldsymbol{P}_1(x_1,y_1)$，则视点坐标由 $\boldsymbol{Center}_0$，$dDistance0$ 和 h_0, p_0, r_0 平移变换到 $\boldsymbol{Center}_1$，$dDistance1$ 和 h_1, p_1, r_1 的过程如下。

1) 计算当前视点的旋转矩阵。首先，绕 z 轴旋转 h_0，然后，绕 y 轴旋转 p_0，最后，绕 x 轴旋转 r_0：

$$\boldsymbol{M}_{hpr} = \boldsymbol{M}_x(r_0)\cdot\boldsymbol{M}_y(p_0)\cdot\boldsymbol{M}_z(h_0) \tag{10}$$

2) 计算屏幕坐标移动距离 px 和 py，并将其转换到 $[-1,1]$ 区间，设场景窗口高度为 $MaxY$，场景窗口宽度为 $MaxX$：

$$px = \frac{x_1 - x_0}{MaxX} \tag{11}$$

由于屏幕坐标纵坐标原点为屏幕左上角，所以 $py = \dfrac{y_0 - y_1}{MaxY}$。

3) 屏幕坐标的平移量 $\boldsymbol{Dv}$。视点平移量应该与视点距离存在一定关系，视点距离越大，屏幕坐标移动单位距离对应的视点平移量应该越大。本文设视点平移量与视点距离成线性关系，比例因子为 λ (取 $\lambda = -1.0$)：

$$\boldsymbol{Dv} = (px\times\lambda\times Distance, py\times\lambda\times Distance, 0)^{\mathrm{T}} \tag{12}$$

4) 计算相机参考点坐标平移量：

$$\boldsymbol{CenterOffset} = \boldsymbol{M}_{hpr}\cdot\boldsymbol{Dv} \tag{13}$$

5) 经过平移操作后，得到的新的视点在球坐标下表示形式如下：

$$\begin{cases} \boldsymbol{Center}_1 = \boldsymbol{Center}_0 + \boldsymbol{CenterOffset} \\ dDistance1 = dDistance0 \\ h_1 = h_0 \\ p_1 = p_0 \\ r_1 = r_0 \end{cases} \tag{14}$$

在空间态势场景中，同时按下鼠标左键和右键拖动鼠标可平移观察视点，图 2 为视点平移前后对比场景。通过场景平移，可以了解空域空间目标的分布情况。左图的视点中心 $\boldsymbol{Center}_0$ 为 (0,0,0)，视点距离为 1 180，窗口宽度为 1 366，窗口高度为 768，屏幕坐标移动距离 $px=\frac{320}{1366}$ 和 $py=0$，通过平移操作后，右图中视点中心 $\boldsymbol{Center}_1$ 为 (17.5,275.4,16)，视点距离为 1 180。

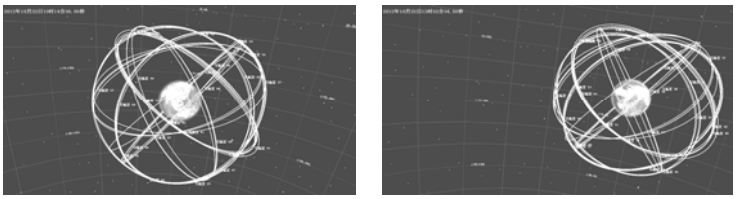

Fig.2 Translating comparison scene
图 2 平移前后对比场景

3.2 视点旋转

视点旋转即在保持相机参考点不变条件下，通过旋转变换改变视点位置和视点向上方向，从而改变视场观察场景实体。若屏幕坐标由 $\boldsymbol{P}_0(x_0,y_0)$ 移动到 $\boldsymbol{P}_1(x_1,y_1)$，则视点坐标由 $\boldsymbol{Center}_0$, $dDistance0$ 和 h_0,p_0,r_0 旋转变换到 $\boldsymbol{Center}_1$, $dDistance1$ 和 h_1,p_1,r_1 的过程如下。

