

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于 $4 \times 4$ 矩阵，可能你会想，为什么不是 $3 \times 3$ 呢？这是因为四个关键运算中有一个无法用 $3 \times 3$ 矩阵来完成，其他三个运算为了统一也就都采用 $4 \times 4$ 矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用 $3 \times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b}) \neq \mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以， $3 \times 3$ 矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了 $4 \times 4$ 矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

### 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ ，这也就意味着三维空间的每个点都加上了点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的 $4 \times 4$ 矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{llll} 0 & 0 & 0 & 1 \end{array}\right] T = \left[\begin{array}{llll} x_0 & y_0 & z_0 & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v}+\mathbf{v}_0$ 的最终结果： $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T = \left[\begin{array}{llll} x+x_0 & y+y_0 & z+z_0 & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

### 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2 \times 2$ 矩阵来表达缩放，在三维立体中则是 $3 \times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3 \times 3$ 矩阵变成了 $4 \times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0
\end{array}\right]
$$
```

$$\begin{bmatrix} 0 & 0 & 1 \\ \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0.9 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

$$T_{00} \cdot R_{45} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}$$

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

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

2. 围绕 $y$ 轴方向旋转：

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

3. 围绕 $z$ 轴方向旋转：

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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{n} \mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(I - n n^T) n = n - n(n^T n) = 0$$

$$(I - n n^T) v = v - n(n^T v) = v$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

$$P = \begin{bmatrix} I - n n^T & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面， $v_0$ 是这个平面上的一个点，现在要把 $v_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $v_0$ 平移到原点；
2. 沿着 $n$ 方向投影；
3. 再平移回 $v_0$ 。

整个过程通过数学公式来表达就是：

$$T_{-v_0} P T_{+v_0} = \begin{bmatrix} I - n n^T & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -v_0 \\ 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

# 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $v_{\{0\}}$ ，也就是点 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $f(a+b) \neq f(a)+f(b)$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $v_{\{0\}}$ 平移整个三维空间，把原点平移到了 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ ，这也就意味着三维空间的每个点都加上了点 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ 。使用齐次坐标，把整个空间平移了 $v_{\{0\}}$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{lllll} 0 & 0 & 0 & 0 & 1 \end{array}\right] T=\left[\begin{array}{llll} x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_{\{0\}}$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_{\{0\}}$ 的最终结果： $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T=\left[\begin{array}{llll} x+x_{\{0\}} & y+y_{\{0\}} & z+z_{\{0\}} & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 0.9
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

```
$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$
```

```
$$
R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```
$$
```





而投影到平面的投影矩阵是： $I - \mathbf{n}\mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(I - \mathbf{n}\mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(I - \mathbf{n}\mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

$$P = \begin{bmatrix} I - \mathbf{n}\mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面， $\mathbf{v}_0$ 是这个平面上的一个点，现在要把 $\mathbf{v}_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $S(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $\mathbf{v}_0$ 平移到原点；
2. 沿着 $\mathbf{n}$ 方向投影；
3. 再平移回 $\mathbf{v}_0$ 。

整个过程通过数学公式来表达就是：

$$T_{-\mathbf{v}_0} P T_{+\mathbf{v}_0} = \begin{bmatrix} I - \mathbf{n}\mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结



今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $v_0$ ，也就是点 $(x_0,y_0,z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $f(a+b)$ 不等于 $f(a)+f(b)$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $v_0$ 平移整个三维空间，把原点平移到了 $(x_0,y_0,z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0,y_0,z_0)$ 。使用齐次坐标，把整个空间平移了 $v_0$ 的4×4矩阵 $T$ 如下所示。

$$T=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} T=\begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果： $\begin{bmatrix} x & y & z & 1 \end{bmatrix} T=\begin{bmatrix} x+x_0 & y+y_0 & z+z_0 & 1 \end{bmatrix}$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从2×2就变成了3×3矩阵 $R$ 。

```
$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$

$$
R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；

2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```


$$T_{\theta} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix}$$


```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

```


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


```

2. 围绕 $y$ 轴方向旋转：

```


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


```

3. 围绕 $z$ 轴方向旋转：

```


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


```

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{n} \mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面， $\mathbf{v}_0$ 是这个平面上的一个点，现在要把 $\mathbf{v}_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $\mathbf{v}_0$ 平移到原点；
2. 沿着 $\mathbf{n}$ 方向投影；
3. 再平移回 $\mathbf{v}_0$ 。

整个过程通过数学公式来表达就是：

$$\mathbf{T}_{-\mathbf{v}_0} \mathbf{P} \mathbf{T}_{+\mathbf{v}_0} = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{v}_0 \\ \mathbf{0} & 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b})$ 不等于 $\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ ，这也就意味着三维空间的每个点都加上了点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果：
$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} T = \begin{bmatrix} x+x_0 & y+y_0 & z+z_0 & 1 \end{bmatrix}$$

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $vTSS$ ，如果我们要先缩放再平移，那应该这样乘： $vSTS$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从2×2就变成了3×3矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$
$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \end{bmatrix}$$

```
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```
$$
v T_{00} R T_{45} = \left[ \begin{array}{ccc}
x & y & 1 \\
\end{array} \right] \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
-4 & -5 & 1
\end{array} \right] \left[ \begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array} \right] \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
4 & 5 & 1
\end{array} \right]
$$
```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

