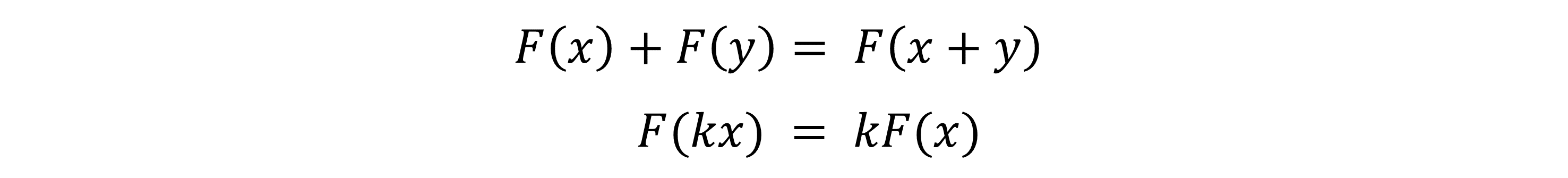
笔者最近在回顾一些图形学基础知识，遂整理在此，此文主要讲述图形学中的变换。

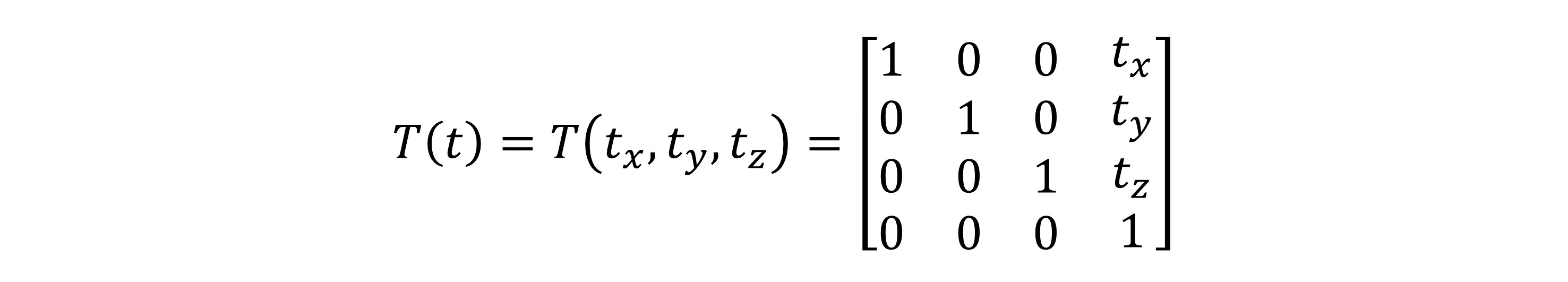
关于变换的内容过多，就打算分为几篇文章分别讲述。本文首先就几种基础的变换进行讲述，如平移、旋转、缩放等；随后进行一些扩展：如绕任意轴旋转的矩阵、视点矩阵的推导、对法向量变换的方法、求矩阵的逆的方法

线性变换(Linear transform)，满足以下式子：

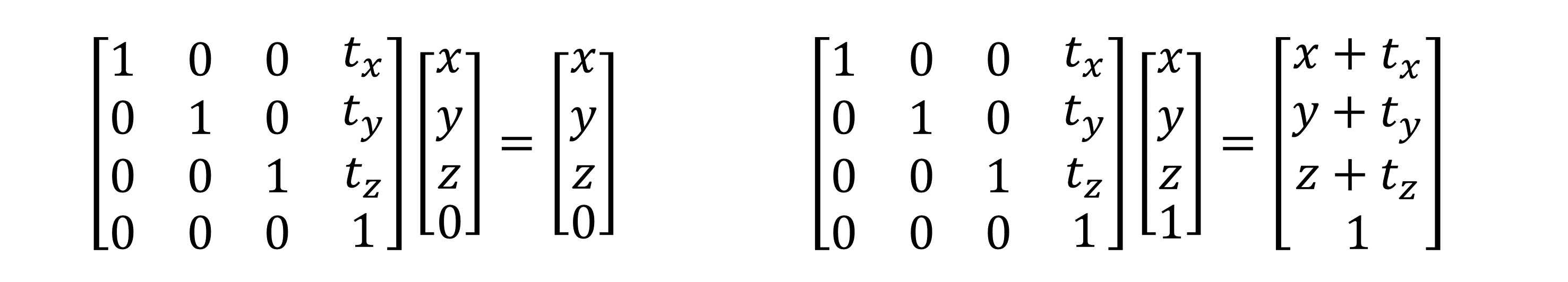


线性变换的性质：对于n维向量的所有线性变换都可以用n\*n矩阵表示。常见的缩放和旋转都是线性变换，但平移不是。线性变换结合平移，称之为仿射变换(Affine transform)，对于3维向量的仿射变换通常可以用4维矩阵表示，那么向量也需要4维的表示方式，本文对于向量的表示采取列向量的形式，且遵循右手定则。区别方向向量和点向量，方向向量代表一个方向，表示为(x,y,z,0)，点向量表示空间中的一个点，表示为(x,y,z,1)

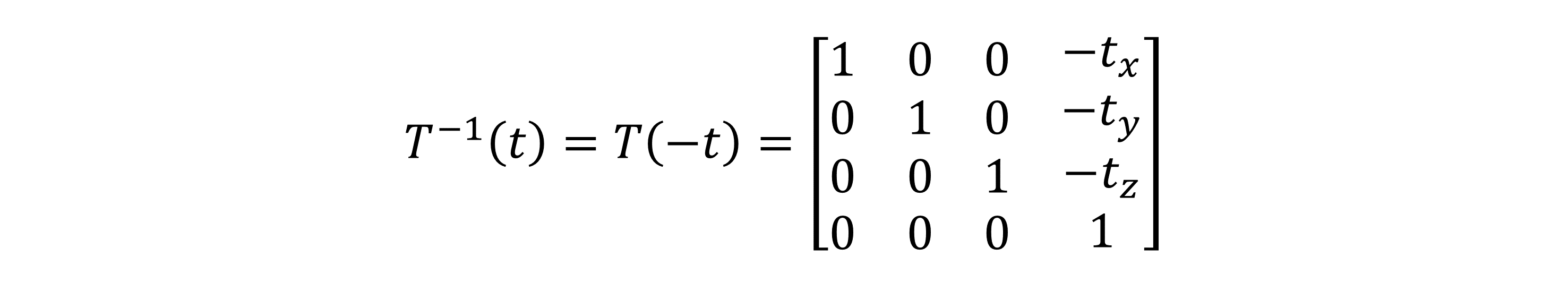
1、平移变换，设平移矢量为t，那么平移变换矩阵表示如下：



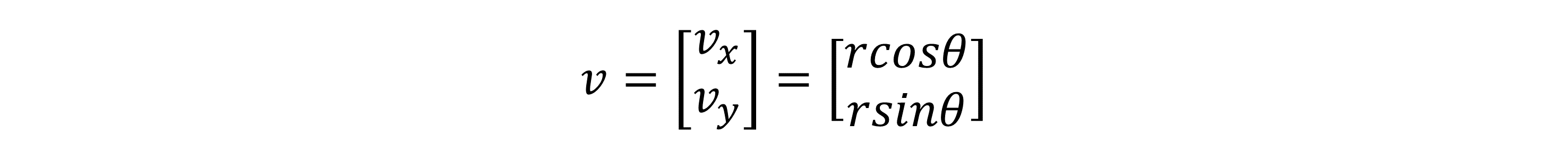
平移操作是针对点而言的，所以分别对方向(x,y,z,0)和点(x,y,z,1)操作会得到不同的结果：



