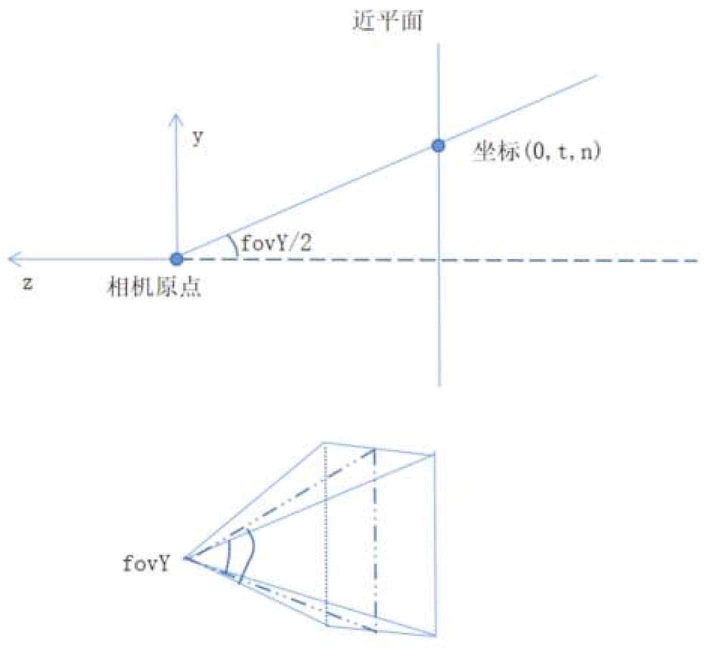


图形学 旋转与投影矩阵—2


随遇而安
不忘初心

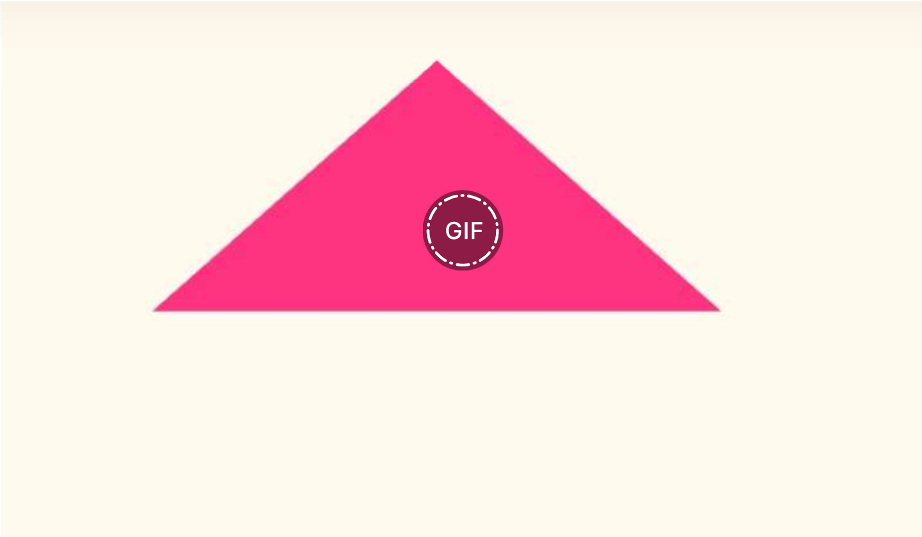
图形学 旋转与投影矩阵—2

> game101 第二次作业； webgl 实现

使用 THREEJS 作为基础框架，构建各类矩阵，自定义矩阵运算，最终完成

1. 正确构建模型矩阵
2. 正确构建透视投影矩阵
3. 看到变换后的三角形
4. 按 A 和 D 三角形能够进行旋转
5. 按 Q 和 E 三角形能够绕任意过原点的向量进行旋转