```
$$
R_x = \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```

2. 围绕 $y$ 轴方向旋转：

```
$$
R_y = \left[ \begin{array}{ccc}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```

3. 围绕 $z$ 轴方向旋转：

```
$$
R_z = \left[ \begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```



\$\$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

\$\$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

\$\$

\$\$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

\$\$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

\$\$

$$\mathbf{P} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 1 \end{matrix}$$

$$\end{bmatrix}$$

\$\$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

\$\$

$$\mathbf{T}_{-\mathbf{v}_0} \mathbf{P} \mathbf{T}_{\mathbf{v}_0} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} 1 & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} -\mathbf{v}_0 & 1 \end{matrix}$$

$$\end{bmatrix} \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} 0 & 1 \end{matrix}$$

$$\end{bmatrix} \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} 1 & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} \mathbf{v}_0 & 1 \end{matrix}$$

$$\end{bmatrix}$$

\$\$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换

到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于 $4 \times 4$ 矩阵，可能你会想，为什么不是 $3 \times 3$ 呢？这是因为四个关键运算中有一个无法用 $3 \times 3$ 矩阵来完成，其他三个运算为了统一也就都采用 $4 \times 4$ 矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用 $3 \times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b})$ 不等于 $\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以， $3 \times 3$ 矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了 $4 \times 4$ 矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的 $4 \times 4$ 矩阵 $T$ 如下所示。

\$\$  
T=\left[\begin{array}{llll}\end{array}\right]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果：

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} T = \begin{bmatrix} x+x_0 & y+y_0 & z+z_0 & 1 \end{bmatrix}$$

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2 \times 2$ 矩阵来表达缩放，在三维立体中则是 $3 \times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3 \times 3$ 矩阵变成了 $4 \times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \end{bmatrix}$$

```

\sin \theta & \cos \theta \\
\end{array} \right] \\
\end{array} \\
\\
\\
R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\\
\\

```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点  $(4,5)$ ，让平面旋转  $\theta$  角度的话：

1. 首先，要把  $(4,5)$  平移到  $(0,0)$ ；
2. 接着，旋转  $\theta$  角度；
3. 最后，再把  $(0,0)$  平移回  $(4,5)$ 。

整个过程通过数学公式来表达就是：

```

\\
v_{T_{00}} R_{T_{45}} = \begin{bmatrix} x & y & 1 \\ \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}
\\
\\

```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕  $\lambda=1$  的特征向量的一条线翻转。

现在，我们来看看分别围绕  $x$ 、 $y$  和  $z$  轴方向旋转的矩阵  $R$  有什么不同？

1. 围绕  $x$  轴方向旋转：

```

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

```

2. 围绕  $y$  轴方向旋转：

```

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

```

3. 围绕  $z$  轴方向旋转：

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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n} (\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n} (\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下\$4 \times 4\$的投影矩阵：

$$P = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 0 & 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

$$T_{-\mathbf{v}_0} P T_{+\mathbf{v}_0} = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}_0 & 1 \end{bmatrix}$$

# 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b}) \neq \mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。



## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：  
 $\left[\begin{array}{llll} 0 & 0 & 0 & 1 \end{array}\right] T = \left[\begin{array}{llll} x_0 & y_0 & z_0 & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x, y, z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $(x, y, z, 1)$ ，然后， $(x, y, z, 1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v} + \mathbf{v}_0$ 的最终结果：  
 $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T = \left[\begin{array}{llll} x+x_0 & y+y_0 & z+z_0 & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $\mathbf{v}TS$ ，如果我们要先缩放再平移，那应该这样乘： $\mathbf{v}ST$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转



二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2\times 2$ 就变成了 $3\times 3$ 矩阵 $R$ 。

```


$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$


$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```


$$T_{00} R_{45} = \begin{bmatrix} x & y & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}$$


```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

```


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


```

2.围绕 $y$ 轴方向旋转：

```


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


```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \\ \end{pmatrix}$$

3.围绕\$z\$轴方向旋转：

$$R_z = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下\$4 \times 4\$的投影矩阵：

$$P = \begin{pmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

$$T_{-\mathbf{v}_0} P T_{+\mathbf{v}_0} = \begin{pmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$$

$$\begin{array}{l} I - n \wedge \{T\} \ \& \ 0 \\\ 0 \ \& \ 1 \\\ \end{array} \right] \left[ \begin{array}{l} \{II\} \\\ I \ \& \ 0 \\\ v_{\{0\}} \ \& \ 1 \\\ \end{array} \right]$$

# 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

# 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于 $4 \times 4$ 矩阵，可能你会想，为什么不是 $3 \times 3$ 呢？这是因为四个关键运算中有一个无法用 $3 \times 3$ 矩阵来完成，其他三个运算为了统一也就都采用 $4 \times 4$ 矩阵了，这四个关键运算是：

- 平移;
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 $3 \times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{s}_{\{0\}}$ ，也就是点 $\mathbf{s}(\mathbf{x}_{\{0\}}, \mathbf{y}_{\{0\}}, \mathbf{z}_{\{0\}})$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点

