

OpenCASCADE Coordinate Transforms

eryar@163.com

Abstract. The purpose of the OpenGL graphics processing pipeline is to convert 3D descriptions of objects into a 2D image that can be displayed. In many ways, this process is similar to using a camera to convert a real-world scene into a 2D print. To accomplish the transformation from 3D to 2D, OpenGL defines several coordinate spaces and transformations between those spaces. Each coordinate space has some properties that make it useful for some part of the rendering process. The transformations defined by OpenGL afford applications a great deal of flexibility in defining 3D-to-2D mapping. So understanding the various transformations and coordinate spaces used by OpenGL is essential. The blog use GLUT to demonstrate the conversion between 3D and 2D point.

Key Words. OpenCASCADE, OpenGL, Coordinate Transform,

1. Introduction

交互式计算机图形学的迅速发展令人兴奋，其广泛的应用使科学、艺术、工程、商务、工业、医药、政府、娱乐、广告、教学、培训和家庭等各方面均获得巨大收益。与图形交互的方式目前主要还是键盘和鼠标或 pad 上的触摸方式。交互的方式主要分为两种，一是在二维的屏幕或窗口上对三维的模型进行控制；一是对三维视图进行控制（在 OpenSceneGraph 中对应拖拽器和漫游器）。这就需要对计算机图形学中的坐标变换的原理有所理解。

三维场景视图的计算机生成步骤有点类似拍一张照片的过程，但与使用照相机相比，利用图形软件生成的场景视图有更大灵活性和更多的选择。我们可以选择平行投影或透视投影，还可以有选择地沿视线消除一些场景等。

通过从三维模型到二维屏幕窗口的实现过程理解，方便理解计算机中对场景观察的实现。场景的观察即注重一个从三维世界转换到二维屏幕的过程。假设场景的观察者使用一台相机来全程记录世界的变化，那么相机位置的移动、角度偏转、焦距变化及镜头类型的选用都会改变底片上呈现的内容，也就是观察这个世界的方式。

通过从二维屏幕到三维场景中实现的理解，方便用户通过二维的屏幕窗口的程序界面来操纵三维世界中的模型，如三维中模型的选择及对选择中的模型进行相关的变换，如移动、旋转等。

本文结合 OpenCASCADE 中的类 Graphic3d_Camera 及 GLUT 来说明这两个过程，进而对 OpenCASCADE 中可视化模块的理解。通过对 OpenGL 中坐标变换原理的理解，为 OpenCASCADE 中对视图的操作及拓朴模型（点、线、面和体）选择的实现的理解打下基础，良好的交互可以让场景中的模型充满生机。

2. From 3D to 2D

从三维的模型到二维屏幕，OpenGL 的渲染管线分为以下几个部分：

- ❖ 模型视点变换 MODELVIEW TRANSFORMATION;
- ❖ 投影变换 PROJECTION TRANSFORMATION;
- ❖ 视口变换 VIEWPORT TRANSFORMATION;