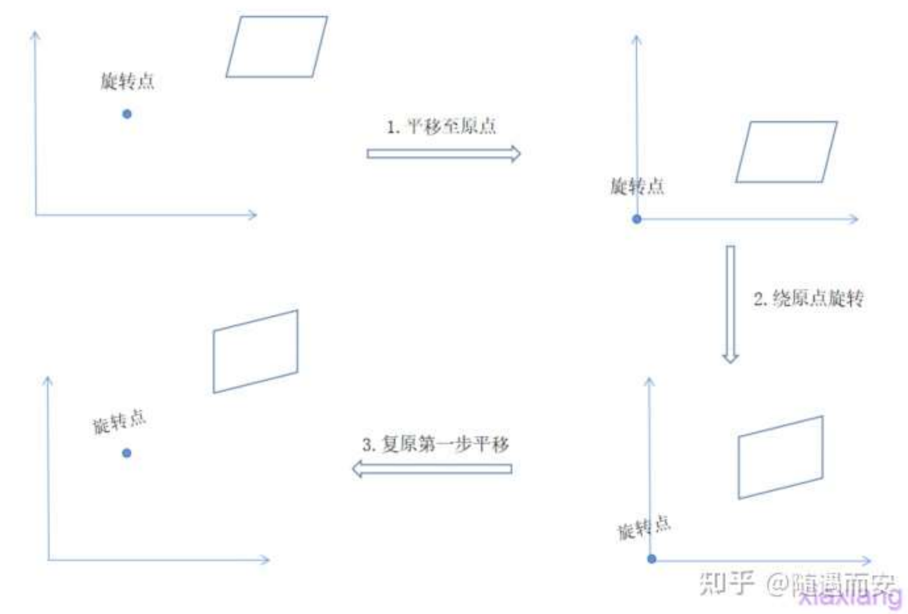
> 最终效果



前文简介

在旋转与投影矩阵第一篇中，以二维坐标系举例，详细解释了三种基础变换的矩阵形式，其中包括缩放变换，旋转变换和平移变换。平移变换比较特殊，由于 2*2 的矩阵无法表示平移变换，增加一个维度表示平移变换能获得很多好处，因此后续均采用 3*3 的矩阵表示基础变换。

补充：绕原点旋转的矩阵已经在前文推断出来，如果想绕指定点进行旋转，此时应该如何处理？这里转个弯，先将指定点和物体同时移动相同距离直至指定点到原点，再应用绕原点旋转的矩阵，最后再还原移动，恢复开始移动的距离。具体流程如图一所示。



图一：绕任意点旋转

三维变换

在三维中的变换和在二维中的变换是类似的，可以采用相同的推断，
转，无法表示平移变换，因此需要引入第四个维度，使用 4*4 的矩阵
的变换。

$$\begin{bmatrix} z' \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & S_z \end{bmatrix} \quad \begin{bmatrix} z \end{bmatrix}$$

表示在 x, y, z 上分别缩放 S_x, S_y, S_z 倍

刚刚也谈到要用四维坐标表示，四维坐标扩展很容易，扩展位除了（3， 3）位全是 0，（3， 3）位是 1

$$缩放矩阵 = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

平移变换比较简单，数值处于第四列

$$平移矩阵 = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

表示在 x, y, z 上分别移动 T_x, T_y, T_z 个单位长度

旋转矩阵稍微特殊一点，推导方式与二维一致

一个立方体，绕原点旋转 θ 角度，选中点进行代入计算，最终算出矩阵每个元素的值，三维旋转可以分成绕 X 轴旋转，绕 Y 轴旋转和绕 Z 轴旋转，可以想象一下绕这三种旋转最终得到的图形。经过代入计算可得

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

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

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

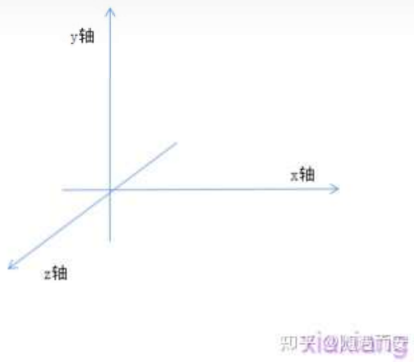
这个旋转矩阵和二维旋转是非常类似的矩阵

$$二维旋转矩阵 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Ry(θ) 稍有不同，Rx 和 Rz 相当于是在单位矩阵上固定相应轴的值，在进行二维旋转。Ry 的特殊性在于

$$\begin{gathered} \vec{x} \times \vec{y} = \vec{z} \\ \vec{y} \times \vec{z} = \vec{x} \\ 特殊处：\vec{x} \times \vec{z} = -\vec{y} \end{gathered}$$