之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $f(a+b) \neq f(a)+f(b)$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $v_{\{0\}}$ 平移整个三维空间，把原点平移到了 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ ，这也就意味着三维空间的每个点都加上了点 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ 。使用齐次坐标，把整个空间平移了 $v_{\{0\}}$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{lllll} 0 & 0 & 0 & 0 & 1 \end{array}\right] T=\left[\begin{array}{llll} x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_{\{0\}}$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_{\{0\}}$ 的最终结果： $\left[\begin{array}{lll} x & y & z \end{array}\right] T=\left[\begin{array}{llll} x+x_{\{0\}} & y+y_{\{0\}} & z+z_{\{0\}} & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $VTS$ ，如果我们要先缩放再平移，那应该这样乘： $STS$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

```
$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$

R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```
$$
vT_{00}RT_{45}=\left[\begin{array}{l}
x & y & 1
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
-4 & -5 & 1
\end{array}\right]\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
4 & 5 & 1
\end{array}\right]
$$
```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

```
$$
R_x=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

2.围绕 $y$ 轴方向旋转：

```
$$
R_y=\left[\begin{array}{ccc}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta
\end{array}\right]
$$
```

3.围绕 $z$ 轴方向旋转：

```
$$
R_z=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{n} \mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

```
$$
(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}
$$
```

```
$$
(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}
$$
```

接下来，我们在齐次坐标中来看一下 $4 \times 4$ 的投影矩阵：

```
$$
P = \left[\begin{array}{ccc|c}
& & & 0 \\
& & & 0 \\
& \mathbf{I} - \mathbf{n} \mathbf{n}^T & & 0 \\
& & & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

假设现在有一个不过原点的平面， $\mathbf{v}_0$ 是这个平面上的一个点，现在要把 $\mathbf{v}_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $\mathbf{v}_0$ 平移到原点；
2. 沿着 $\mathbf{n}$ 方向投影；
3. 再平移回 $\mathbf{v}_0$ 。

整个过程通过数学公式来表达就是：



```

$$
T_{-v_{0}} P T_{+v_{0}} = \left[ \begin{array}{cc}
I & 0 \\
-v_{0} & 1
\end{array} \right] \left[ \begin{array}{cc}
I - n n^T & 0 \\
0 & 1
\end{array} \right] \left[ \begin{array}{l}
I & 0 \\
v_{0} & 1
\end{array} \right]
$$

```

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；



- 投影。

平移就是那个无法用 $3\times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b})$ 不等于 $\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以， $3\times 3$ 矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了 $4\times 4$ 矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的 $4\times 4$ 矩阵 $T$ 如下所示。

```


$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$


```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：  

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v}+\mathbf{v}_0$ 的最终结果：  

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2\times 2$ 矩阵来表达缩放，在三维立体中则是 $3\times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3\times 3$ 矩阵变成了 $4\times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

```


$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


```

三维立体中图片放大90%就是：

```


$$S = \begin{bmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$


```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```


$$S = \begin{bmatrix} 0.75 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


```

$$S=\begin{bmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S\mathbf{T}$ ，如果我们要先缩放再平移，那应该这样乘： $S\mathbf{S}\mathbf{T}$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2\times 2$ 就变成了 $3\times 3$ 矩阵 $R$ 。

$$Q=\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$R=\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

$$\mathbf{T}_{\{00\}} \mathbf{R}_\theta \mathbf{T}_{\{45\}}=\begin{bmatrix} x & y & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}$$

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

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

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

2. 围绕\$y\$轴方向旋转：

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

3. 围绕\$z\$轴方向旋转：

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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下\$4 \times 4\$的投影矩阵：

$$P = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & 0 \\ 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，

和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $v_0$ 平移到原点；
2. 沿着 $n$ 方向投影；
3. 再平移回 $v_0$ 。

整个过程通过数学公式来表达就是：

```


$$T_{-v_0} P T_{+v_0} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$


```

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维

世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b}) \neq \mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：  
 $\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] T = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} x_0 \\ y_0 \\ z_0 \\ 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v}+\mathbf{v}_0$ 的最终结果：  
 $\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} x \\ y \\ z \\ 1 \end{array}\right] T = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} x+x_0 \\ y+y_0 \\ z+z_0 \\ 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0
\end{array}\right]
$$
```

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ \end{pmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{pmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

$$R = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

$$T_{00} \cdot R_{45} = \begin{pmatrix} x & y & 1 \\ \end{pmatrix} \cdot \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{pmatrix}$$



说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

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

2.围绕 $y$ 轴方向旋转：

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

3.围绕 $z$ 轴方向旋转：

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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{n} \mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下 $4 \times 4$ 的投影矩阵：

$$P = \begin{bmatrix} \dots \end{bmatrix}$$





而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $\mathbf{(x_0, y_0, z_0)}$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f(a+b)} \neq \mathbf{f(a)} + \mathbf{f(b)}$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $\mathbf{(0,0,0,1)}$ ，那就能解决平移的问题了。点 $\mathbf{(x,y,z)}$ 的齐次坐标就是 $\mathbf{(x,y,z,1)}$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v_0}$ 平移整个三维空间，把原点平移到了 $\mathbf{(x_0, y_0, z_0)}$ ，这也就意味着三维空间的每个点都加上了点 $\mathbf{(x_0, y_0, z_0)}$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v_0}$ 的4×4矩阵 $\mathbf{T}$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\mathbf{\left[\begin{array}{lllll} 0 & 0 & 0 & 1 \end{array}\right] T = \left[\begin{array}{llll} x_0 & y_0 & z_0 & 1 \end{array}\right]}$ 。

平移的整个过程是这样的：假设要把原来的某个点 $\mathbf{(x,y,z)}$ 平移 $\mathbf{v_0}$ ，我们需要切换到齐次坐标 $\mathbf{(x,y,z,1)}$ ，然后， $\mathbf{(x,y,z,1)}$ 再乘 $\mathbf{T}$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v+v_0}$ 的最终结果： $\mathbf{\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T = \left[\begin{array}{llll} x+x_0 & y+y_0 & z+z_0 & 1 \end{array}\right]}$ 。

这里你需要注意：一个行向量乘 $\mathbf{T}$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```

$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$