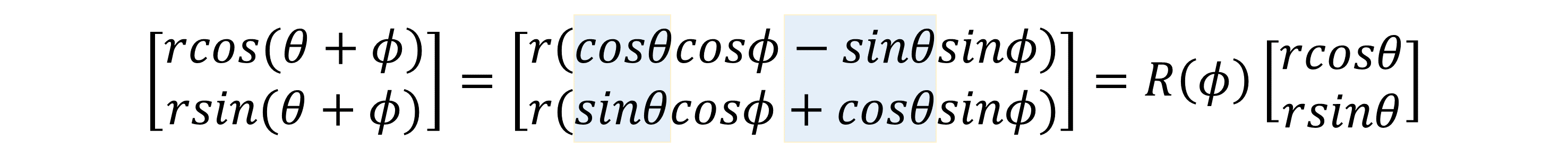
平移变换的逆就相当于平移回去，即：



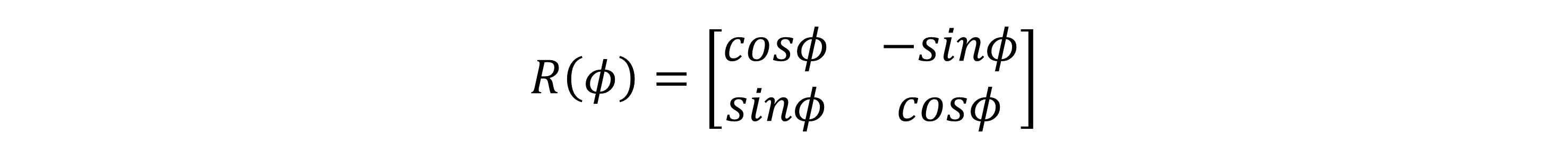
2、旋转变换，先看一下二维的旋转矩阵，假设一个二维向量及其极坐标系下的表示：



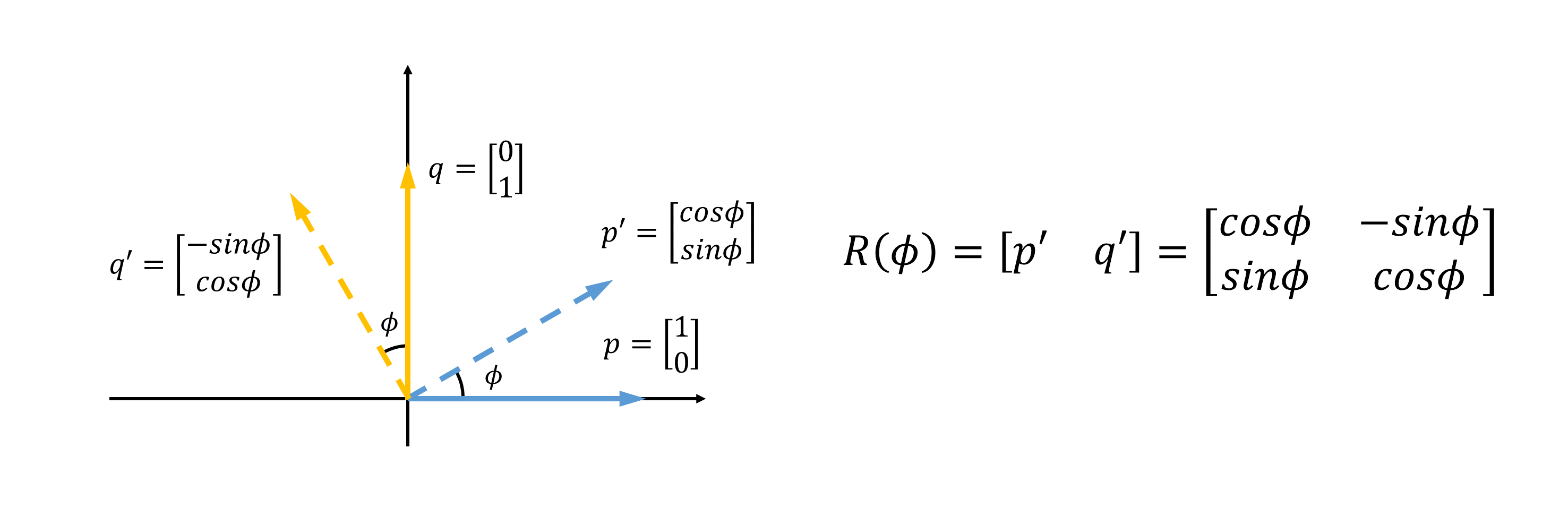
如果绕原点将其逆时针旋转一个角度（此文遵守右手定则），那么可表示为：



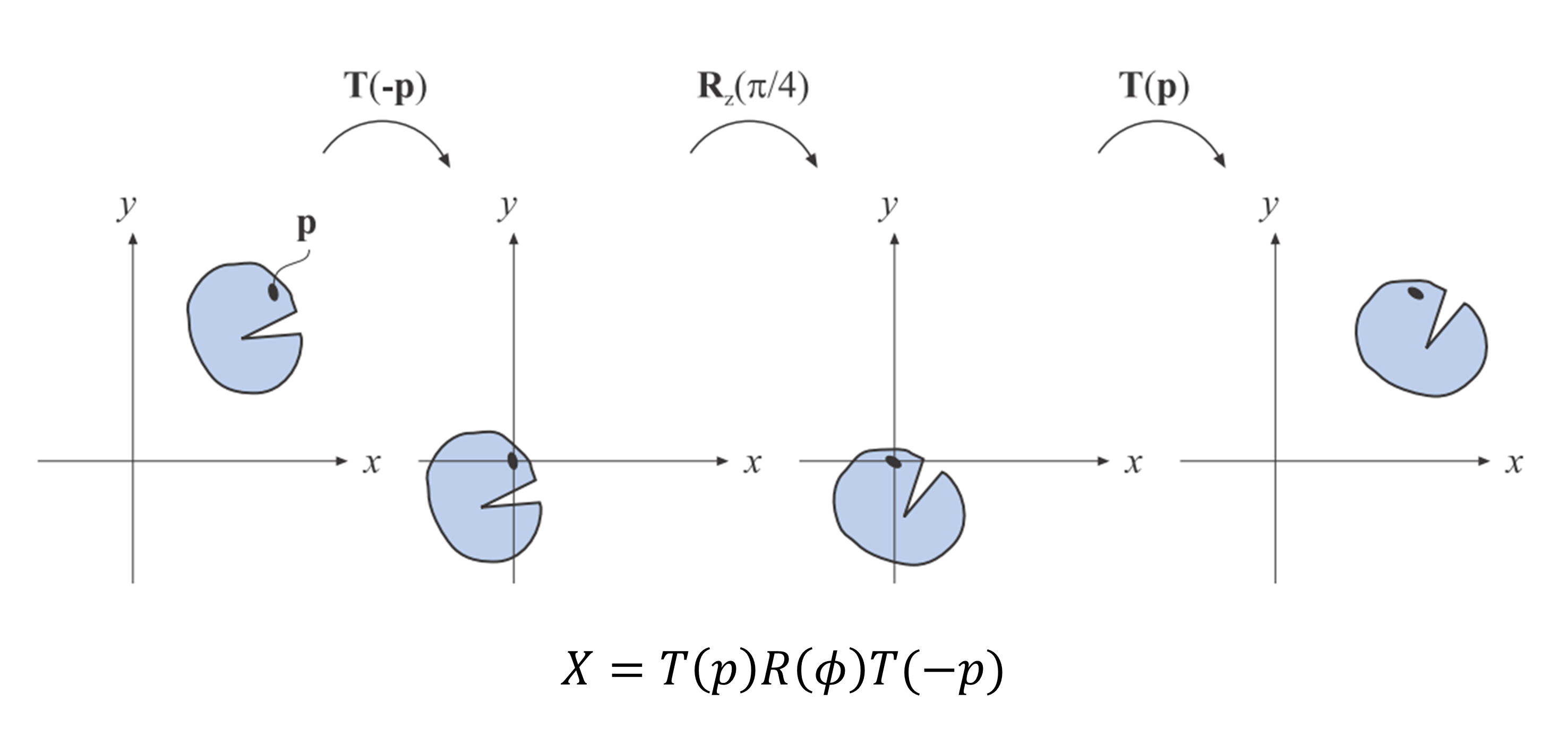
于是可推出绕原点的二维旋转矩阵：



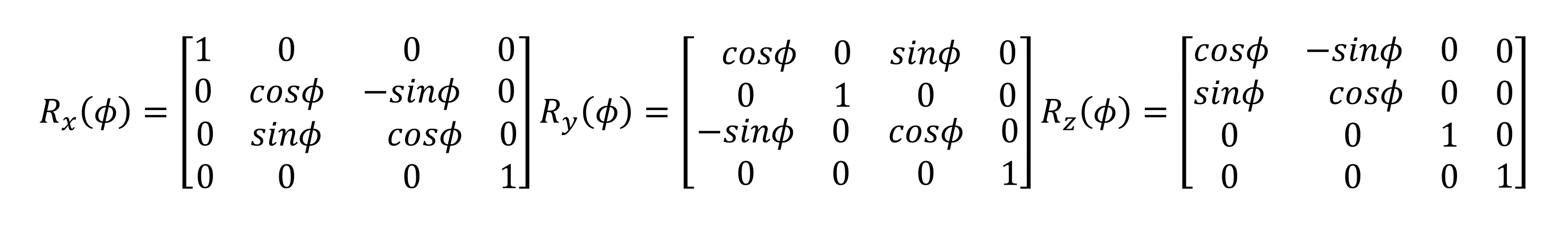