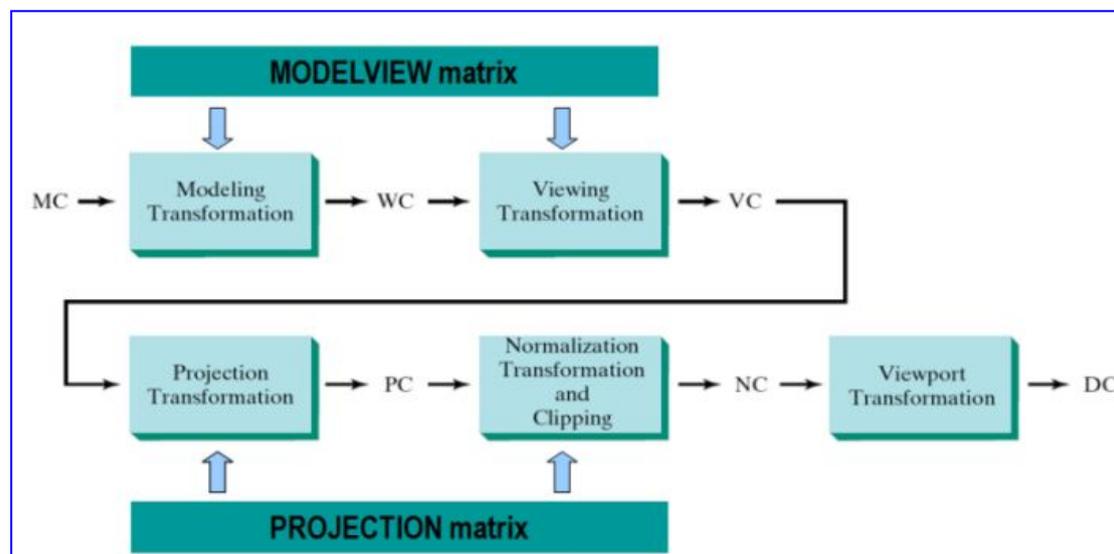


Figure 2.1 OpenGL Vertex Transformation Pipeline

如上图所示为从模型坐标系到设备坐标系变换的管线，即模型经过这些变换后，最终才能显示在设备屏幕上。

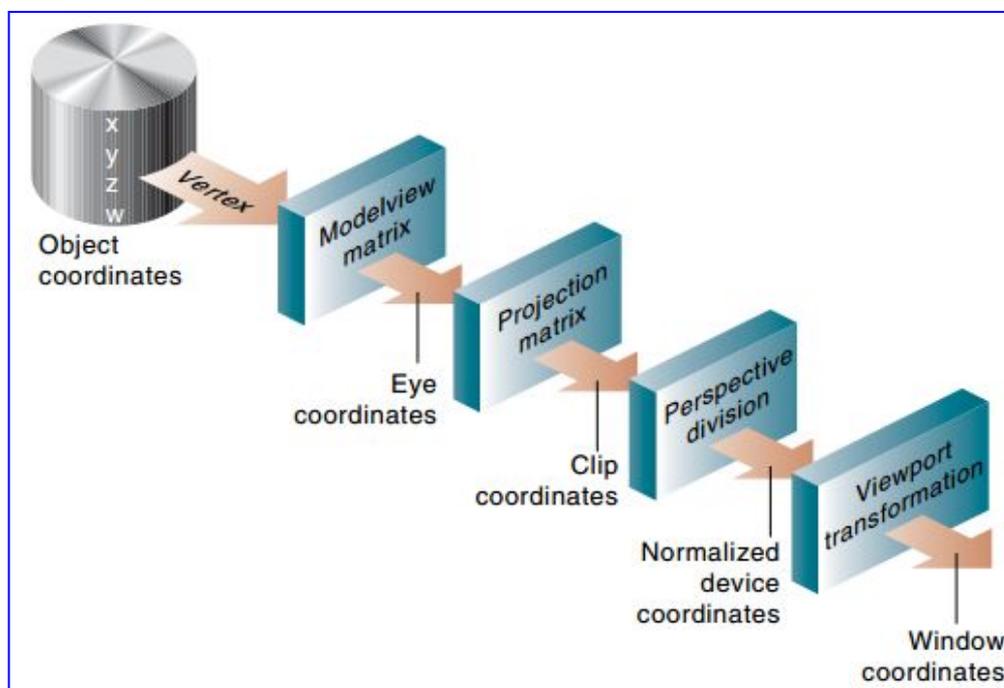


Figure 2.2 Stages of Vertex Transformation(OpenGL Programming Guide)

在 OpenGL 中每个变换都对应了一个矩阵运算，如对模型视点的变换对应了模型视点矩阵，可以通过 `glMatrixMode(GL_MODELVIEW)` 来定义当前的矩阵模式，再通过