```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```

$$
S=\left[\begin{array}{ccc}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$

```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $SvT$ ，如果我们要先缩放再平移，那应该这样乘： $SvST$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

```

$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$

$$
R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$

```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```

$$
v T_{00} R T_{45}=\left[\begin{array}{ccc}
x & y & 1
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
-4 & -5 & 1
\end{array}\right]\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0
\end{array}\right]
$$

```

$$\begin{array}{ccc} 0 & 1 & 0 \\ 4 & 5 & 1 \end{array}$$

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{pmatrix}$$

2.围绕 $y$ 轴方向旋转：

$$R_y = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix}$$

3.围绕 $z$ 轴方向旋转：

$$R_z = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{n} \mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

```
$$
P=\left[\begin{array}{lll}
&&0\\
&I-nr^T&0\\
&&0\\
0&0&0&1
\end{array}\right]
$$
```

假设现在有一个不过原点的平面， $v_0$ 是这个平面上的一个点，现在要把 $v_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $v_0$ 平移到原点；
2. 沿着 $n$ 方向投影；
3. 再平移回 $v_0$ 。

整个过程通过数学公式来表达就是：

```
$$
T_{-v_0} P T_{+v_0}=\left[\begin{array}{cc}
I&0\\
-v_0&1
\end{array}\right]\left[\begin{array}{cc}
I-nr^T&0\\
0&1
\end{array}\right]\left[\begin{array}{ll}
I&0\\
v_0&1
\end{array}\right]
$$
```

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于 $4 \times 4$ 矩阵，可能你会想，为什么不是 $3 \times 3$ 呢？这是因为四个关键运算中有一个无法用 $3 \times 3$ 矩阵来完成，其他三个运算为了统一也就都采用 $4 \times 4$ 矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用 $3 \times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b}) \neq \mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以， $3 \times 3$ 矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $\mathbf{x}(x,y,z)$ 的齐次坐标就是 $\mathbf{x}(x,y,z,1)$ ，这就变成了 $4 \times 4$ 矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

### 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ ，这也就意味着三维空间的每个点都加上了点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的 $4 \times 4$ 矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1&0&0&0\\
0&1&0&0\\
0&0&1&0\\
x_0&y_0&z_0&1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{llll} \text{|||||} & 0 & 0 & 0 \end{array}\right] T = \left[\begin{array}{llll} \text{|||} & x_0 & y_0 & z_0 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $\mathbf{x}(x,y,z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $\mathbf{x}(x,y,z,1)$ ，然后， $\mathbf{x}(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v}+\mathbf{v}_0$ 的最终结果： $\left[\begin{array}{llll} \text{|||} & x & y & z \end{array}\right] T = \left[\begin{array}{llll} \text{|||} & x+x_0 & y+y_0 & z+z_0 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

### 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2 \times 2$ 矩阵来表达缩放，在三维立体中则是 $3 \times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3 \times 3$ 矩阵变成了 $4 \times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9&0&0\\
0&0.9&0
\end{array}\right]
$$
```



$$\begin{bmatrix} 0 & 0 & 1 \\ \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0.9 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 \\ 0 & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

$$T_{00} R_{45} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}$$

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

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

2. 围绕 $y$ 轴方向旋转：

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

3. 围绕 $z$ 轴方向旋转：

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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $\mathbf{n}$ ，那么平面中的向量 $\mathbf{v}$ ，满足这个等式： $\mathbf{n}^T \mathbf{v} = 0$ 。

而投影到平面的投影矩阵是： $\mathbf{I} - \mathbf{nn}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(I - n n^T) n = n - n(n^T n) = 0$$

$$(I - n n^T) v = v - n(n^T v) = v$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

$$P = \begin{bmatrix} I - n n^T & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面， $v_0$ 是这个平面上的一个点，现在要把 $v_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $v_0$ 平移到原点；
2. 沿着 $n$ 方向投影；
3. 再平移回 $v_0$ 。

整个过程通过数学公式来表达就是：

$$T_{-v_0} P T_{+v_0} = \begin{bmatrix} I - n n^T & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

# 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $v_0$ ，也就是点 $(x_0,y_0,z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $f(a+b) \neq f(a)+f(b)$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $v_0$ 平移整个三维空间，把原点平移到了 $(x_0,y_0,z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0,y_0,z_0)$ 。使用齐次坐标，把整个空间平移了 $v_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{lllll} 0 & 0 & 0 & 0 & 1 \end{array}\right] T=\left[\begin{array}{llll} x_0 & y_0 & z_0 & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果： $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T=\left[\begin{array}{llll} x+x_0 & y+y_0 & z+z_0 & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 0.9
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

