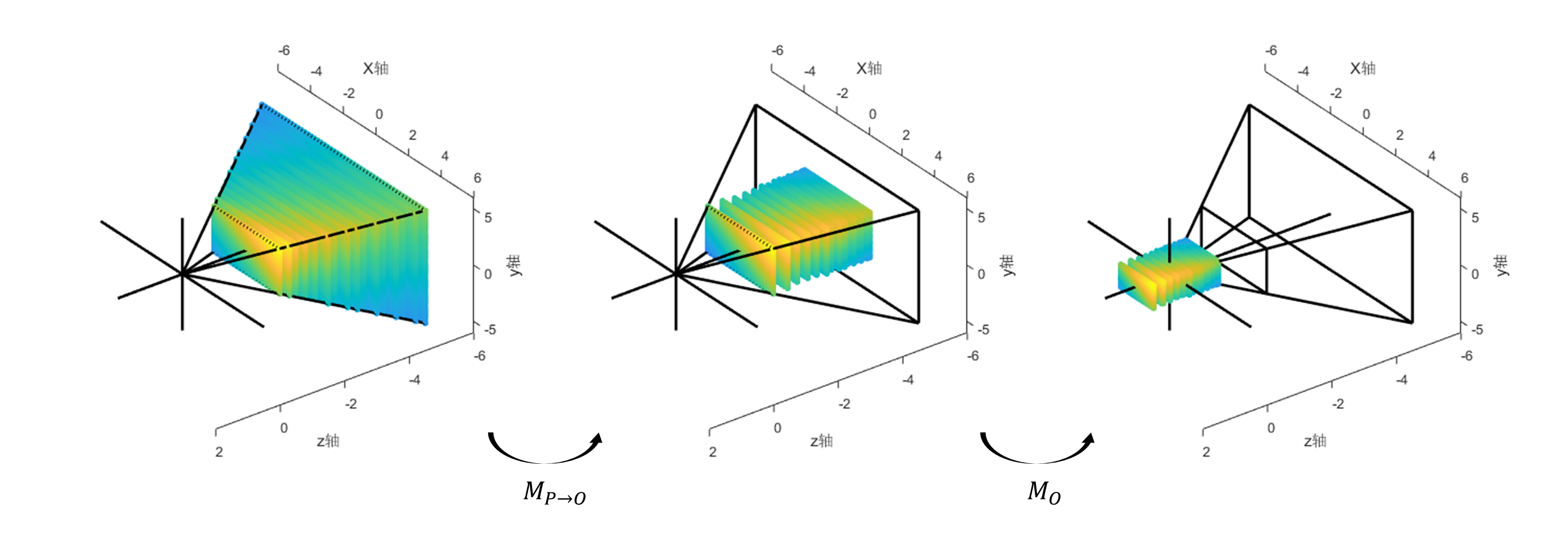
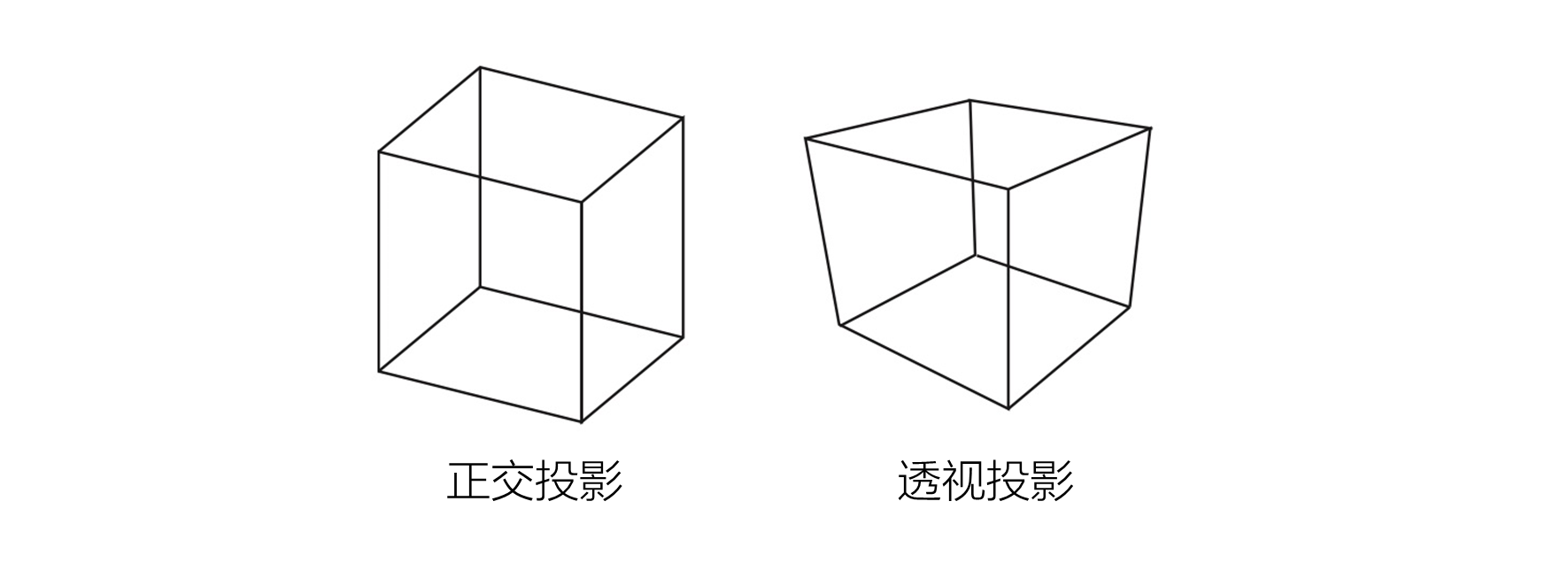
笔者最近在回顾一些图形学基础知识，遂整理在此，此文主要讲述图形学中的变换。