依图所示加上右手定则，向量叉乘可得出，因此绕 y 轴的旋转矩阵会



描述了绕 x,y,z 轴的旋转矩阵，实际上还有一种，绕原点任意方向进行旋转，这个方程在程序中也会被用到，设向量为 n，旋转角度为 θ

$$R(n,\theta)=\cos(\theta)\times I+(1-\cos(\theta))\times n\times n^T+\sin(\theta)\times\begin{bmatrix}0 & -n_z & n_y\\ n_z & 0 & -n_x\\ -n_y & n_x & 0\end{bmatrix}$$

n_x,n_y,n_z 分别是向量 $\vec{n}$ 的x,y,z值

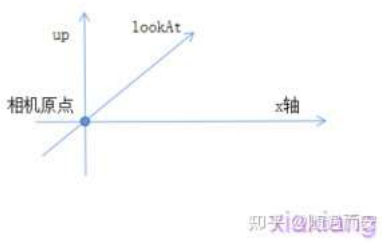
该式该需要转换为 4*4 适配整套计算，方法很简单，[3, 3] 位置为 1， 其余位置补充为0。这里提及的是绕过原点的 n 向量旋转，那想求过任意点的 n 向量旋转该怎么做呢？方法之前也提到过，先将任意点移动到原点，旋转后在纠正回去即可。

因为旋转矩阵运算不适合插值计算，因此需要四元数处理旋转，在计算时更加方便和高效。

视图矩阵

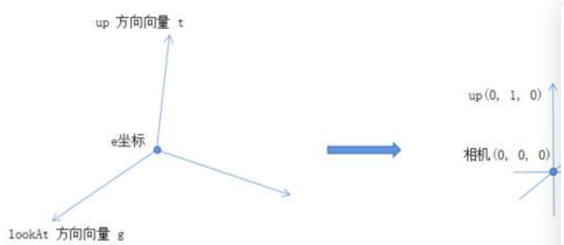
之前的各种三维变换，把他们组合在一起，就形成了**模型矩阵**，也就是说，模型矩阵是物体的三维变换的矩阵乘积。模型的位置基本确定，接下来得确定相机的位置。相机默认在原点处，up 向上方向是 y 轴正方向，lookAt 方向位 z 轴负方向，如果相机不在，那么就得矫正，美名曰规范化。

规范化分为两步，第一步，将相机移动到原点，第二步，将相机的 up 和 lookAt 方向规范



规范化的相机视角

假设现在相机移动，旋转后，如果复原呢？



分步骤计算

第一步：转换位置到原点。由于当前相机位置是 e，对应的坐标为(xe, ye, ze)，容易写出平移矩阵 Tview，如下所示

$$T_{view} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

第二步：转换 up 和 looAt 方向。向着复原过程求解不太容易，转换为求从标准状态转换为当前状态更好理解。也就是说，现在可以求规范化的相机如何旋转到目前的方向，然后对这个方向求逆矩阵。还是代入计算

$$R_{view}^{-1} \times \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_{-g} \\ y_{-g} \\ z_{-g} \\ 0 \end{bmatrix}$$

$lookAt$方向计算

$$R_{view}^{-1} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ z_t \\ 0 \end{bmatrix}$$

up方向计算

得到$lookAt$方向和up方向后，x轴方向为$lookAt \times up$

$$R_{view}^{-1} = \begin{bmatrix} x_{g\times t} & x_t & x_{-g} & 0 \\ y_{g\times t} & y_t & y_{-g} & 0 \\ z_{g\times t} & z_t & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = R^T$$

得到

$$R = \begin{bmatrix} x_{g\times t} & y_{g\times t} & z_{g\times t} & 0 \\ x_t & y_t & z_t & 0 \\ x_{-g} & y_{-g} & z_{-g} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

第三步，求得视图矩阵 Mview = Rview * Tview

规范立方体

给出一个立方体，用 top, bottom, left, right, near, far，六个面切割形成一个四棱柱，正交矩阵的作用是将这个四棱柱规范化，将中心置为原点处，长宽高分别为2，形成 [-1, -1, -1] 到 [1, 1, 1] 的规范立方体。