```
$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$
```

```
$$
R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```
$$
```

$$\begin{array}{l}
 \text{\texttt{v T_{00} R T_{45}=\left[\begin{array}{l} \{III\} \\ x \& y \& 1 \\ \end{array}\right]\left[\begin{array}{l} \{ccc\} \\ 1 \& 0 \& 0 \\ 0 \& 1 \& 0 \\ -4 \& -5 \& 1 \\ \end{array}\right]\left[\begin{array}{l} \{ccc\} \\ \cos \theta \& -\sin \theta \& 0 \\ \sin \theta \& \cos \theta \& 0 \\ 0 \& 0 \& 1 \\ \end{array}\right]\left[\begin{array}{l} \{ccc\} \\ 1 \& 0 \& 0 \\ 0 \& 1 \& 0 \\ 4 \& 5 \& 1 \\ \end{array}\right]} \\
 \text{\texttt{\end{array}\right]} \\
 \text{\texttt{\$ \$}}
 \end{array}$$

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

$$\begin{array}{l}
 \text{\texttt{R_{x}=\left[\begin{array}{l} \{cccc\} \\ 1 \& 0 \& 0 \& 0 \\ 0 \& \cos \theta \& -\sin \theta \& 0 \\ 0 \& \sin \theta \& \cos \theta \& 0 \\ 0 \& 0 \& 0 \& 1 \\ \end{array}\right]}} \\
 \text{\texttt{\end{array}\right]} \\
 \text{\texttt{\$ \$}}
 \end{array}$$

2.围绕 $y$ 轴方向旋转：

$$\begin{array}{l}
 \text{\texttt{R_{y}=\left[\begin{array}{l} \{cccc\} \\ \cos \theta \& 0 \& \sin \theta \& 0 \\ 0 \& 1 \& 0 \& 0 \\ -\sin \theta \& 0 \& \cos \theta \& 0 \\ 0 \& 0 \& 0 \& 1 \\ \end{array}\right]}} \\
 \text{\texttt{\end{array}\right]} \\
 \text{\texttt{\$ \$}}
 \end{array}$$

3.围绕 $z$ 轴方向旋转：

$$\begin{array}{l}
 \text{\texttt{R_{z}=\left[\begin{array}{l} \{cccc\} \\ \cos \theta \& -\sin \theta \& 0 \& 0 \\ \sin \theta \& \cos \theta \& 0 \& 0 \\ 0 \& 0 \& 1 \& 0 \\ 0 \& 0 \& 0 \& 1 \\ \end{array}\right]}} \\
 \text{\texttt{\end{array}\right]} \\
 \text{\texttt{\$ \$}}
 \end{array}$$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是 $n$ ，那么平面中的向量 $v$ ，满足这个等式： $n^T v=0$ 。



而投影到平面的投影矩阵是： $I - \mathbf{n}\mathbf{n}^T$ 。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量 $\mathbf{n}$ 投影后成为了0向量，而平面向量 $\mathbf{v}$ 投影后还是其自身。

$$(I - \mathbf{n}\mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(I - \mathbf{n}\mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

$$P = \begin{bmatrix} I - \mathbf{n}\mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}$$

假设现在有一个不过原点的平面， $\mathbf{v}_0$ 是这个平面上的一个点，现在要把 $\mathbf{v}_0$ 投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度经历的三个步骤类似：

1. 把 $\mathbf{v}_0$ 平移到原点；
2. 沿着 $\mathbf{n}$ 方向投影；
3. 再平移回 $\mathbf{v}_0$ 。

整个过程通过数学公式来表达就是：

$$T_{-\mathbf{v}_0} P T_{+\mathbf{v}_0} = \begin{bmatrix} I - \mathbf{n}\mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{v}_0 & 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $v_{\{0\}}$ ，也就是点 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $f(a+b)$ 不等于 $f(a)+f(b)$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $v_{\{0\}}$ 平移整个三维空间，把原点平移到了 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ ，这也就意味着三维空间的每个点都加上了点 $(x_{\{0\}},y_{\{0\}},z_{\{0\}})$ 。使用齐次坐标，把整个空间平移了 $v_{\{0\}}$ 的4×4矩阵 $T$ 如下所示。

$$T=\left[\begin{array}{llll} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1 \end{array}\right]$$

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如： $\left[\begin{array}{lllll} 0 & 0 & 0 & 1 \end{array}\right] T=\left[\begin{array}{llll} x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_{\{0\}}$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_{\{0\}}$ 的最终结果： $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T=\left[\begin{array}{llll} x+x_{\{0\}} & y+y_{\{0\}} & z+z_{\{0\}} & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘T的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $S_y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $S_x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从2×2就变成了3×3矩阵 $R$ 。

```
$$
Q=\left[\begin{array}{cc}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{array}\right]
$$

$$
R=\left[\begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；

2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```


$$T_{45} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix}$$


```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

```


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


```

2. 围绕 $y$ 轴方向旋转：

```


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


```

3. 围绕 $z$ 轴方向旋转：

```


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


```

你看出来哪里不同了吗？其实主要就是1的位置不同，以及 $y$ 轴方向旋转的 $\sin$ 互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？



《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b})$ 不等于 $\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0)$ ，这也就意味着三维空间的每个点都加上了点 $\mathbf{x}_0, \mathbf{y}_0, \mathbf{z}_0$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：



$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果：
$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} T = \begin{bmatrix} x+x_0 & y+y_0 & z+z_0 & 1 \end{bmatrix}$$

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2 \times 2$ 矩阵来表达缩放，在三维立体中则是 $3 \times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3 \times 3$ 矩阵变成了 $4 \times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $vTSS$ ，如果我们要先缩放再平移，那应该这样乘： $vSTS$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$
$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \end{bmatrix}$$

