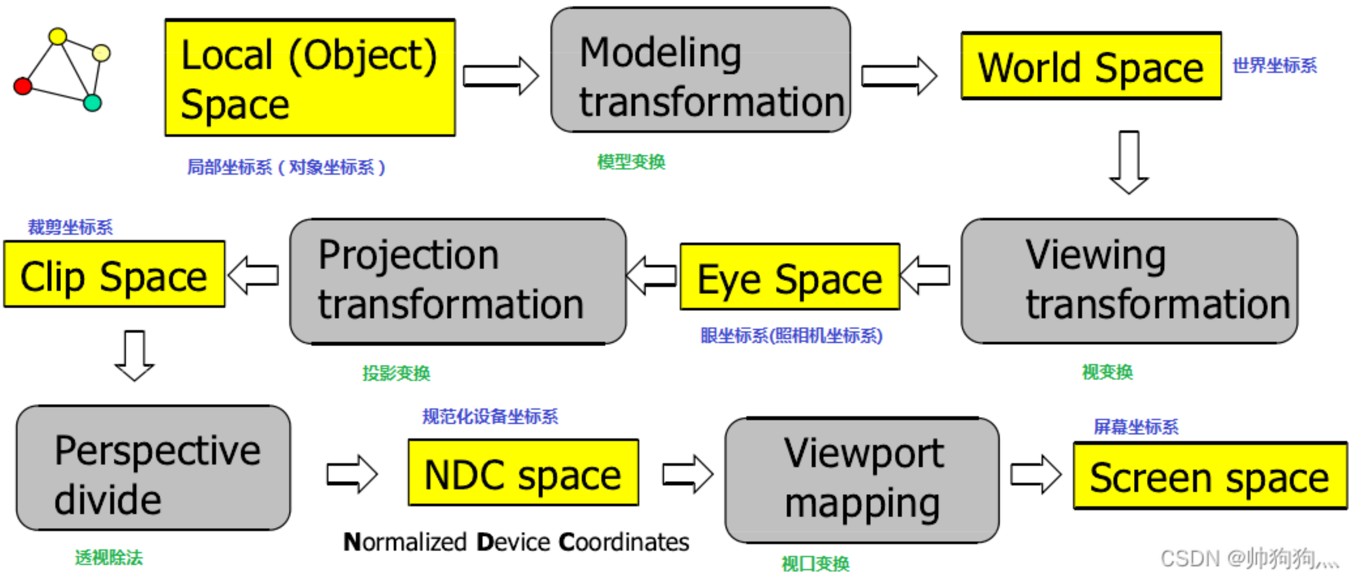
位置变换

💡手撸一遍顶点变化，从局部空间开始，最后输出到屏幕的像素点 **<源于西山居笔试编程题~>**



⚠️注意的点：

- view矩阵，通过right、up、forward、cameraW构造view矩阵要注意什么，得到的view矩阵需要取逆吗
- 如何求矩阵的逆，对于正交矩阵来说，它的逆是如何
- 透视除法中，w分量为0怎么办
- 不同api的差异：比如ndc空间的范围是多少（不同api下）
- 映射到屏幕空间的像素点如何计算，需要四舍五入（`round`）吗