关于绕原点的二维旋转矩阵，还有另外一种推导方法：把基矢量变换后的结果直接放进矩阵的列（如果向量表示为行向量，则放进行），即可构成变换矩阵。于是有：



但是如果我们想要绕任意点p旋转，就需要结合平移操作，先将p点平移到原点，执行旋转后再平移回去：

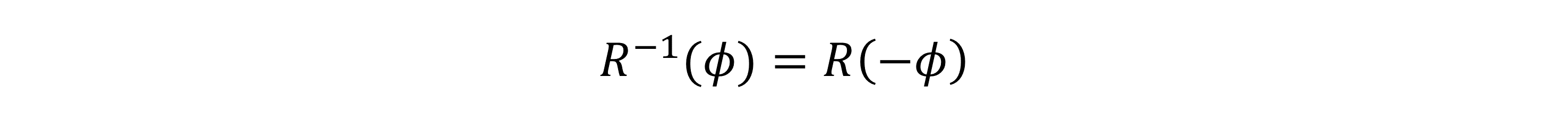


二维绕原点的旋转矩阵很容易推广到三维绕坐标轴的旋转矩阵：

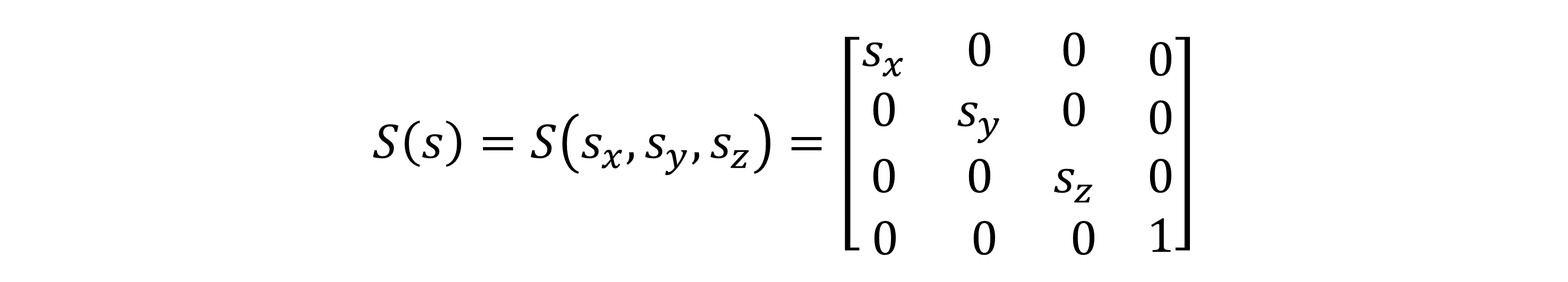


注意此处绕y轴旋转的矩阵与x和z轴的sin参量的正负号不一样，我不太能理解。这里给出闫令琪老师的一个解释：绕x轴旋转是从y到z，绕z轴旋转是从x到y，y到z和x到y都符合平常所说的xyz的顺序，而绕y轴旋转是从z到x，不符合xyz的顺序。以此可辅助我们来记忆绕y轴旋转矩阵的符号相反情况