1) 计算当前视点的旋转矩阵。首先，绕 z 轴旋转 h_0，然后，绕 y 轴旋转 p_0，最后，绕 x 轴旋转 r_0：

$$\boldsymbol{M}_{hpr} = \boldsymbol{M}_x(r_0)\cdot\boldsymbol{M}_y(p_0)\cdot\boldsymbol{M}_z(h_0) \tag{15}$$

2) 计算旋转轴 $\boldsymbol{Axis}$ 和旋转角 θ。默认视点所确定的屏幕向左、向上和向内的 3 个向量可表示为：

$$\begin{cases} \boldsymbol{Dx} = \{1,0,0\} \\ \boldsymbol{Dy} = \{0,1,0\} \\ \boldsymbol{Dz} = \{0,0,-1\} \end{cases} \tag{16}$$

当前视点所确定的屏幕向左、向上和向内的 3 个向量可表示为：

$$\begin{cases} \boldsymbol{Side} = \boldsymbol{M}_{hpr}\cdot\boldsymbol{Dx} \\ \boldsymbol{Up} = \boldsymbol{M}_{hpr}\cdot\boldsymbol{Dy} \\ \boldsymbol{Front} = \boldsymbol{M}_{hpr}\cdot\boldsymbol{Dz} \end{cases} \tag{17}$$

鼠标移动方向所确定的向量 $\boldsymbol{MoveP}$ 计算如下：

$$\boldsymbol{MoveY} = (y_0 - y_1)\cdot\boldsymbol{Up} \tag{18}$$

$$\boldsymbol{MoveX} = (x_1 - x_0)\cdot\boldsymbol{Side} \tag{19}$$

$$\boldsymbol{MoveP} = \boldsymbol{MoveX} + \boldsymbol{MoveY} \tag{20}$$

视点旋转轴 $\boldsymbol{Axis}$ 为鼠标移动方向所确定的向量与屏幕向内方向的叉积：

$$\boldsymbol{Axis} = \boldsymbol{MoveP} \otimes \boldsymbol{Front} \tag{21}$$

旋转角的大小与鼠标移动距离相关，距离越大，旋转角度越大。本文旋转角度 θ 采用如下计算方式：

$$\theta = \arcsin\left(\frac{|y_0 - y_1|}{MaxDis} + \frac{|x_1 - x_0|}{MaxDis}\right) \tag{22}$$

式中 $MaxDis$ 为鼠标拖动最大距离。设场景窗口高度为 $MaxY$，场景窗口宽度为 $MaxX$，则

$$MaxDis = \frac{|y_0 - y_1|}{|x_1 - x_0| + |y_0 - y_1|}MaxY + \frac{|x_1 - x_0|}{|x_1 - x_0| + |y_0 - y_1|}MaxX,\ (x_0 \neq x_1 \text{ 或 } y_0 \neq y_1) \tag{23}$$

3) 由式(4)可知，绕旋转轴 $\boldsymbol{Axis}$ 旋转 θ 后的变换矩阵为：

$$\boldsymbol{M}_{Axis\theta} = \boldsymbol{M}_z(-\alpha)\cdot\boldsymbol{M}_y(\beta)\cdot\boldsymbol{M}_x(\theta)\cdot\boldsymbol{M}_y(-\beta)\cdot\boldsymbol{M}_z(\alpha) \tag{24}$$

式中，α 为旋转轴 $\boldsymbol{Axis}$ 在 xy 平面上投影与 X 轴正方向之间夹角，β 为旋转轴 $\boldsymbol{Axis}$ 与 xy 平面之间夹角，当旋转轴 $\boldsymbol{Axis}$ 在 z 方向的分量为正值时，β 为正值，反之为负值。

4) 视点旋转变换后得到新的旋转矩阵 $\boldsymbol{M}'_{hpr}$：

$$\boldsymbol{M}'_{hpr} = \boldsymbol{M}_{Axis\theta}\cdot\boldsymbol{M}_{hpr} \tag{25}$$