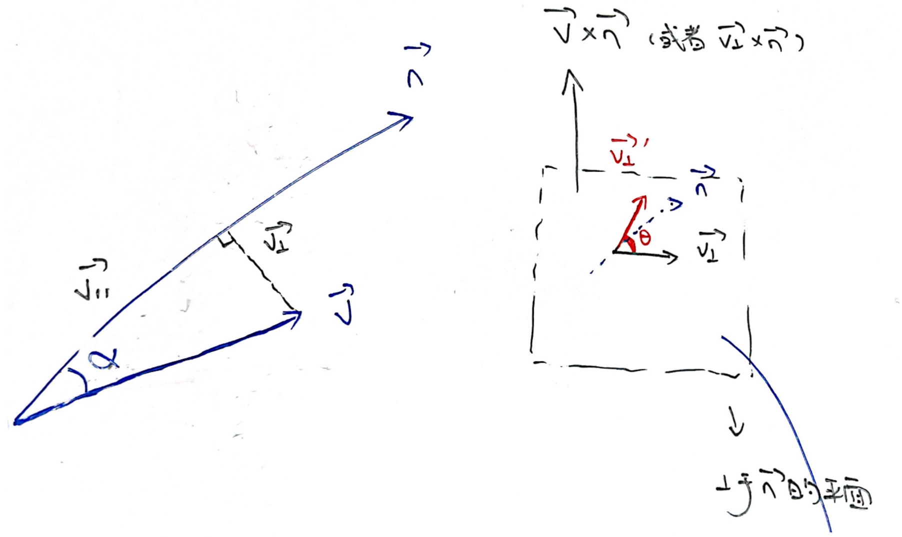
代码：在main.cpp中，数据在data.txt中，使用重定向将数据导入「代码基于DirectX书写」

```
g++ ./main.cpp -o main -std=c++11
./main < data.txt
```

‼️**Api之间的差别**：

	OpenGL	DirectX
坐标系	右手坐标系	左手坐标系
摄像机空间正前方	z负半轴	z正半轴
矩阵/向量	列矩阵	行矩阵
向量/矩阵乘法	左乘	右乘
ndc空间坐标范围	$x,y,z \in [-1,1]^3$	$x,y \in [-1,1]^2$ $\boldsymbol{z \in [0,1]}$

罗德里格旋转公式：



$$
\begin{gathered}
\text{假设：}\vec{n}\text{是单位向量，向量}\vec{v}\text{绕旋转轴}\vec{n}\text{旋转}\theta\text{角度} \\
\vec{v} = \vec{v_{\parallel}} + \vec{v_{\perp}} \\
\vec{v_{\parallel}} = (\vec{n}\cdot\vec{v})\vec{n} \quad \vec{v_{\perp}} = \vec{v} - \vec{v_{\parallel}} \\
\text{对}\vec{v_{\perp}}\text{进行旋转，我们建立一个垂直于}\vec{n}\text{的平面} \\
x\text{轴：}\vec{v_{\perp}} \\
y\text{轴：}(\vec{v_{\perp}} \times \vec{n})\text{或者}(\vec{v} \times \vec{n}) \\
\because ||\vec{v} \times \vec{n}|| = ||\vec{v}|| \cdot sin\alpha = ||\vec{v_{\perp}}|| \\
\vec{v_{\perp}}' = ||\vec{v_{\perp}}|| \cdot cos\theta \cdot \frac{\vec{v_{\perp}}}{||\vec{v_{\perp}}||} + ||\vec{v_{\perp}}|| \cdot sin\theta \cdot \frac{\vec{v} \times \vec{n}}{||\vec{v} \times \vec{n}||} \\
= \vec{v_{\perp}} \cdot cos\theta + \vec{v} \times \vec{n} \cdot sin\theta \\
\therefore \vec{v}' = \vec{v_{\parallel}} + \vec{v_{\perp}}' \\
\vec{v}' = (\vec{n}\cdot\vec{v})\vec{n} + (\vec{v} - (\vec{n}\cdot\vec{v})\vec{n}) \cdot cos\theta + (\vec{v} \times \vec{n}) \cdot sin\theta
\end{gathered} \tag{1}
$$

所以总公式为：

$$
\begin{gathered}
R_n(v) = proj_n(v) + R_n(v_{\perp}) \\
= cos\theta\mathbf{v} + (1 - cos\theta)(\mathbf{n}\cdot\mathbf{v})\mathbf{n} + sin\theta(\mathbf{n} \times \mathbf{v})
\end{gathered} \tag{2}
$$