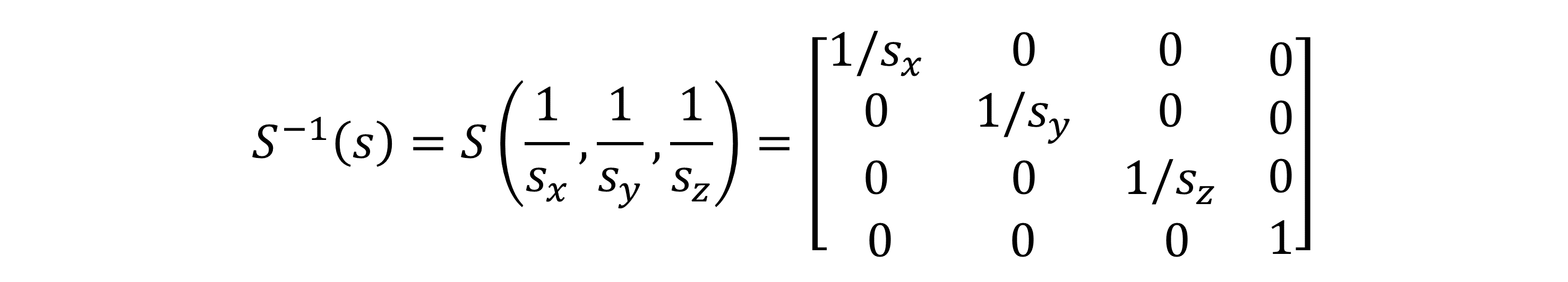
旋转矩阵的逆和平移类似，就是旋转回去，即：



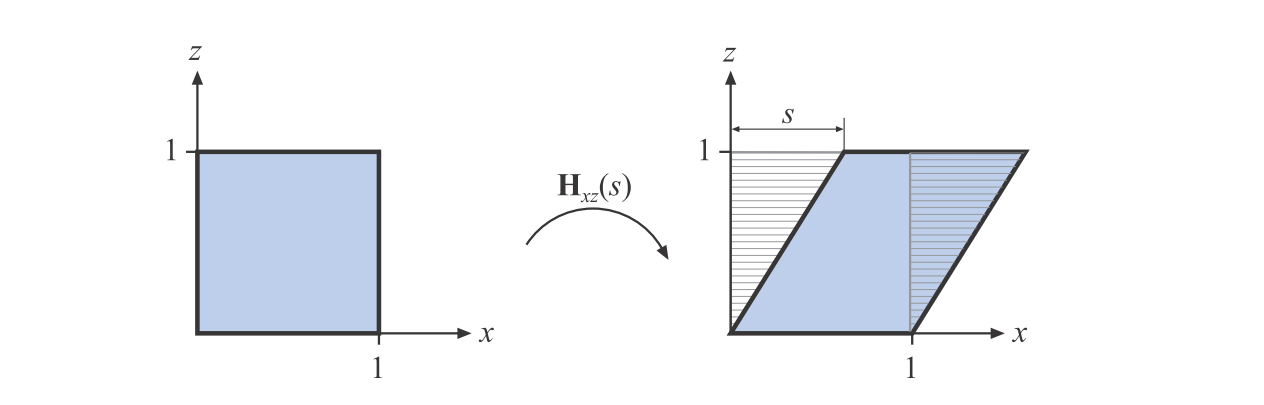
3、缩放，设缩放因子为s，则沿坐标轴的缩放变换矩阵可写作：



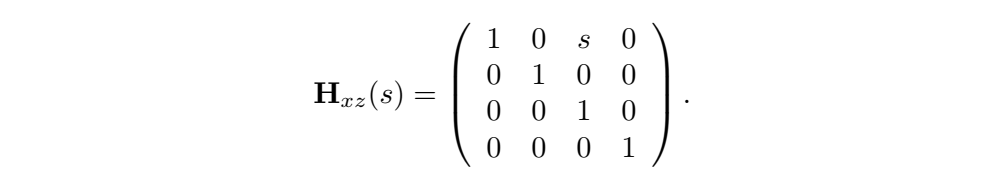
若缩放因子的三个分量相等，则称之为均匀缩放，否则为非均匀缩放。缩放矩阵的逆，就是取缩放因子的倒数：