这个过程也是分为两步，第一步，将中心置于原点处，第二步，将图形压缩成规范立方体，由此可得正交矩阵，先设 top, bottom, left, right, near, far 值分别为 t, b, l, r, n, f

$$M_{ortho} = \text{旋转矩阵} * \text{平移矩阵}$$

$$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

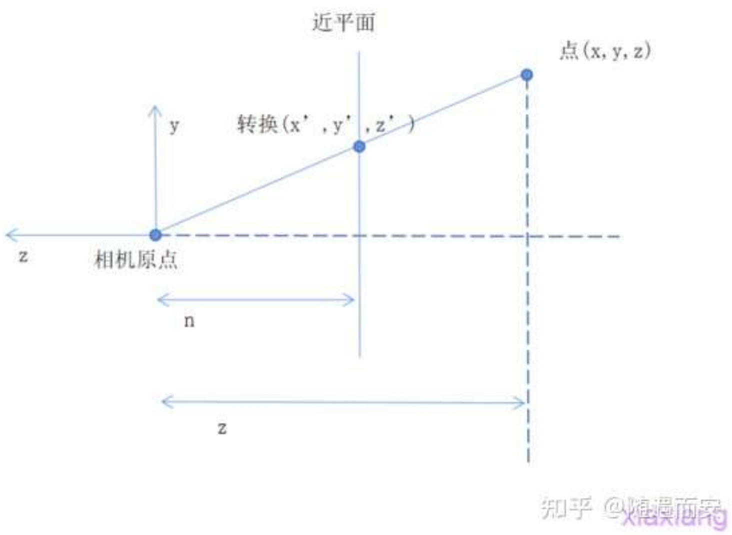
注意：先操作的矩阵在右边，后操作的矩阵在左边，这是因为元素坐
阵。这也代表元素最先相乘的是所有变换矩阵的右侧，案例见下列算

投影矩阵

投影相机看到的视角和人眼看到的视角可以保持一致，因此，在三维世界中采用的一般是投影相机，加入运算的便是投影矩阵，如果直接求解投影矩阵是比较麻烦的，可以将过程转换为，先把投影矩阵转换为正交矩阵再进行计算

$$M_{persep} = M_{ortho} \times M_{persp \to ortho}$$

因为正交矩阵上文已经求解出来了，现在只需要求解投影到正交的转换矩阵。现在有一个点的坐标是(x, y, z)，经过转换后投影再相机的近平面上，经过转换后的点的坐标为 (x', y', z')，转换过程如下图所示



从投影到正交的转换过程中，我们可以得到一些等式从而解出这个转换方程式，具体如下

$$y' = \frac{n}{z} \times y$$
$$x' = \frac{n}{z} \times x$$
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \frac{n}{z}x \\ \frac{n}{z}y \\ ? \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ ? \\ z \end{bmatrix}$$

这时，我们得到了转换后点的坐标 [x', y', z', 1] = [nx, ny, ?, z]，为啥可以同时乘 z ，前文中也提到，最后一个参数最后总会归一化成 1，在之前无论怎么变换，最后，x,y,z 都会同时 /w ，因此在计算时，可以 x, y, z, w 同时乘上一个数，意义不变。

已知转换前后点的坐标，再列方程式

$$M_{persp \to ortho} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ ? \\ z \end{bmatrix}$$
$$\Downarrow$$
$$M_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ ? & ? & ? & ? \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

注意转换过程中有两个性质

因此可以代入两个点的变换

$$
1: \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ n \\ 1 \end{bmatrix} = \begin{bmatrix} nx' \\ ny' \\ n^2 \\ n \end{bmatrix}
$$

$$
2: \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \\ f \\ 1 \end{bmatrix} = \begin{bmatrix} fx' \\ fy' \\ f^2 \\ f \end{bmatrix}
$$

可得

$$
\begin{bmatrix} ? & ? & ? & ? \end{bmatrix} \times \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} = n^2
$$

$$
\begin{bmatrix} ? & ? & ? & ? \end{bmatrix} \times \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = f^2
$$

因为得出的结果与x, y没有关系，因此可以推断出前两个问号为0，设第三和第四个问号为A，B

$$
\begin{bmatrix} 0 & 0 & A & B \end{bmatrix} \times \begin{bmatrix} x \\ y \\ n \\ 1 \end{bmatrix} = n^2
$$

$$
\begin{bmatrix} 0 & 0 & A & B \end{bmatrix} \times \begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} = f^2
$$

$$
\Downarrow
$$

$$
\begin{cases} A \times n + B = n^2 \\ A \times f + B = f^2 \end{cases} \rightarrow \begin{cases} A = n + f \\ B = -n \times f \end{cases}
$$