glLoadMatrix 来指定这个变换矩阵。同理，对应投影变换的相关函数为 glMatrixMode(GL_PROJECTION)。

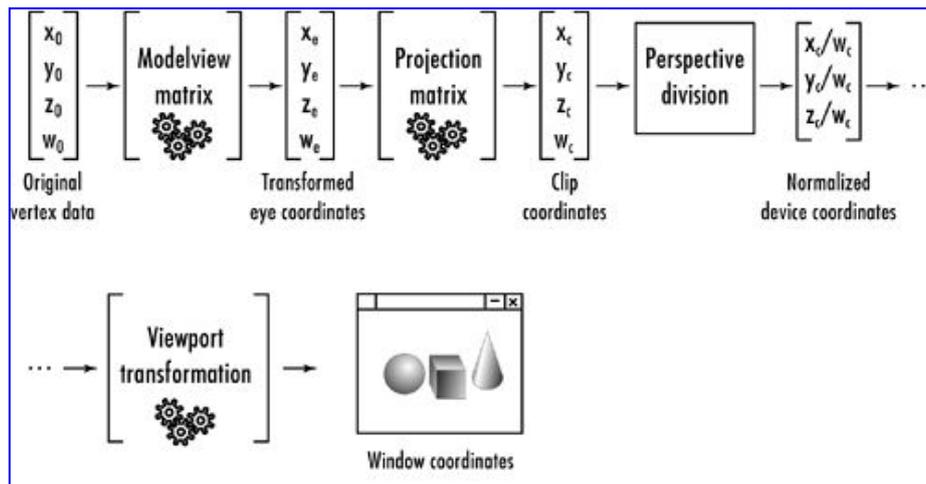


Figure 2.3 The vertex transformation pipeline(OpenGL SuperBible)

2.1 模型视点变换 Model-View Transformation

模型视点变换相当于调整要拍摄的物体的位置、姿态及调整相机使其对准景物的过程，前者称为模型变换，后者称为视点变换。在 OpenGL 中，模型变换和视点变换的结果被合并为模型视点变换矩阵（Model-View Matrix）。在绘制物体前可以使用 glTranslate, glRotate 等来实现模型的位置、姿态变换。与模型变换对应，我们还可以采用视点变换改变观察点的位置和方向，即改变相机的位置和拍摄角度，从而改变最终拍摄的结果。可以将视点变换的行为视作模型变换的逆操作，如将相机在 X 方向移动 N 米，就相当于将整个场景在 -X 方向移动了 N 米。

经过模型视点变换后，即可认为场景从世界坐标系（World Coordinate System, WCS）转换到了相机坐标系（Viewing Coordinate System, VCS）。对于世界坐标系中任意向量 V_{WCS} 将其变换到相机坐标系的公式为：

$$V_{VCS} = V_{WCS} \cdot M_{model-view}$$

在 VCS 坐标系中，习惯用 n,u,v 三个分量来表示坐标基向量，如下图所示为从世界坐标系变换到视点坐标系的变换：

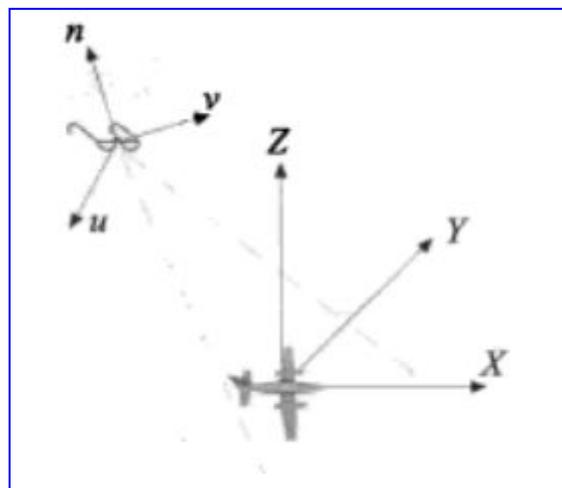


Figure 2.4 From World Cooidante System to Viewing Coordinate System

其中模型视点变换矩阵内容如下所示：

$$M_{model-view} = \begin{bmatrix} ux & vx & nx & 0 \\ uy & vy & ny & 0 \\ uz & vz & nz & 0 \\ -ex & -ey & -ez & 1 \end{bmatrix}$$

2.2 投影变换 Projection Transformation

投影变换相当于拍照时通过选择镜头和调整焦距，将景物投射到二维底片的过程。主要操作对象是一个立方体或棱台形状的视景体。在视景体之外的对象将被裁剪，因而不会投影到二维面上。