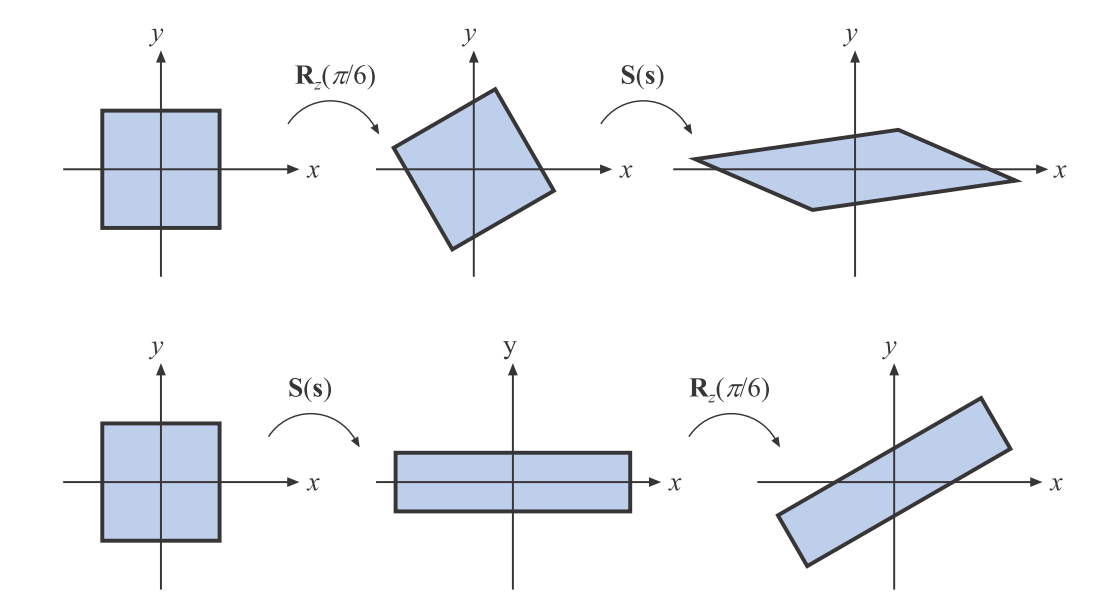
4、错切，就好像把一根坐标轴倾斜了一样，比如把z分量的s倍加到x分量上去：



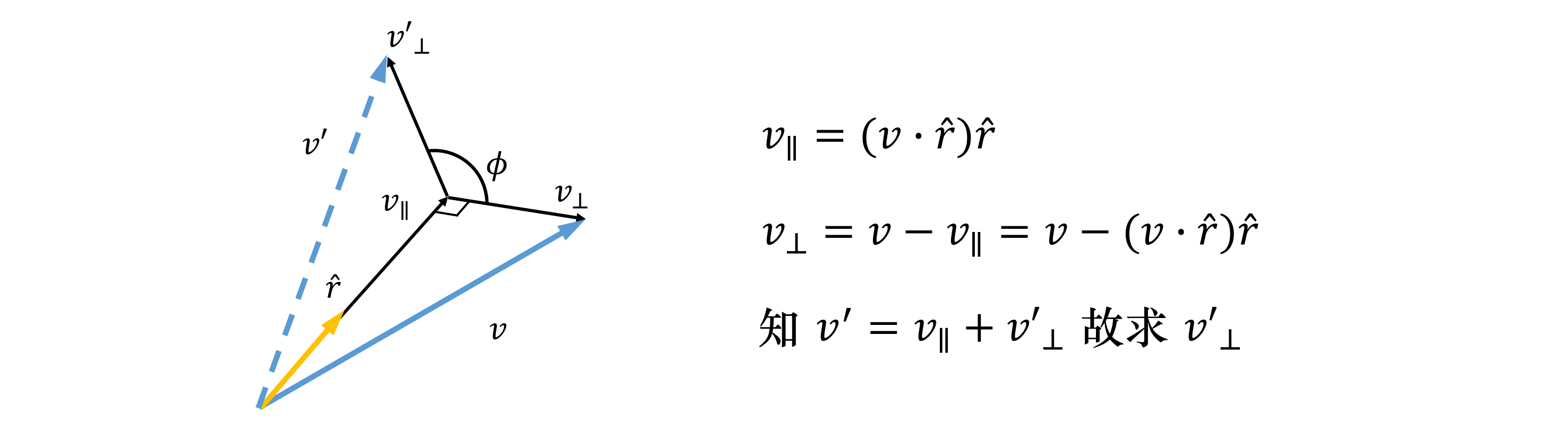
此错切矩阵可写为：



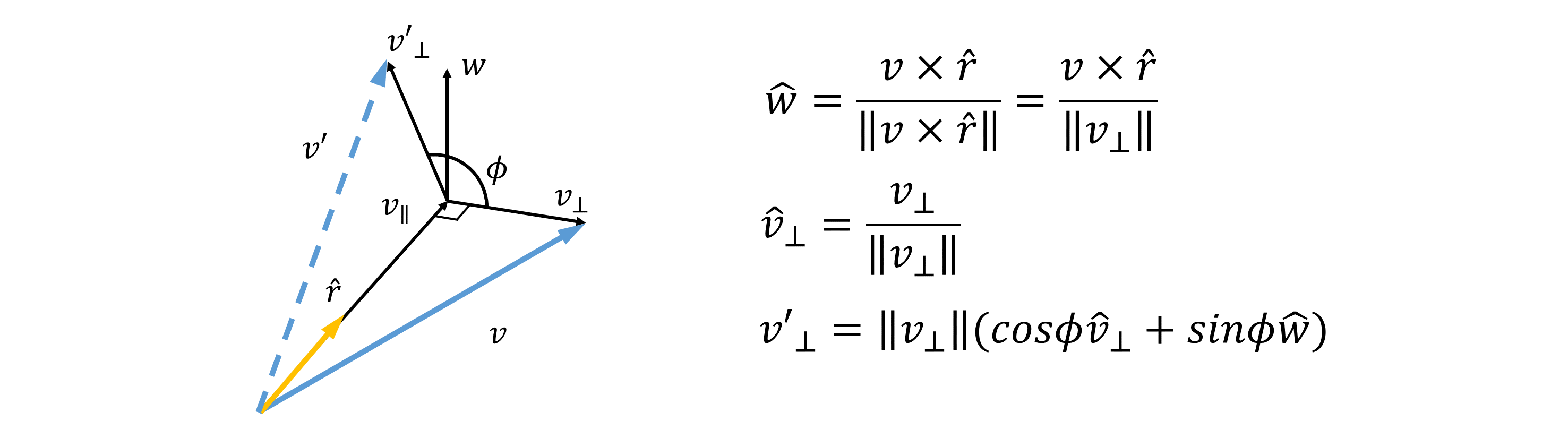
最常用的变换有三个：平移、旋转、缩放，虽然我们可以将多个变换矩阵组合为一个，但是要注意次序不同会带来不同的结果。一种比较常用的顺序是先缩放再旋转最后平移，即TRS的顺序。下面一个例子，先执行旋转再执行缩放可能得不到我们想要的结果：



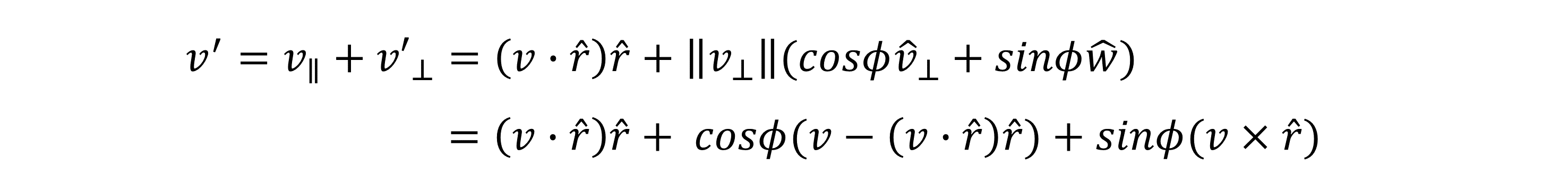
扩展一：三维空间中绕任意轴旋转矩阵的推导。如图所示，我们将向量v绕r单位向量旋转一个角度，即要写出v’的表达式，可考虑把v分解为垂直于r和平行于r方向的两个向量：