```
0 & 0 & 1
\end{array}\right]
$$
```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```
$$
v T_{00} R T_{45} = \left[ \begin{array}{ccc}
x & y & 1 \\
\end{array} \right] \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
-4 & -5 & 1
\end{array} \right] \left[ \begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{array} \right] \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
4 & 5 & 1
\end{array} \right]
$$
```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1. 围绕 $x$ 轴方向旋转：

```
$$
R_x = \left[ \begin{array}{ccc}
1 & 0 & 0 \\
0 & \cos \theta & -\sin \theta \\
0 & \sin \theta & \cos \theta \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```

2. 围绕 $y$ 轴方向旋转：

```
$$
R_y = \left[ \begin{array}{ccc}
\cos \theta & 0 & \sin \theta \\
0 & 1 & 0 \\
-\sin \theta & 0 & \cos \theta \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```

3. 围绕 $z$ 轴方向旋转：

```
$$
R_z = \left[ \begin{array}{ccc}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{array} \right]
$$
```

\$\$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

\$\$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

\$\$

\$\$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

\$\$

接下来，我们在齐次坐标中来看一下4×4的投影矩阵：

\$\$

$$\mathbf{P} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} & & & 0 \\ & & & \\ & & & \\ & & & \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 1 \end{matrix}$$

$$\end{bmatrix}$$

\$\$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

\$\$

$$\mathbf{T}_{-\mathbf{v}_0} \mathbf{P} \mathbf{T}_{\mathbf{v}_0} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} 1 & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} -\mathbf{v}_0 & 1 \end{matrix}$$

$$\end{bmatrix} \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} 0 & 1 \end{matrix}$$

$$\end{bmatrix} \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$$

$$\begin{matrix} 1 & 0 \\ & \\ & \\ & \end{matrix}$$

$$\begin{matrix} \mathbf{v}_0 & 1 \end{matrix}$$

$$\end{bmatrix}$$

\$\$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换

到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于 $4 \times 4$ 矩阵，可能你会想，为什么不是 $3 \times 3$ 呢？这是因为四个关键运算中有一个无法用 $3 \times 3$ 矩阵来完成，其他三个运算为了统一也就都采用 $4 \times 4$ 矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用 $3 \times 3$ 矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b})$ 不等于 $\mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以， $3 \times 3$ 矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了 $4 \times 4$ 矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的 $4 \times 4$ 矩阵 $T$ 如下所示。

\$\$  
T=\left[\begin{array}{llll}\end{array}\right]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

平移的整个过程是这样的：假设要把原来的某个点 $(x,y,z)$ 平移 $v_0$ ，我们需要切换到齐次坐标 $(x,y,z,1)$ ，然后， $(x,y,z,1)$ 再乘 $T$ ，就能得到每个原来的向量 $v$ 平移到 $v+v_0$ 的最终结果：

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} T = \begin{bmatrix} x+x_0 & y+y_0 & z+z_0 & 1 \end{bmatrix}$$

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用 $2 \times 2$ 矩阵来表达缩放，在三维立体中则是 $3 \times 3$ 矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说， $3 \times 3$ 矩阵变成了 $4 \times 4$ 矩阵。

比如，二维平面中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

三维立体中图片放大90%就是：

$$S = \begin{bmatrix} 0.9 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0.9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

$$S = \begin{bmatrix} \frac{3}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $S \cdot T$ ，如果我们要先缩放再平移，那应该这样乘： $S \cdot T$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2 \times 2$ 就变成了 $3 \times 3$ 矩阵 $R$ 。

$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \end{bmatrix}$$

```

\sin \theta & \cos \theta \\
\end{array} \right] \\
\end{array} \\
\\
\\
R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\\
\\

```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点  $(4,5)$ ，让平面旋转  $\theta$  角度的话：

1. 首先，要把  $(4,5)$  平移到  $(0,0)$ ；
2. 接着，旋转  $\theta$  角度；
3. 最后，再把  $(0,0)$  平移回  $(4,5)$ 。

整个过程通过数学公式来表达就是：

```

\\
v_{T_{00}} R_{T_{45}} = \begin{bmatrix} x & y & 1 \\ \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}
\\
\\

```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕  $\lambda=1$  的特征向量的一条线翻转。

现在，我们来看看分别围绕  $x$ 、 $y$  和  $z$  轴方向旋转的矩阵  $R$  有什么不同？

1. 围绕  $x$  轴方向旋转：

```

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

```

2. 围绕  $y$  轴方向旋转：

```

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

```

3. 围绕  $z$  轴方向旋转：



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

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n} (\mathbf{n}^T \mathbf{n}) = 0$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n} (\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下\$4 \times 4\$的投影矩阵：

$$P = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

$$T_{-\mathbf{v}_0} P T_{+\mathbf{v}_0} = \begin{bmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{v}_0 \\ 0 & 1 \end{bmatrix}$$