对应的旋转矩阵为：

$$
\begin{gathered}
n = (x, y, z) \quad \theta \\
R_n = \begin{bmatrix} cos\theta + (1 - cos\theta)x^2 & (1 - cos\theta)xy + sin\theta z & (1 - cos\theta)xz - sin\theta y \\ (1 - cos\theta)xy - sin\theta z & cos\theta + (1 - cos\theta)y^2 & (1 - cos\theta)yz + sin\theta x \\ (1 - cos\theta)xz + sin\theta y & (1 - cos\theta)yz - sin\theta x & cos\theta + (1 - cos\theta)z^2 \end{bmatrix}
\end{gathered} \tag{3}
$$

该推导的关键点在于：旋转平面的y轴方向，如果能用v代替v垂直，则问题迎刃而解～

线性变换的推导：=> 矩阵每行是标准基向量做某种变换后的新向量

线性变换需要满足的条件：

$$
\begin{gathered}
\tau(\vec{u} + \vec{v}) = \tau(\vec{u}) + \tau(\vec{v}) \\
\tau(k\vec{u}) = k\tau(\vec{u})
\end{gathered} \tag{4}
$$

如果一个变换是线性变换，那么它就可以写成矩阵的形式：

$$
\begin{aligned}
p &= (x, y, z) = x\vec{i} + y\vec{j} + z\vec{k} \\
\tau(p) &= \tau(x\vec{i} + y\vec{j} + z\vec{k}) = x \cdot \tau(\vec{i}) + y \cdot \tau(\vec{j}) + z \cdot \tau(\vec{k}) \\
&= \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} \tau(\vec{i}) \\ \tau(\vec{j}) \\ \tau(\vec{k}) \end{bmatrix}
\end{aligned} \tag{5}
$$

> 注意：具体乘法的时候又是"交叉"的，也就是说x'=x·m[0][0]+y·m[0][1]+z·m[0][2]
> m矩阵的一列贯穿了三个行向量，但是x只会乘上\tau(i)的对应分量

那就只需要证明S、R是线性变换，加上T是仿射变化即可：

- 缩放操作：是线性变换，但不是正交变换
  - 因为**是线性变换**，所以可以写成行矩阵的形式
    - 写成行矩阵不代表其是正交变换，正交变换需要满足基向量是单位长度，且相互正交，缩放操作打破了“单位长度”这一点
  - 但因为它**不是正交变换**，所以它的逆矩阵不是它的转置矩阵（这很明显）
  - 逆矩阵：对角线变成倒数
- 旋转操作：是线性变换，也是正交变换
  - 线性变换可以写成行向量的形式，每个行向量是单位向量，所以它又是一个正交变换
  - 逆矩阵：3x3部分进行转置矩阵
- 平移操作：都不是，仿射变换=线性变换+平移
  - 逆矩阵：平移部分取反

那么如果我们要想求model矩阵的逆矩阵：

- 方法0：直接用求逆函数
- 方法1：如果我们分别知道S、R、T，那么我们分别对SRT求逆矩阵，然后相乘
- 方法2：如果我们只知道model矩阵，我们想办法从model矩阵的特征中，提取出S、R、T矩阵
  - $$
\begin{aligned}
M &= SRT \\
&= \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ? & ? & ? & 0 \\ ? & ? & ? & 0 \\ ? & ? & ? & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}
\end{aligned} \tag{6}
$$
  - 我们先只考虑SR相乘，我们会发现Sx会乘在第一行每个元素，Sy会乘在第二行每个元素，Sz会乘在第三行每个元素，本身R中3x3部分，每行/列都是单位向量 => 最终的M矩阵的3x3部分，**每行的模就是缩放的倍数**，剩下的就是旋转部分，最后观察平移部分得到平移位置

‼️坐标系之间的变换：尤其是local => world => view

两个坐标系之间的变换，假设坐标系A的xyz轴和原点相对于坐标系B的坐标为uvw和Q，那么从坐标系A到坐标系B的坐标转换矩阵为**【A在B坐标系的坐标，得到A转换到B的变化矩阵】**：

$$
\begin{bmatrix} \leftarrow u \rightarrow \\ \leftarrow v \rightarrow \\ \leftarrow w \rightarrow \\ \leftarrow Q \rightarrow \end{bmatrix} \tag{7}
$$