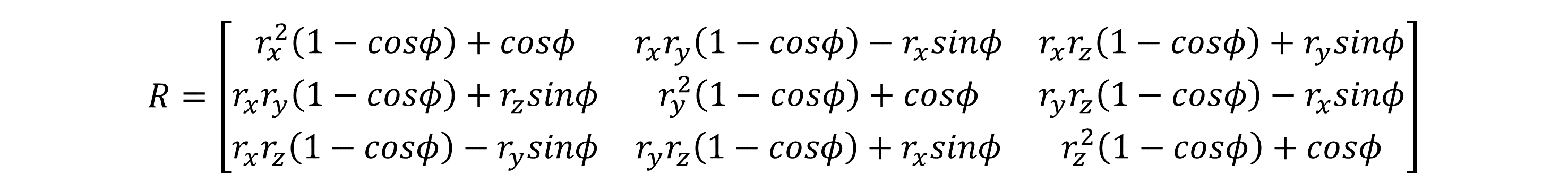
于是引入w向量，建立坐标系：



于是可把v’ 表示出来：

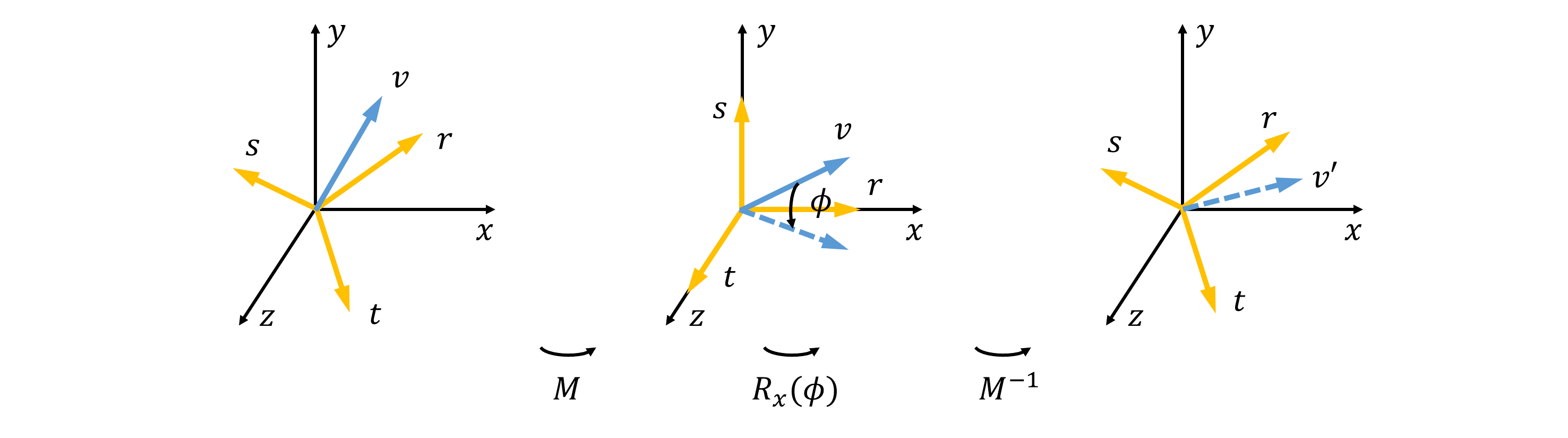


将其展开为矩阵形式：

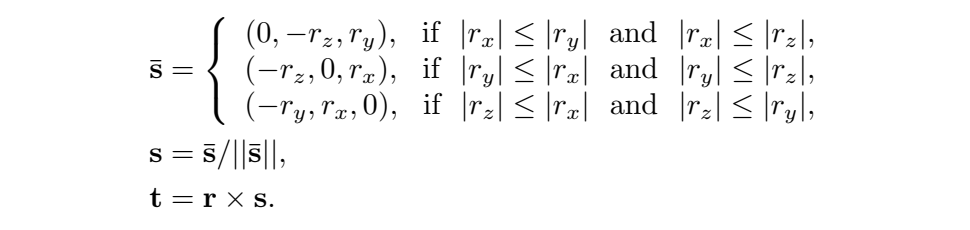


此外再介绍一种绕任意向量旋转矩阵的推导方法：

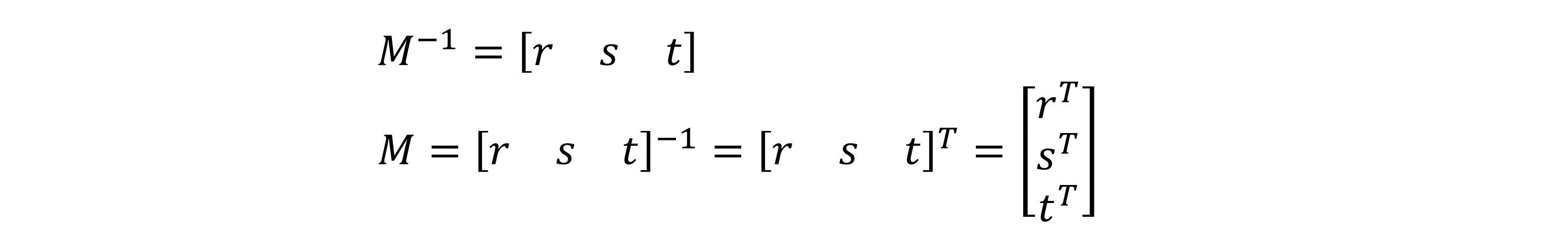
要绕r向量旋转，可以再找到两个单位向量st，使rst 构成一个新的坐标系。如果rst坐标系变换到与xyz坐标系重合，那么绕r向量旋转也就转化为了绕x轴旋转的问题，之后再变换回来即可。



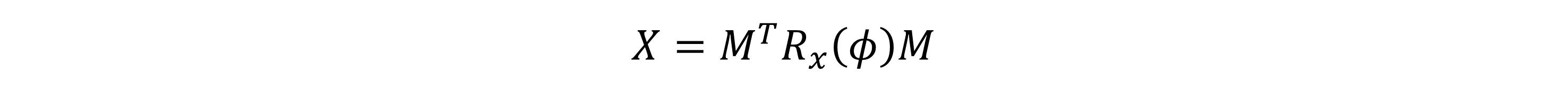
先来找 st，使 rst 两两垂直，一个数值稳定的方法是：将 r 向量中绝对值最小的那个分量置为0，再把另外两个分量交换，且将其中一个取相反数，归一化后即可得到一个比较合适的 s, 然后将 r 与 s 求叉积可得到 t ：



再来确定把rst变换到与xyz重合的矩阵M，前面另一种推导二维旋转矩阵的时候也说到：把基矢量变换后的结果直接放进矩阵的列，即可构成变换矩阵。考虑从xyz变换为rst，即把rst分别放在一个矩阵的三列，就构成了M的逆，那么得到（由于rst是单位向量且正交，故M的逆也为M的转置）：