# 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。

你好，我是朱维刚。欢迎你继续跟我学习线性代数，今天我要讲的内容是“如何通过矩阵转换让3D图形显示到二维屏幕上”。

在第八篇的[线性映射](#)中，我从二维直角坐标系的角度，讲解了线性映射和变换矩阵。其中，我特别讲到了，二维平面图形图像处理中的线性变换，比如物体的拉伸和旋转。在第九篇的[仿射空间](#)中，更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样，我会从更实践的角度，让你了解到二维平面和三维空间的变换，以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没，接下来我们一起来看下矩阵到底是怎么做到的。

## 三维空间变换

我们都知道，计算机图形图像处理的是图片，且计算机屏幕是二维的。那你有没有想过，我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢？这个就要涉及到三维到二维的投影技术了，这类技术都离不开矩阵，而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵，可能你会想，为什么不是3×3呢？这是因为四个关键运算中有一个无法用3×3矩阵来完成，其他三个运算为了统一也就都采用4×4矩阵了，这四个关键运算是：

- 平移；
- 缩放；
- 旋转；
- 投影。

平移就是那个无法用3×3矩阵来完成的特殊运算，也是看起来最简单的运算，只是每个点都加上向量 $\mathbf{v}_0$ ，也就是点 $(x_0, y_0, z_0)$ 。

但是，你别被这个假象欺骗了，平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线，以及平移后各点与原点之间的连线就知道了。或者，你也可以从公式的角度理解，就是 $\mathbf{f}(\mathbf{a}+\mathbf{b}) \neq \mathbf{f}(\mathbf{a})+\mathbf{f}(\mathbf{b})$ 。而为了表示平移，以及现实世界的描述，就需要使用第九篇中说的[仿射空间](#)。所以，3×3矩阵是无法平移原点的。

但是，如果我们把原点坐标变成 $(0,0,0,1)$ ，那就能解决平移的问题了。点 $(x,y,z)$ 的齐次坐标就是 $(x,y,z,1)$ ，这就变成了4×4矩阵。接下来，我分别介绍这四个关键运算，它们是3D图形显示在屏幕上的第一步，也就是坐标系变换要做的事情，比如：将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

## 平移

我们沿着向量 $\mathbf{v}_0$ 平移整个三维空间，把原点平移到了 $(x_0, y_0, z_0)$ ，这也就意味着三维空间的每个点都加上了点 $(x_0, y_0, z_0)$ 。使用齐次坐标，把整个空间平移了 $\mathbf{v}_0$ 的4×4矩阵 $T$ 如下所示。

```
$$
T=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
x_0 & y_0 & z_0 & 1
\end{array}\right]
$$
```

这里很重要的一点是，计算机图形图像是基于行向量计算的。也就是说，计算方法是行乘矩阵，而不是矩阵乘列，比如：  
 $\left[\begin{array}{llll} 0 & 0 & 0 & 1 \end{array}\right] T = \left[\begin{array}{llll} x_0 & y_0 & z_0 & 1 \end{array}\right]$ 。

平移的整个过程是这样的：假设要把原来的某个点 $(x, y, z)$ 平移 $\mathbf{v}_0$ ，我们需要切换到齐次坐标 $(x, y, z, 1)$ ，然后， $(x, y, z, 1)$ 再乘 $T$ ，就能得到每个原来的向量 $\mathbf{v}$ 平移到 $\mathbf{v} + \mathbf{v}_0$ 的最终结果：  
 $\left[\begin{array}{llll} x & y & z & 1 \end{array}\right] T = \left[\begin{array}{llll} x+x_0 & y+y_0 & z+z_0 & 1 \end{array}\right]$ 。

这里你需要注意：一个行向量乘 $T$ 的结果还是一个行向量。

## 缩放

在前端开发中，我们经常会调整图片宽度和高度来适配页面，比如：把图片整体放大90%，那么在线性代数中就是0.9乘单位矩阵。在二维平面中，我们通常用2×2矩阵来表达缩放，在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中，就不一样了，需要大一个维度，也就是说，3×3矩阵变成了4×4矩阵。

比如，二维平面中图片放大90%就是：

```
$$
S=\left[\begin{array}{ccc}
0.9 & 0 & 0 \\
0 & 0.9 & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

三维立体中图片放大90%就是：

```
$$
S=\left[\begin{array}{cccc}
0.9 & 0 & 0 & 0 \\
0 & 0.9 & 0 & 0 \\
0 & 0 & 0.9 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]
$$
```

缩放还可以在不同的方向上进行，比如：一个二维平面图片从整页适配调整到半页适配， $y$ 方向就要乘 $\frac{1}{2}$ ，创建一个 $\frac{1}{4}$ 的页边留白， $x$ 方向就要乘 $\frac{3}{4}$ ，这样得到的缩放矩阵就是：

```
$$
S=\left[\begin{array}{lll}
\frac{3}{4} & 0 & 0 \\
0 & \frac{1}{2} & 0 \\
0 & 0 & 1
\end{array}\right]
$$
```

平移和缩放组合情况会怎样呢？如果我们要先平移再缩放，那应该这样乘： $\mathbf{v}TS$ ，如果我们要先缩放再平移，那应该这样乘： $\mathbf{v}ST$ 。注意：它们乘的顺序是不同的，哪个运算先做就先乘，因为矩阵的左乘和右乘的结果是不同的。

在第九篇的[仿射空间](#)中提到了平移和缩放矩阵，你也可以回过头再去看看。

## 旋转

二维和三维空间的旋转由正交矩阵 $Q$ 来完成，它的行列式是+1。同样我们使用齐次坐标，一个平面旋转的正交矩阵 $Q$ 就从 $2\times 2$ 就变成了 $3\times 3$ 矩阵 $R$ 。