那么从B到A的坐标变换矩阵为该矩阵的逆矩阵，也就是uvw这个3x3局部矩阵的逆和Q取反

具体案例：

- local => world：local在world坐标系中的位置，得到local=>world的变化矩阵
- world => view：「还是利用world矩阵的特殊性」view在world坐标系中的位置，得到view=>world的变化矩阵，但我们所要的是world=>view，所以需要求**矩阵的逆矩阵**

正交矩阵/仿射变换如何求逆矩阵：

正交矩阵的性质：

$$AA^T = I \tag{8}$$

注意：SRT中只有S是正交矩阵，所以要求仿射变换的逆矩阵，可以参考上述的三个方法

view矩阵求逆：

手撸位置变换不可避免的要遇到view矩阵求逆，如果没有现成的求逆函数（其实根本不需要），怎么办？

⚠️注意：view矩阵是一个很特殊的矩阵，它比model矩阵还要特殊，因为**view矩阵是只包含R（旋转）、T（平移）的矩阵，不包括S（缩放）**，所以它的逆矩阵很容易求得

‼️‼️【**笔试时犯错的地方**】『**世纪大坑**🕳』假设我知道view矩阵中旋转部分和平移部分，以及它们分别的逆矩阵，那么最终的变换矩阵就是将3x3的旋转部分转置，然后将1x3的平移部分取反吗？❌大错特错！！！！！！

坑产生原因：因为我们要求的是变换矩阵的逆，所以矩阵的乘法被交换顺序了，不再像原来那样组合旋转3x3部分和平移1x3部分就可以了，查看如下公式：👇【这里写错了，这里写成OpenGL规范了~】

$$\begin{aligned}(RT)^{-1} = T^{-1}R^{-1}\\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T_x & -T_y & -T_z & 1 \end{bmatrix} \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ -u\cdot T & -v\cdot T & -w\cdot T & 1 \end{bmatrix} \neq \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ -T_x & -T_y & -T_z & 1 \end{bmatrix}\end{aligned} \tag{9}$$

透视除法除以0怎么处理：

在图形api中，不会出现透视除法除以0的情况，因为**不在视锥体范围的点早就被裁剪掉了**「视锥体剔除可以在cpp端进行，自己判断物体是否与视锥体的boundingbox相交」，如果手撸位置变换需要判断，可以进行特殊处理，比如将0变成0.0001之类的

正交投影矩阵：

我们需要确定那部分被投影，所以需要六个参数left、right、top、bottom、near、far，这六个参数定义了一个盒子，盒子内的点会被映射到范围[-1,1]*[-1,1]*[0,1]

$$以x为例：(x - \frac{r+l}{2})/\frac{r-l}{2} \tag{10}$$

所以我们很容易构造矩阵，那还考虑到要透视除法，将w令为1即可

遇到的新问题与解决：

- model矩阵：
  - 分别推导S、R、T矩阵时，S、R、T矩阵初始化必须是单位矩阵，而不能是零矩阵

一些很简单的问题（不在这里赘述了，罗列在下方）：

- projection矩阵的推导：透视矩阵
- 屏幕映射：y上下颠倒

🤔**一些发现**：

- near=1，far=100，当距离为z=10时，归一化深度值已经超过0.9，所以说从10～100，归一化深度值的范围只有0.1给它们分配，所以z-fighting问题很严重

将行向量填入矩阵还是列向量?

从线性变换开始证明:

$$\tau(\mathbf{u}) = x\tau(\mathbf{i}) + y\tau(\mathbf{j}) + z\tau(\mathbf{k}) = [x \quad y \quad z]\begin{bmatrix} \leftarrow \tau(\mathbf{i}) \rightarrow \\ \leftarrow \tau(\mathbf{j}) \rightarrow \\ \leftarrow \tau(\mathbf{k}) \rightarrow \end{bmatrix} \tag{11}$$

所以可以看出，x只会与τ(i)的分量相乘，因为τ(i)存储的就是变换后的x轴坐标 => 所以不管是哪种API，我们只要保持这个规律即可