再结合绕x轴的旋转矩阵（此前已经给出），可得到整体的变换矩阵：

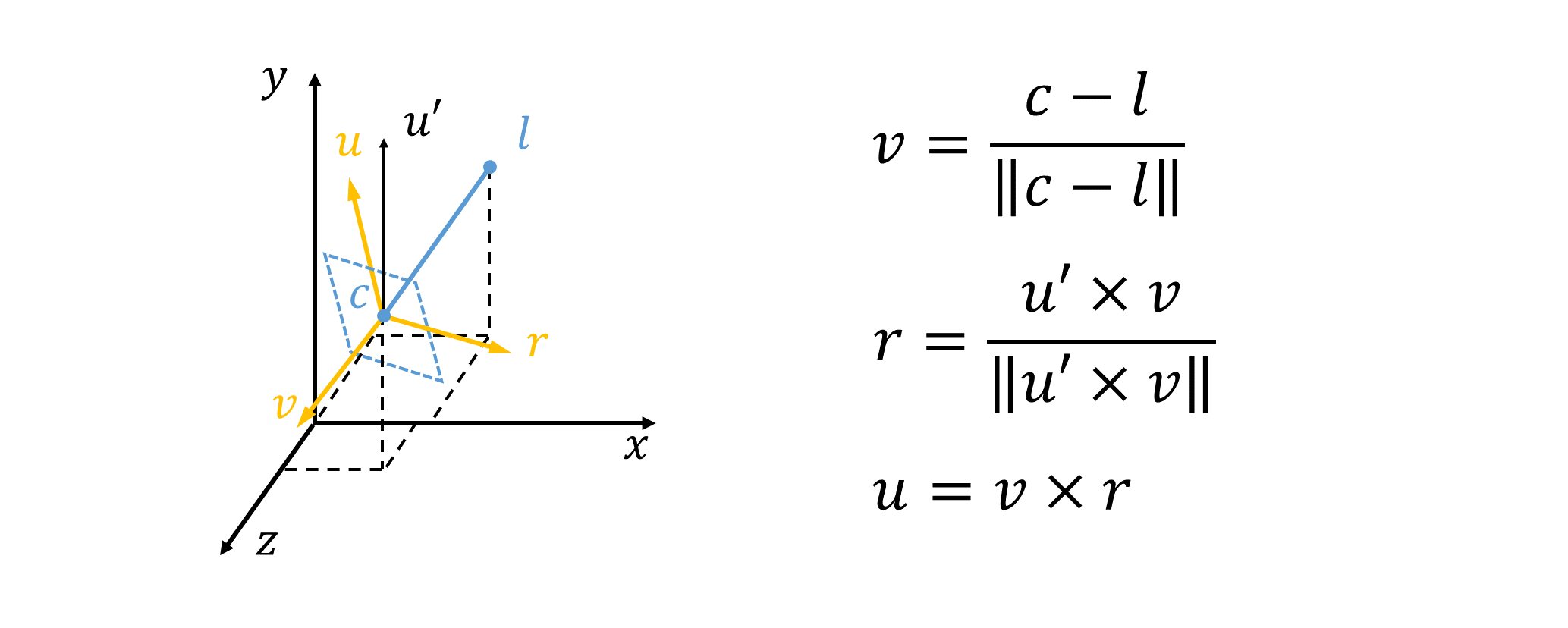


如若展开，也可得到和上一种方法一样的结果。

扩展二：视点矩阵的推导

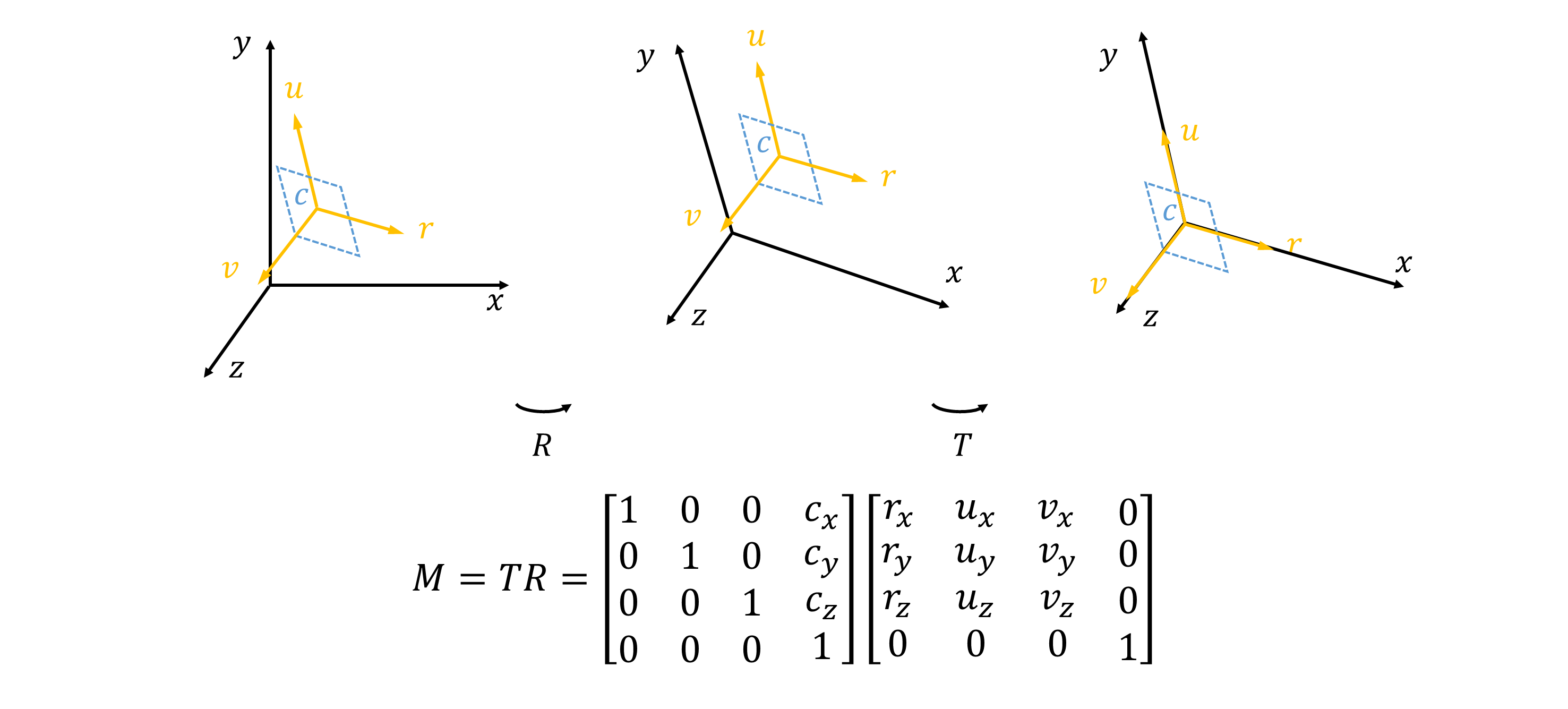
视点矩阵，用于将世界空间中的矢量转到相机空间，即求出矢量在相机坐标系中的坐标，这个变换矩阵涉及到平移和旋转，是严格的刚体变换，即不改变物体的形状和大小。此处主要呈现OpenGL中gluLookAt()矩阵的推导方式：

设相机的世界位置坐标为c，且相机正朝向世界系中一点l，且给出相机的向上向量为u’（OpenGL里面一般取u’=(0,1,0)）,一种约定俗成的相机坐标系建立方式为：取c-l并归一化后为相机坐标系的v轴，其方向从l指向c，u’与v叉乘并归一化后为相机坐标系的r轴，其方向向右，最后再将v与r叉乘得到相机坐标系的u轴，其方向向上：

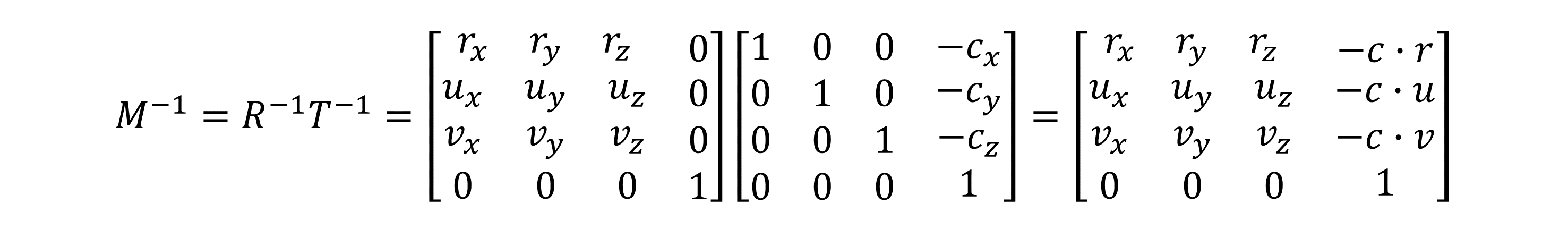


我对这种建立相机坐标系的方法谈一些自己的理解：首先就是那个向上向量u’，OpenGL里面一般传的是y轴方向(0,1,0)进去，这样经过一步步确定vru可以让v向量、u向量和u’向量在同一平面内，且此平面与xOz平面始终垂直。想象你在游戏场景里面操控一个人的头，你可以鼠标左右滑动改变Yaw左右转头，可以鼠标上下滑动改变Pitch抬头或者低头，却不能改变Roll来左右歪头。实际上在传入的u’确定的情况下推出来的相机坐标系，始终都不能让你有歪头动作，因为一旦歪头，那么v向量u向量和u’向量就不在同一平面了，这与上述的推导是相驳的。假设另外一种场景，在第一人称视角模拟开飞机，那么如果要翻滚Roll的话，就应该是实时改变传入的u’，不让其固定为(0,1,0)