立方体的视景体称为平行投影视景体，通过使用 `glOrtho()` 函数创建，其结果是生成一个正射投影矩阵，如下图所示：

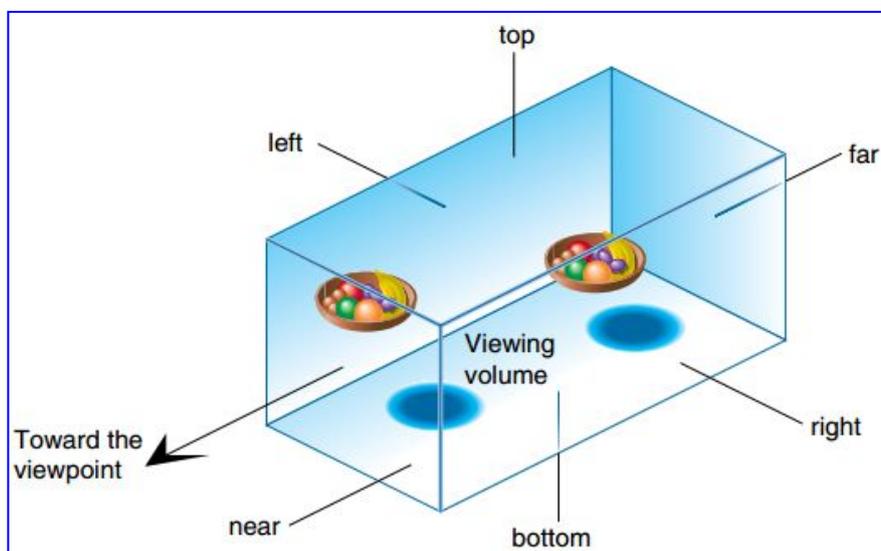


Figure 2.5 Orthographic Viewing Volume(OpenGL Programming Guide)

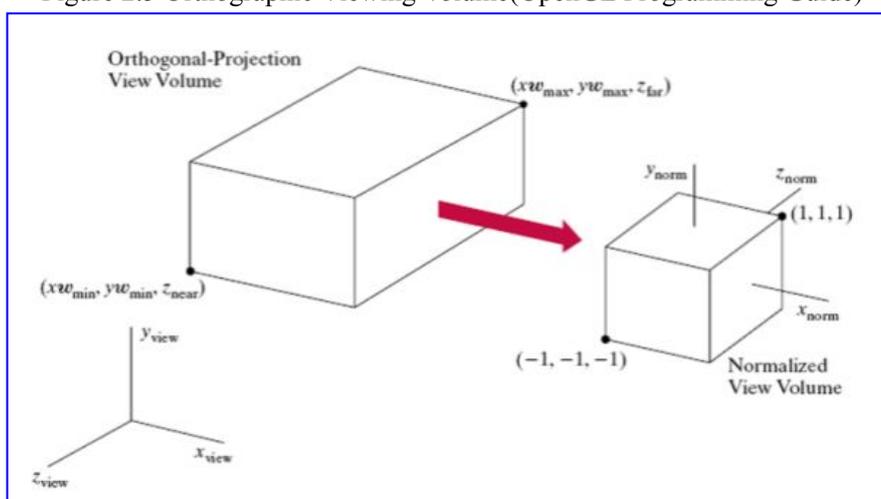


Figure 2.6 Orthogonal Projection View Volume to Normalized View Volume

由于屏幕坐标系经常指定为左手系，因此规范化观察体也常指定为左手系统。正投影观察体的规范化变换矩阵是：详细推导请参考《Computer Graphics with OpenGL》。

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{xw_{max} - xw_{min}} & 0 & 0 & \frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} \\ 0 & \frac{2}{yw_{max} - yw_{min}} & 0 & \frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} \\ 0 & 0 & \frac{-2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

棱台形状的视景物又称为视锥体，主要用于完成透视投影的工作，这种投影更符合人们的心理习惯，即离视点近的物体较大，离视点远的物体较小。在 OpenGL 中由函数 `glFrustum` 和 `gluPerspective` 来实现。