由此可得到投影转换矩阵的具体内容

$$
M_{persp \to ortho} = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}
$$

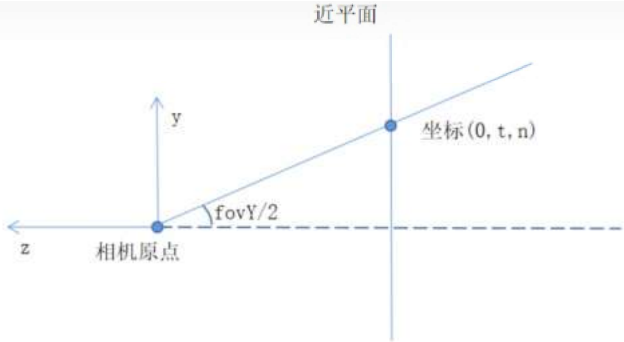
最终的投影矩阵的表达式为

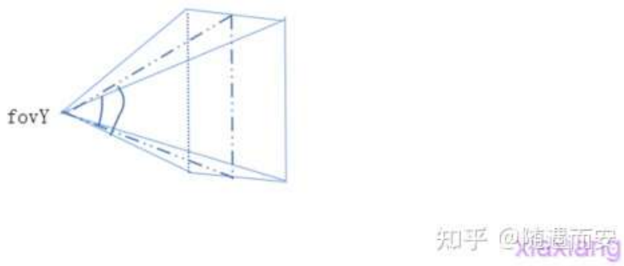
$$
M_{persep} = M_{ortho} \times M_{persp \to ortho}
$$

透视矩阵参数

正交矩阵需要的参数是 n, f, l, r, t, b,分别表示 near, far, left, right, to
这么多，但是很多参数都是可以推导的，这里添加两个概念，宽高比

宽高比比较好理解，字如其意，aspect = 宽/高；视角分为两种，一种
在 THREEJS 中，fov 代表的就是 fovY，表示如下图所示





可以写出下面两个式子

aspect = \frac r t = \frac {2r}{2t} = \frac 宽 高 \\ tan{\frac {fovY} 2} = \frac t {|n|}

由于相机是朝z轴负方向看，因此 n 和 f 是负数，这里需要加个绝对值。n 和 f 都是必须要知道的，已知 n 和 fovY，就能求出 t, t 和 f 互为相反数，因此可以得到 f, 知道 aspect 和 t，就能知道 r, r 和 l 互为相反数。

因此，知道了 fovY, aspect，n 和 f 就能求出所有需要的值

结论

在三维世界中，有三个重要的矩阵，模型矩阵，视图矩阵和投影矩阵，只有知道这三个矩阵才能将元素的元素坐标转换为规范立方体内的坐标

gl_Position = M_{persp} \times M_{view} \times M_{model} \times position

目前，每个矩阵均已经求解完毕，只要提供 position，就能知道最终坐标。

旋转与投影矩阵分为了三篇，第一篇描述了二维坐标下的矩阵变换，第二篇即本篇描述了三维坐标系下的基础概念，和相应的转换公式，基于这些转换公式便可以开始进行实验，第三篇以代码为主，主要讲实现文章开头的五个需求，构建一个三维世界出来

编辑于 2021-12-25 15:03

计算机图形学

还没有评论，发表第一个评论吧

推荐阅读

图形学：正交/透视投影矩阵的推导（多个思路）

Hier.... 发表于游戏引擎与...

斜交投影矩阵第一弹

本次针对一种较为特殊的投影矩阵：斜投影矩阵进行介绍，其与正交投影阵的主要区别在于其不再满足对称性，但仍保留幂等的特性，以下引出此类矩阵的定义并对部分性质进行证明。（本次更新内容...

阿里多多的... 发表于线性模型与...

投影算法（一）：正（Orthographic Pr

英文原文链接：投影矩导投影矩阵主要分为两影、透视投影，这篇文的是正交投影矩阵的推再介绍透视投影原理构矩阵相对于透视投影来

太上无名氏