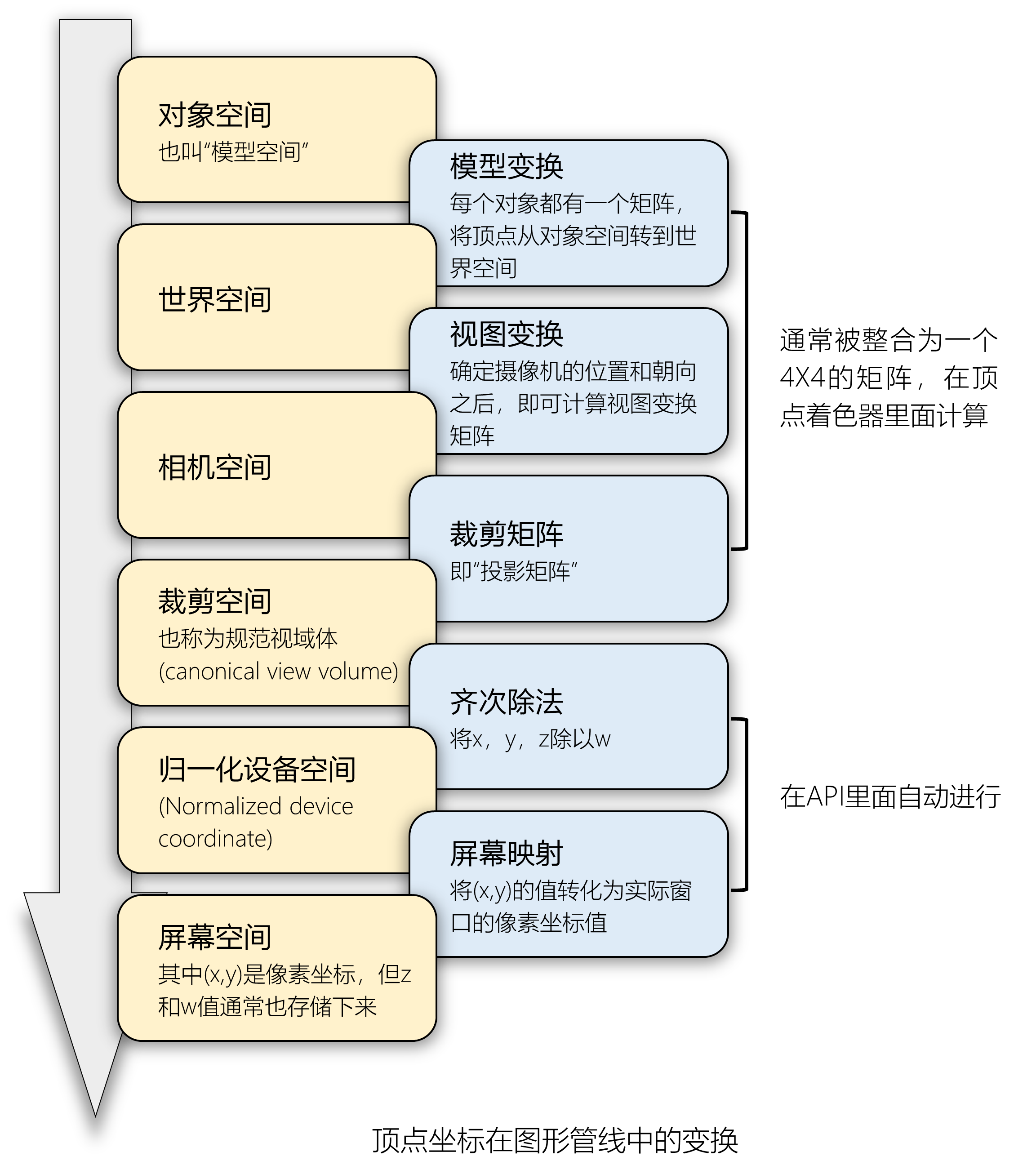
这是关于变换的第四篇文章。对正交投影矩阵和透视投影矩阵进行推导，即推导下面这张图的变换矩阵