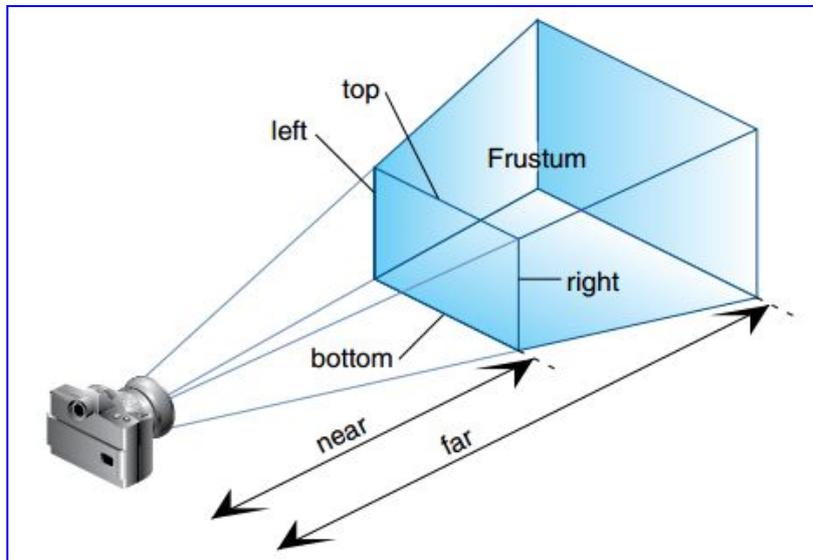


Figure 2.7 Perspective Viewing Volume Specified by `glFrustum`

一般的透视投影规范化变换矩阵如下所示：详细推导请参考《Computer Graphics with OpenGL》。

$$M_{pers,norm} = \begin{bmatrix} \frac{-2z_{near}}{xw_{max} - xw_{min}} & 0 & \frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} & 0 \\ 0 & \frac{-2z_{near}}{yw_{max} - yw_{min}} & \frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & \frac{2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & \frac{z_{near} - z_{far}}{-1} & \frac{z_{near} - z_{far}}{0} \end{bmatrix}$$

相机坐标系的向量 V_{vcs} 变换到投影坐标系下（Clipping Coordinate System, CCS）的计算公式为：

$$V_{ccs} = V_{vcs} \cdot M_{projection}$$

2.3 视口变换 Viewport Transformation

这一步的意义是将投影变换得到的结果反映到指定的屏幕窗口上去,有点类似将底片冲洗到照片上。如果照片的大小与底片不相符,那么将可能产生成像的放大或缩小。OpenGL中函数 `glViewport()` 指定了视口变换的结果区域。这一变换的同时还意味着场景最终变换到窗口坐标系 (Device Coordinate System, DCS, 又称为设备坐标系)。

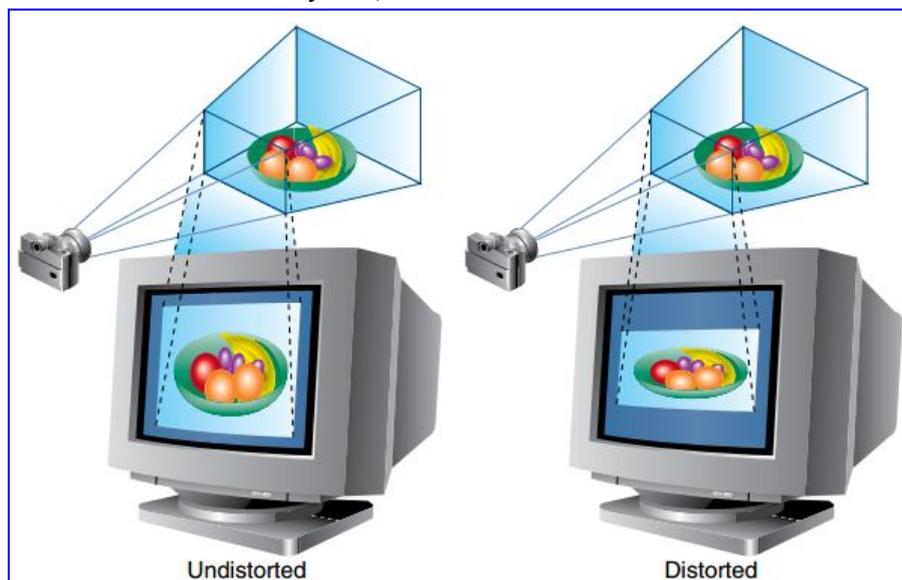


Figure 2.8 Mapping the Viewing Volume to the Viewport

从规范化的视景物到屏幕坐标系的变换矩阵是:

$$M_{normviewvol,screen} = \begin{bmatrix} \frac{xv\ max - xv\ min}{2} & 0 & 0 & \frac{xv\ max + xv\ min}{2} \\ 0 & \frac{yv\ max - yv\ min}{2} & 0 & \frac{yv\ max + yv\ min}{2} \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

综上所述,场景从三维世界坐标系变换到屏幕窗口的所有变换过程进行叠加,则得到如下的变换公式:

$$\begin{aligned} V_{dcs} &= V_{wcs} \cdot M_{model-view} \cdot M_{projection} \cdot M_{viewport} \\ M_{projection} &= M_{ortho,norm} \quad or \quad M_{pers,norm} \\ M_{viewport} &= M_{normviewvol,screen} \end{aligned}$$