```


$$Q = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$


$$R = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$


```

这个矩阵是围绕原点旋转了平面，那如果矩阵旋转时围绕的不是原点，而是其他点呢？这个就稍微复杂一些，不是直接旋转，而是先平移再旋转，比如我们要围绕点 $(4,5)$ ，让平面旋转 $\theta$ 角度的话：

1. 首先，要把 $(4,5)$ 平移到 $(0,0)$ ；
2. 接着，旋转 $\theta$ 角度；
3. 最后，再把 $(0,0)$ 平移回 $(4,5)$ 。

整个过程通过数学公式来表达就是：

```


$$T_{00} R_{45} = \begin{bmatrix} x & y & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -5 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix}$$


```

说完二维我们再来说三维。不过在三维空间中，旋转就有些不一样了，因为它是围绕一个轴“翻转”的。更“数学”的说法就是，围绕 $\lambda=1$ 的特征向量的一条线翻转。

现在，我们来看看分别围绕 $x$ 、 $y$ 和 $z$ 轴方向旋转的矩阵 $R$ 有什么不同？

1.围绕 $x$ 轴方向旋转：

```


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


```

2.围绕 $y$ 轴方向旋转：

```


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


```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. 围绕\$z\$轴方向旋转：

$$R_z = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

你看出来哪里不同了吗？其实主要就是1的位置不同，以及\$y\$轴方向旋转的\$\sin\$互换了。

## 投影

现在，我们想把3D图形显示到二维屏幕上，该怎么做呢？

从数学角度理解就是把三维向量投影到平面上。在线性代数中，我们看到的大部分的平面都是通过原点的，但在现实生活中则不是。一个通过原点的平面是一个向量空间，而其他的平面则是仿射空间，具体仿射空间的定义你可以回顾一下[第九篇](#)的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面，它的单位法向量是\$\mathbf{n}\$，那么平面中的向量\$\mathbf{v}\$，满足这个等式：\$\mathbf{n}^T \mathbf{v} = 0\$。

而投影到平面的投影矩阵是：\$\mathbf{I} - \mathbf{n} \mathbf{n}^T\$。

如果把原来的向量和这个投影矩阵相乘，就能投影这个向量。我们可以用这个投影矩阵来验证一下：单位法向量\$\mathbf{n}\$投影后成为了0向量，而平面向量\$\mathbf{v}\$投影后还是其自身。

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{n} = \mathbf{n} - \mathbf{n}(\mathbf{n}^T \mathbf{n}) = \mathbf{0}$$

$$(\mathbf{I} - \mathbf{n} \mathbf{n}^T) \mathbf{v} = \mathbf{v} - \mathbf{n}(\mathbf{n}^T \mathbf{v}) = \mathbf{v}$$

接下来，我们在齐次坐标中来看一下\$4 \times 4\$的投影矩阵：

$$P = \begin{pmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$$

假设现在有一个不过原点的平面，\$\mathbf{v}\_0\$是这个平面上的一个点，现在要把\$\mathbf{v}\_0\$投影到这个平面，则需要经历三个步骤，和刚才介绍的围绕点\$(4,5)\$，让平面旋转\$\theta\$角度经历的三个步骤类似：

1. 把\$\mathbf{v}\_0\$平移到原点；
2. 沿着\$\mathbf{n}\$方向投影；
3. 再平移回\$\mathbf{v}\_0\$。

整个过程通过数学公式来表达就是：

$$T_{-v_0} P T_{+v_0} = \begin{pmatrix} \mathbf{I} - \mathbf{n} \mathbf{n}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

## 计算机3D图形介绍

有了数学知识的铺垫，我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换（又称为空间变换），才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤，其中，第一步，也是最重要的一步就是坐标系变换，将局部坐标系表示的点变换到世界坐标系中，然后再变换到视图坐标系（或叫摄像机坐标系），接着继续变换到裁剪坐标系（投影坐标系）。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系（摄像机坐标系），则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明，它有个相对复杂的推导过程的，感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系（摄像机坐标系）中的一个点变换到裁剪坐标系（投影坐标系），则可以使用投影矩阵进行操作。

最后，我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战：基于WebGL的3D架构与实现》，作者：步磊峰，这本书描述了3D图形处理的基本数学知识的同时，更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》，作者：Hughes, Van Dam, McGuire, Sklar, Foley, Feiner, Akeley，这本书虽然也有实践，但更偏重计算机图形理论一些。

## 本节小结

今天的整篇内容都是围绕三维空间的变换展开的，你需要掌握三维空间中的四个关键运算：平移、缩放、旋转和投影的基本概念，以及对应的平移、缩放、旋转和投影矩阵，这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中，三维物体从取景到屏幕显示，需要经历一系列的坐标变换，才能生成二维图像显示在输出设备上。了解了这些之后，你就能掌握计算机3D图形处理的本质，也许还能在将来的实践中优化图形渲染效率。

## 线性代数练习场

今天我要给你一道开放题：如果把正方形投影到一个平面上，你会得到一个什么形状的图形？

欢迎在留言区晒出你的结果和思考，我会及时回复。同时，也欢迎你把这篇文章分享给你的朋友，一起讨论、学习。