投影有正交投影（Orthographic Projection）和透视投影（Perspective Projection），以一个正方体线框举例，正交投影后平行线仍然是平行的，而透视投影则会产生近大远小的效果，且原来的平行线会交于一个点：



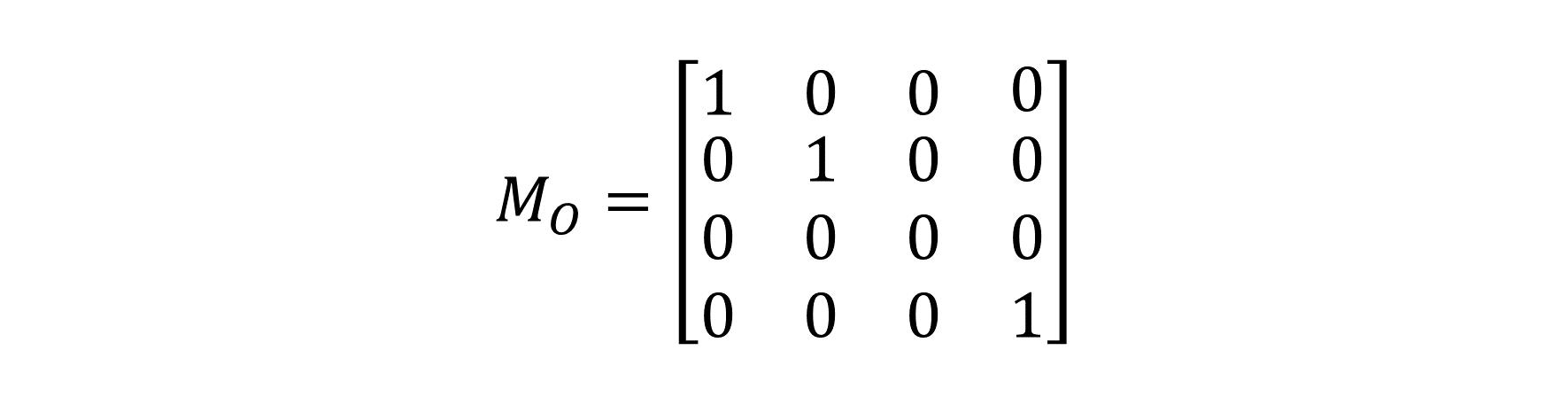
回顾一下之前讲过的顶点变换过程，可以看到投影矩阵实际上是：相机空间到裁剪空间的变换矩阵，而真正要从三维转到二维，则还需要除以w分量，然后取之后的xy的值