由 $\boldsymbol{M}'_{hpr}$ 可计算绕 3 个坐标轴旋转顺序为 321 的欧拉角 h_1,p_1,r_1。

5) 经过旋转操作后，得到的新的视点中相机参考点位置和视点距离不变，即：

$$\begin{cases} \boldsymbol{Center}_1(x_1,y_1,z_1)=\boldsymbol{Center}_0(x_0,y_0,z_0) \\ dDistance1=dDistance0 \end{cases} \tag{26}$$

在空间态势场景中，按下鼠标左键拖动鼠标可旋转观察视点，图 3 为视点旋转前后对比场景。左图的视点中心 $\boldsymbol{Center}_0$ 为 (0,0,0)，视点距离为 1 180，俯仰角 h_0 为 1.72，偏航角 p_0 为 1.25，滚转角 r_0 为 0.10，通过旋转操作后，右图中视点中心和视点距离为不变，俯仰角 h_1 为 1.48，偏航角 p_1 为 0.78，滚转角 r_1 为–0.50。通过场景旋转，可以了解不同区域上空的空间目标的分布态势。

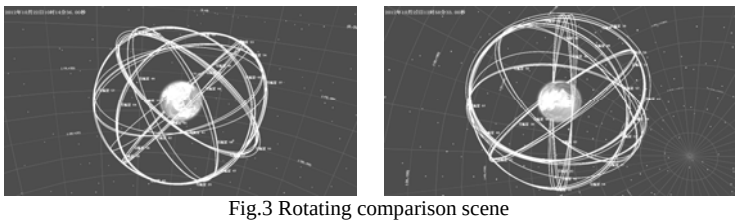

Fig.3 Rotating comparison scene
图 3 旋转前后对比场景

3.3 视点缩放

在球坐标系下，视点缩放就是保持视点中心与视点旋转变换矩阵不变条件下，改变视点与视点中心的距离，从而改变视场观察到的场景的大小。若屏幕坐标由 $\boldsymbol{P}_0(x_0,y_0)$ 移动到 $\boldsymbol{P}_1(x_1,y_1)$，则视点坐标由 $\boldsymbol{Center}_0$，$dDistance0$ 和 h_0,p_0,r_0 缩放变换到 $\boldsymbol{Center}_1$，$dDistance1$ 和 h_1,p_1,r_1 的过程如下。

1) 视点缩放只需考虑鼠标在 y 方向的移动，视点缩放比例因子采用如下计算方式：

$$Scale=1+(y_0-y_1)/MaxY \quad (MaxY \text{ 场景窗口高度}) \tag{27}$$

2) 经过缩放操作后，得到的新的视点在球坐标下表示形式如下：

$$\begin{cases} dDistance1=dDistnce\cdot Scale \\ \boldsymbol{Center}_1=\boldsymbol{Center}_0 \\ h_1=h_0 \\ p_1=p_0 \\ r_1=r_0 \end{cases} \tag{28}$$

在空间态势场景中，按下鼠标右键拖动鼠标可以放大或缩小观察视点，图 4 为视点缩放前后对比场景。左图的视点中心 $\boldsymbol{Center}_0$ 为 (0,0,0)，视点距离为 1 180，屏幕坐标移动距离 px=0 和 py=406，窗口宽度为 1 366，窗口高度为 768，通过视点拉远操作后，右图中视点中心 $\boldsymbol{Center}_1$ 为 (0,0,0)，视点距离为 1 803.8。将场景拉近时可观察低轨卫星的运行态势，将场景拉远时可观察中高轨卫星的运行态势。