这三个矩阵可以合并为一个,称为 MVPW 矩阵,它决定了任意三维空间中的对象在二维屏幕上表达时的变换过程。

3. From 2D to 3D

在理解 OpenGL 的整个渲染管线后,再来理解将屏幕上一点转换为世界坐标就比较容易了。从图形渲染管线的开始到结束,一个模型坐标系中的坐标被变换到了屏幕坐标,那么现在把整个过程倒过来的话,屏幕上一个二维点坐标也可转换到世界坐标系中。即将 MVPW 矩阵求逆即可。如下图所示为从屏幕坐标到世界坐标的变换过程:

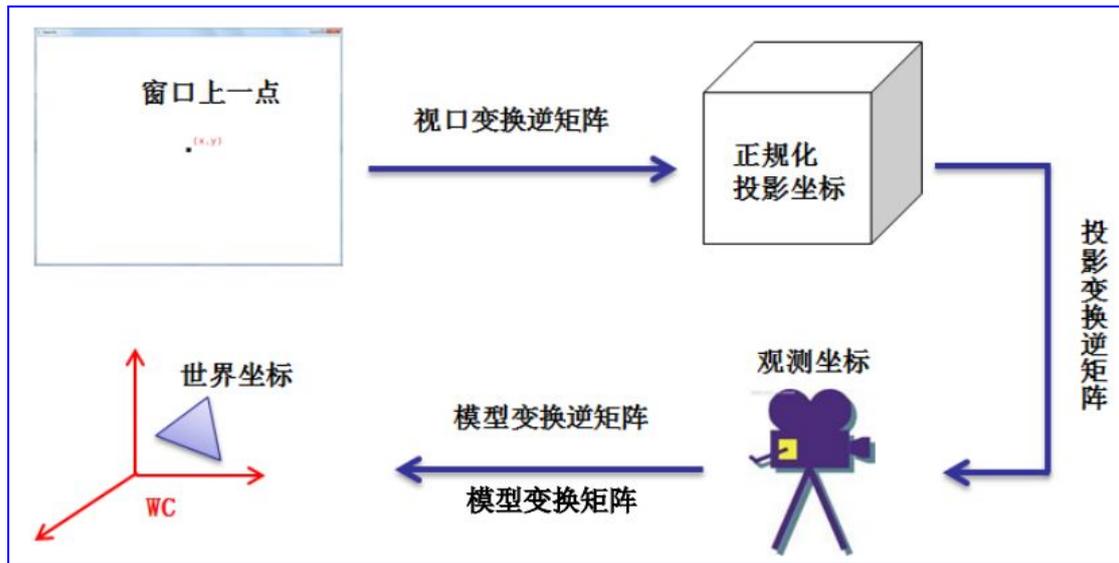


Figure 3.1 Viewport coordinate to World Coordinate(By Zhangci)

当理解从屏幕坐标变换到世界坐标后,也就可以方便理解 Picking 技术的实现了。即可以通过在二维的屏幕窗口中来选择拓扑形状的点、线和面等模型了。可以通过鼠标直接拾取模型后,就可以方便地对模型的位置角度等进行编辑,如 OpenSceneGraph 中的 Dragger:



Figure 3.2 OpenSceneGraph TranslateAxisDragger

如上图所示,用鼠标在任意一个轴上进行拖拽即可以对三维模型的位置进行编辑。

4. Code Example

结合 OpenCASCADE 的类 `Graphic3d_Camera` 和 GLUT 可以方便来验证 OpenGL 中的坐标变换过程。其中类 `Graphic3d_Camera` 已经实现从世界坐标系到规范坐标系的变换，但仍需要从规范坐标系到屏幕窗口变换的矩阵，根据计算公式实现如下代码所示：

```
void BuildNorm2ViewportMatrix(GLsizei theWidth, GLsizei theHeight)
{
    // row 0
    theNorm2Viewport.ChangeValue(0, 0) = (theWidth - 0) / 2.0;
    theNorm2Viewport.ChangeValue(0, 1) = 0.0;
    theNorm2Viewport.ChangeValue(0, 2) = 0.0;
    theNorm2Viewport.ChangeValue(0, 3) = (0 + theWidth) / 2.0;

    // row 1
    theNorm2Viewport.ChangeValue(1, 0) = 0.0;
    theNorm2Viewport.ChangeValue(1, 1) = (0 - theHeight) / 2.0;
    theNorm2Viewport.ChangeValue(1, 2) = 0.0;
    theNorm2Viewport.ChangeValue(1, 3) = (theHeight + 0) / 2.0;

    // row 2
    theNorm2Viewport.ChangeValue(2, 0) = 0.0;
    theNorm2Viewport.ChangeValue(2, 1) = 0.0;
    theNorm2Viewport.ChangeValue(2, 2) = 0.5;
    theNorm2Viewport.ChangeValue(2, 3) = 0.5;

    // row 3
    theNorm2Viewport.ChangeValue(3, 0) = 0.0;
    theNorm2Viewport.ChangeValue(3, 1) = 0.0;
    theNorm2Viewport.ChangeValue(3, 2) = 0.0;
    theNorm2Viewport.ChangeValue(3, 3) = 1.0;
}
```

当用鼠标点击直线的起点时，可以得到相应的屏幕坐标及世界坐标，如下图所示：

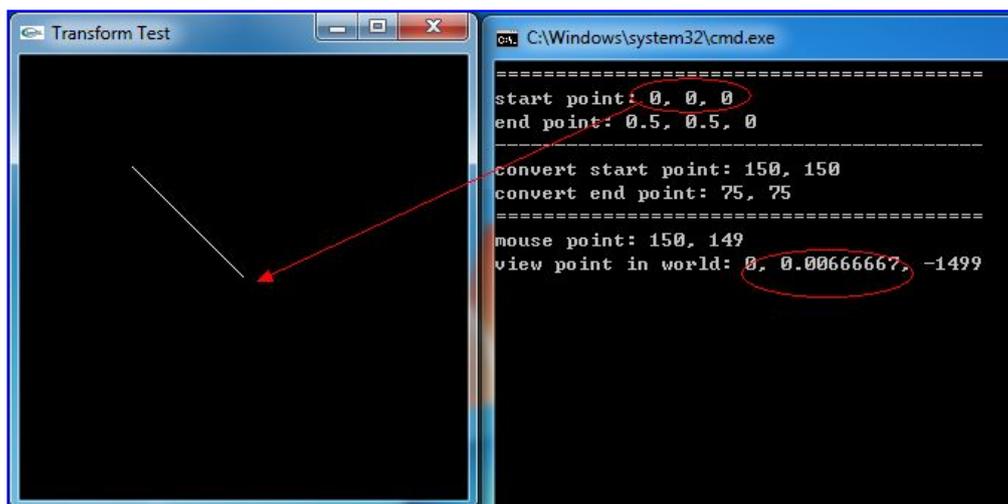


Figure 4.1 Test Coordinate Transformation

其中世界坐标系中的坐标为坐标原点 $(0, 0, 0)$ ，转换到屏幕坐标为 $(150, 150)$ ，与鼠标拾取的点一致。再将鼠标拾取的点转换到三维世界坐标也为原点，只是深度值 Z 不同。

5. Conclusion

综上所述，在 OpenGL 中世界坐标系中的模型经过模型视点变换、投影变换和视口变换最终得到了二维窗口中的显示图像。而这个逆过程即从屏幕坐标到三维世界坐标系的变换对交互更有意义，即拾取技术的实现。

结合 OpenCASCADE 中的类 Graphic3d_Camera 和 GLUT 来试验了这些变换过程，为理解拓朴形状点、线和面等的选择交互的实现打下基础，进一步理解 Visualization 模块。

6. References

1. Dave Shreiner. OpenGL Programming Guide(7th Edition). Addison-Wesley. 2010
2. Richard S. Wright Jr., Benjamin Lipchak. OpenGL SuperBible. Sams. 2004
3. Randi J. Rost. OpenGL Shading Language(2nd Edition). Addison-Wesley. 2006
4. Donald Hearn, M. Pauline Baker. Computer Graphics with OpenGL. 2004
5. Zhangci. OpenGL Rendering Pipeline&Coordinate Transformation.
<http://blog.csdn.net/zhangci226/article/details/5314184>
6. 王锐, 钱学雷. OpenSceneGraph 三维渲染引擎设计与实践. 清华大学出版社. 2009