图形学里面的投影矩阵也称裁剪矩阵（clip），这是因为经过投影变换后，将原本视锥体内裁剪平面以内的点转换到规范视域体（canonical view volume），而其他的点则被抛弃了，所以称之为“裁剪”。首先来看正交投影矩阵：

正交投影矩阵：

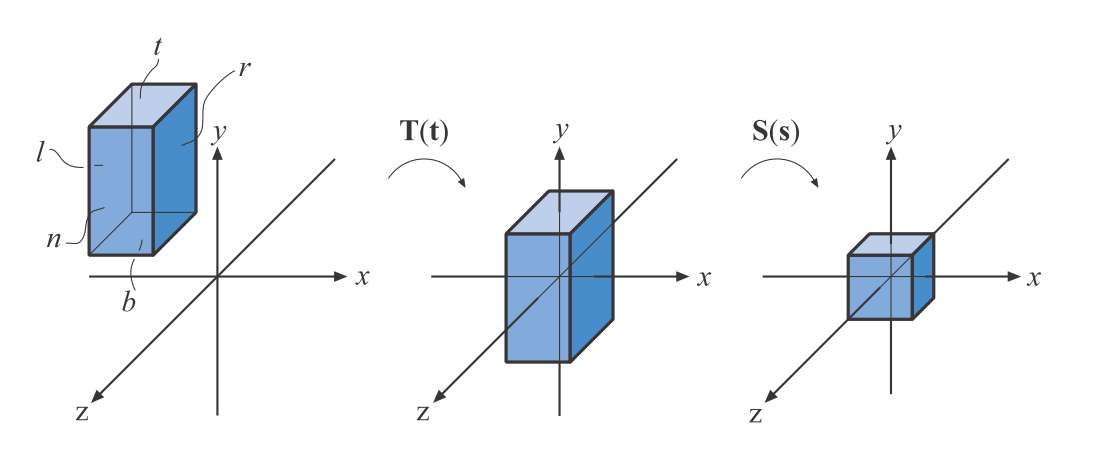
一个简单的正交投影方法就是直接抛弃z坐标，其矩阵形式可写作：



但是它有几个问题：

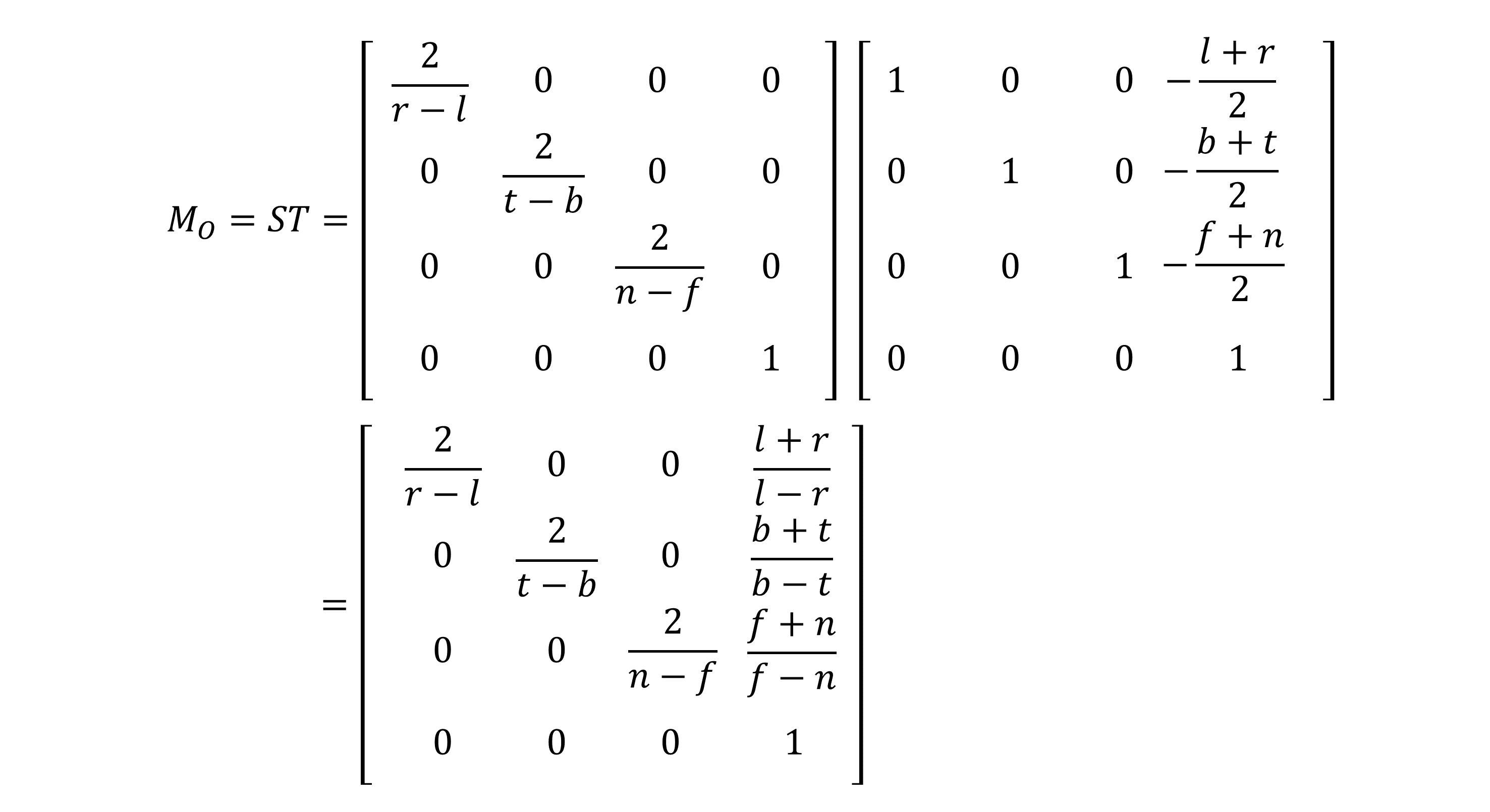
1. 由于其行列式为0，所以此变换不可逆，也就是说，从三维到二维丢失了z方向上这一维的信息
2. 由于同一个(x,y)坐标可能对应不同的值，但这个变换并没有把他们区分开来
3. 此矩阵代表相机在原点时，且对xy坐标[-1,1]范围内的正交投影，范围受限

所以更常用的一种正交投影方法是：用六个参数(l,r,b,t,n,f)定义一个AABB包围盒，然后将这个包围盒变换到规范视域体（OpenGL里是(-1,-1,-1)到(1,1,1)，DirectX里是(-1,-1,0)到(1,1,1)），即从一个长方体变换到另一个定向与之相同的长方体，所以只涉及到平移和缩放操作，如图所示：