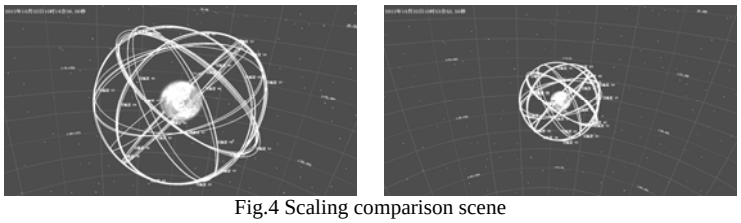

Fig.4 Scaling comparison scene
图 4 缩放前后对比场景

4 结论

本文提出了基于球坐标系的视点变换算法。该算法空间几何意义明确，采用矩阵计算，不涉及三角函数计算，计算相对简单。采用 Visual C++ 6.0 与 OpenGL 实现该变换算法，为空间态势系统视点控制提供了友好、直观的人机交互界面。同时，该视点控制方法具有较好的通用性和可移植性，可在虚拟现实、飞行仿真等可视化系统中应用。

参考文献：

[1] Mason Woo. OpenGL 编程权威指南[M]. 北京:中国电力出版社, 2001. (Mason Woo. OpenGL Programming Guide[M]. Beijing:China Electric Power Press, 2001.)

[2] Donalad Hearn, Baker M Pauline. 计算机图形学[M]. 2 版. 北京:电子工业出版社, 2004. (Donalad Hearn, Baker M Pauline. Computer Graphics[M]. 2nd ed. Beijing:Publishing House of Electronics Industry, 2004.)

[3] 顾荣军,王朴军,王苗苗,等. 雷达模拟训练系统电视跟踪视景仿真实现[J]. 太赫兹科学与电子信息学报, 2014, 12(1):71–75. (GU Rongjun,WANG Pujun,WANG Miaomiao,et al. Implementation of TV tracking scene simulation for radar simulator[J]. Journal of Terahertz Science and Electronic Information Technology, 2014,12(1):71–75.)

[4] 柯晶,赵敏,徐军,等. 基于 OpenGL 的小卫星在轨运行仿真系统研究[J]. 计算机测量与控制, 2007,15(7):940-942. (KE Jing,ZHAO Min,XU Jun,et al. Simulation system of in-orbit small satellite movement based on OpenGL[J]. Computer Measurement & Control, 2007,15(7):940-942.)

[5] 李盛,万敏,吴向东. 基于 OpenGL 视点坐标系的三维场景旋转算法[J]. 计算机工程与应用, 2006(16):83-85. (LI Sheng,WAN Min,WU Xiangdong. Algorithm of 3D scene rotation based on eye coordinate of OpenGL[J]. Computer Engineering and Application, 2006(16):83-85.)

[6] 陈文彤,刘朝军,陈曾平. 基于 OpenGL 的空间目标观测可视化仿真[J]. 系统仿真学报, 2007,19(3):567-569. (CHEN Wentong,LIU Chaojun,CHEN Zengping. Visual simulation of space-objects observation based on OpenGL[J]. Journal of System Simulation, 2007,19(3):567-569.)

[7] 张丹. 基于 OpenGL 的飞行视景仿真研究[D]. 北京:北京邮电大学, 2010. (ZHANG Dan. The research of flight visual simulation based on OpenGL[D]. Beijing:Beijing University of Posts and Telecommunications, 2010.)

[8] 姚裕华. 虚拟校园漫游系统的设计与实现[D]. 成都:电子科技大学, 2011. (YAO Yuhua. Design and implementation of virtual campus roaming system[D]. Chengdu,China:University of Electronic Science and Technology of China, 2011.)

[9] 徐利明,姜昱明. 可漫游的虚拟场景建模与实现[J]. 系统仿真学报, 2006,18(1):120-124. (XU Liming,JIANG Yuming. Modeling and realization of roaming virtual scene[J]. Journal of System Simulation, 2006,18(1):120-124.)

作者简介：

王浩然(1984-)，男，长沙市人，硕士，主要研究方向为视景仿真与虚拟现实. email: wanghaoran6@163.com.

梁彦刚(1979-)，男，陕西省宝鸡市人，博士，讲师，主要研究方向为武器系统建模与仿真，系统分析与评估.

陈　磊(1974-)，男，陕西省咸阳市人，博士，教授，主要研究方向为飞行器动力学制导与控制、空间目标碰撞预警.