建立了相机坐标系，现推导视点矩阵，我们可以逆向思考：对坐标轴执行M变换，相当于对坐标执行M的逆变换。我们要把世界系坐标转到相机坐标系，所以先考虑怎样将世界系变换到与相机系重合，即x与r重合，u与y重合，v与z重合。先进行旋转，即把ruv直接写进矩阵的列得到旋转矩阵，然后再进行位移量为c的平移，即：



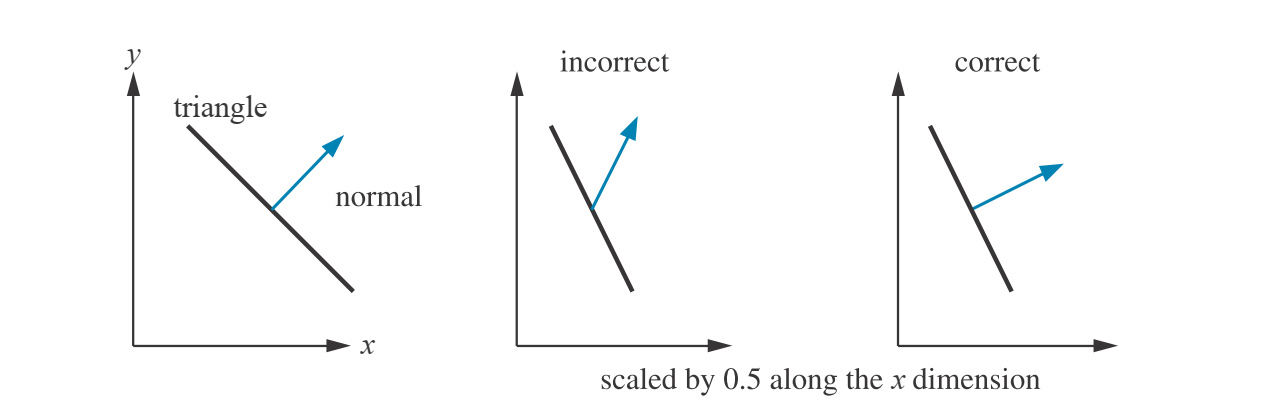
此矩阵是对坐标轴的变换矩阵，那么对坐标的变换矩阵就是其逆矩阵：



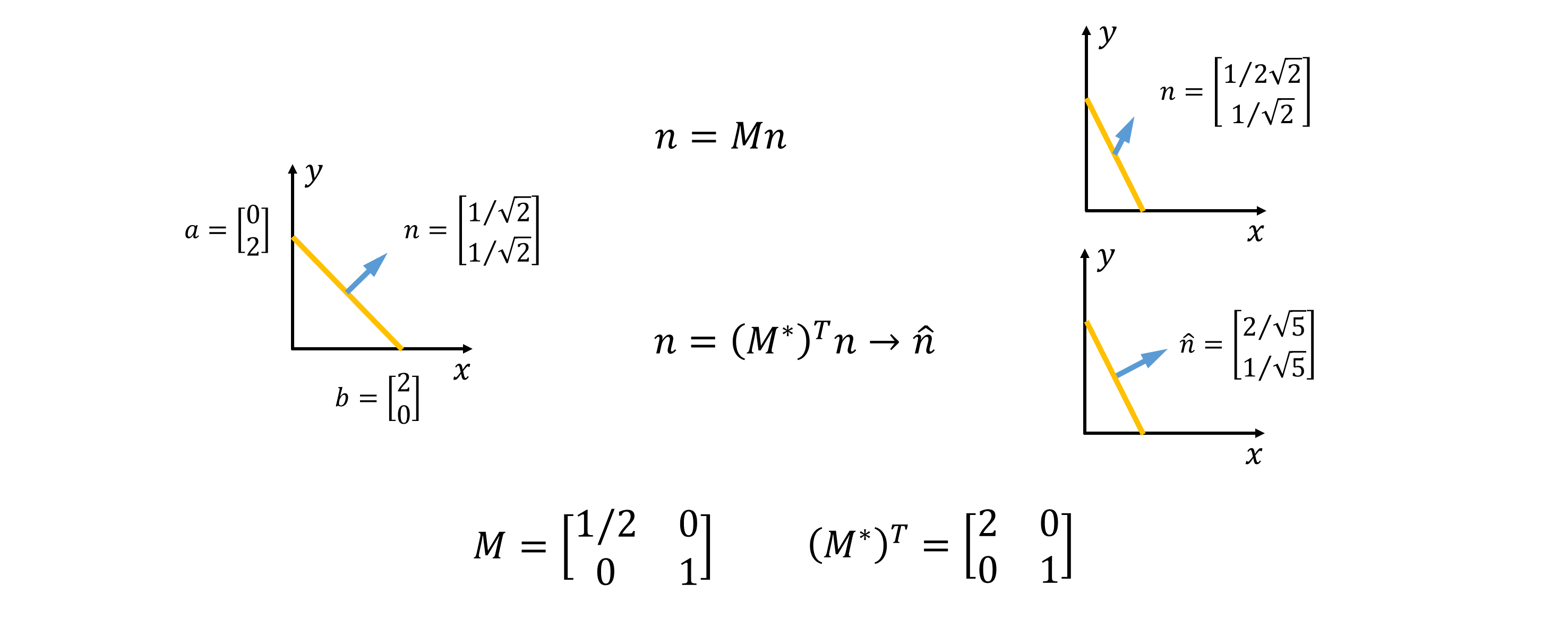
此矩阵即为视点矩阵的最终表达式

扩展三：对法向量变换的处理

考虑下面这种情况，对一条线段进行非均匀缩放，如果对法向量执行相同的矩阵操作，其结果可能就不对：



也就是说，在某些情况下，我们不能直接对法向量采取和点向量相同的矩阵变换，一个有效的方法是：对点执行M变换，那么对法向量执行M伴随阵的转置再归一化：



由于求伴随阵往往比求逆更复杂，且伴随阵与逆只差一个行列式的倍数，在归一化后就抵消了，所以也可这样：对法向量执行M的逆的转置，再归一化。但是如果对每个变换矩阵都求其逆矩阵，计算量也不能接受，所以对一些比较常用的变换有以下结论：平移变换、旋转矩阵不改变法向量，所以不用额外考虑；均匀缩放虽然改变法向量大小，且不改变方向，只需要用缩放因子归一化即可

扩展四：求矩阵逆的几种方法

1. 如果矩阵是几个简单矩阵的组合，那么可分别求简单矩阵的逆。如M=T(t)R(a)，则Inverse(M)=R(-a)T(-t)
2. 如果矩阵已知是正交阵，那么其逆矩阵就是其转置。如Inverse(R)=Transpose(R)
3. 如果是一般矩阵，那么只能用诸如克拉默法则、伴随阵方法等解析方法，和LU分解、高斯消去法等数值方法进行求解。

此文对平移、旋转、缩放等几个基础的变换做了讲述，并进行了一些深入扩展：如绕任意轴的旋转矩阵推导、视点矩阵的推导、处理法向量的特殊方法以及求解逆矩阵的方法。但图形学中关于变换的知识还有很多，比如投影、欧拉角与四元数等内容，后面会继续更新，请多指教！