From：Real-Time Rendering 4 - P94

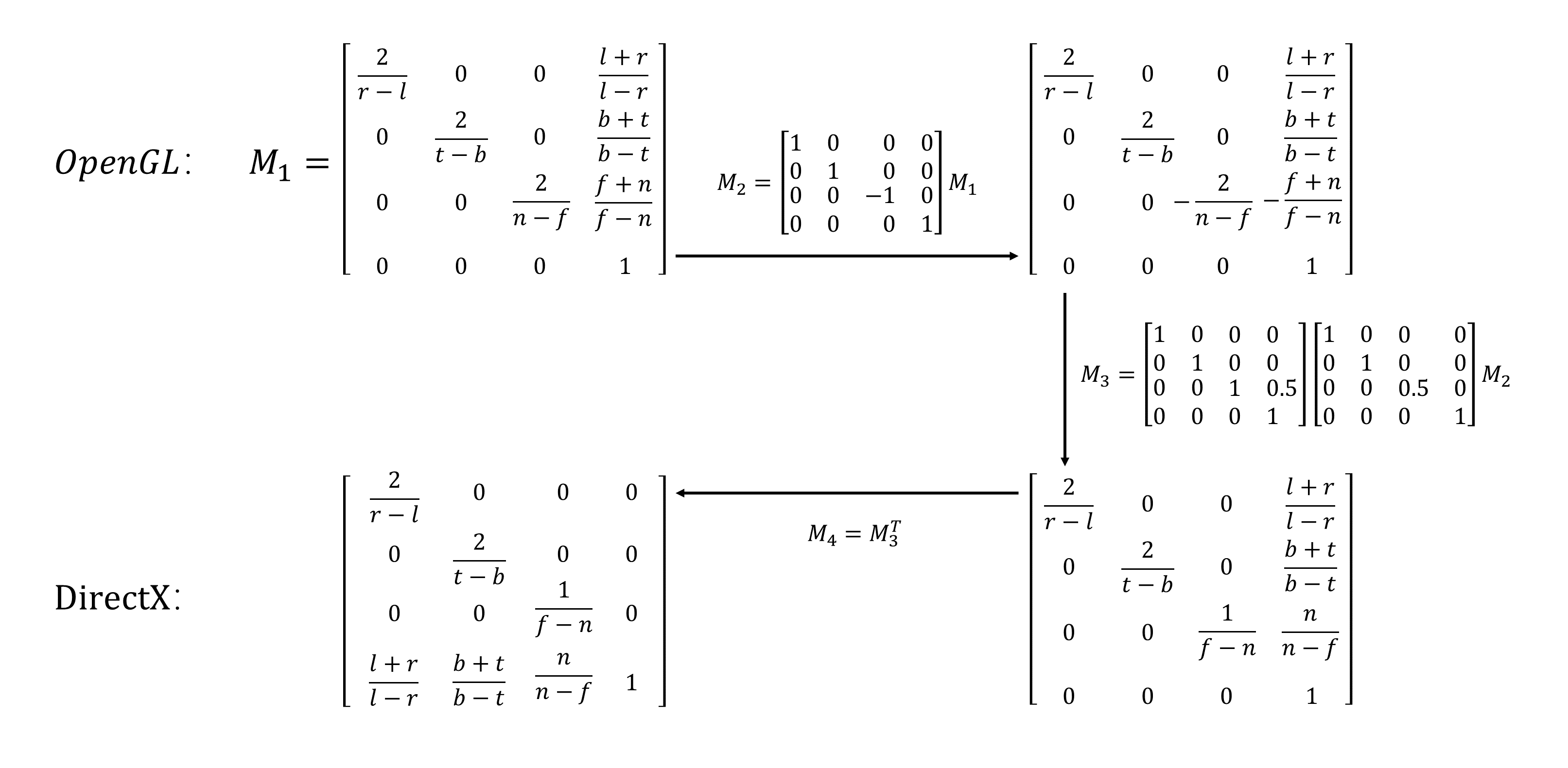
此处以OpenGL的约定为例，遵循右手坐标系，且使用列向量。注意到：由于相机的朝向是-z方向，所以此处的近裁剪面的z坐标n要大于远裁剪面的z坐标f. 先平移再缩放，可以写出以下矩阵：



注意缩放矩阵中s33元素分母是n-f，因为此处n>f，要保证缩放过程中不引入反射，所以此处我的式子与RTR4书上给出的不同。此矩阵即为OpenGL里面使用的正交投影矩阵，若要转化为DirectX里面的，则需要做三点：

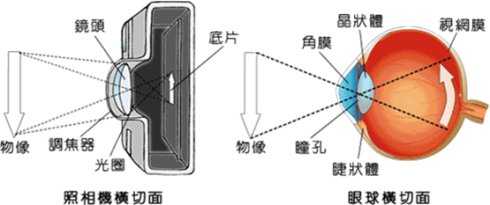
1. 将右手系转换为左手系，即把z变负，即对z坐标进行-1的缩放
2. 把z范围从OpenGL的[-1,1]转到DirectX的[0,1]，此过程可理解为对z先缩放到[-0.5,0.5]再平移到[0,1]，即先缩放0.5倍，再平移0.5个单位
3. 从OpenGL使用的列向量转换到DirectX使用的行向量，故对矩阵转置

上述三个过程可表示为：

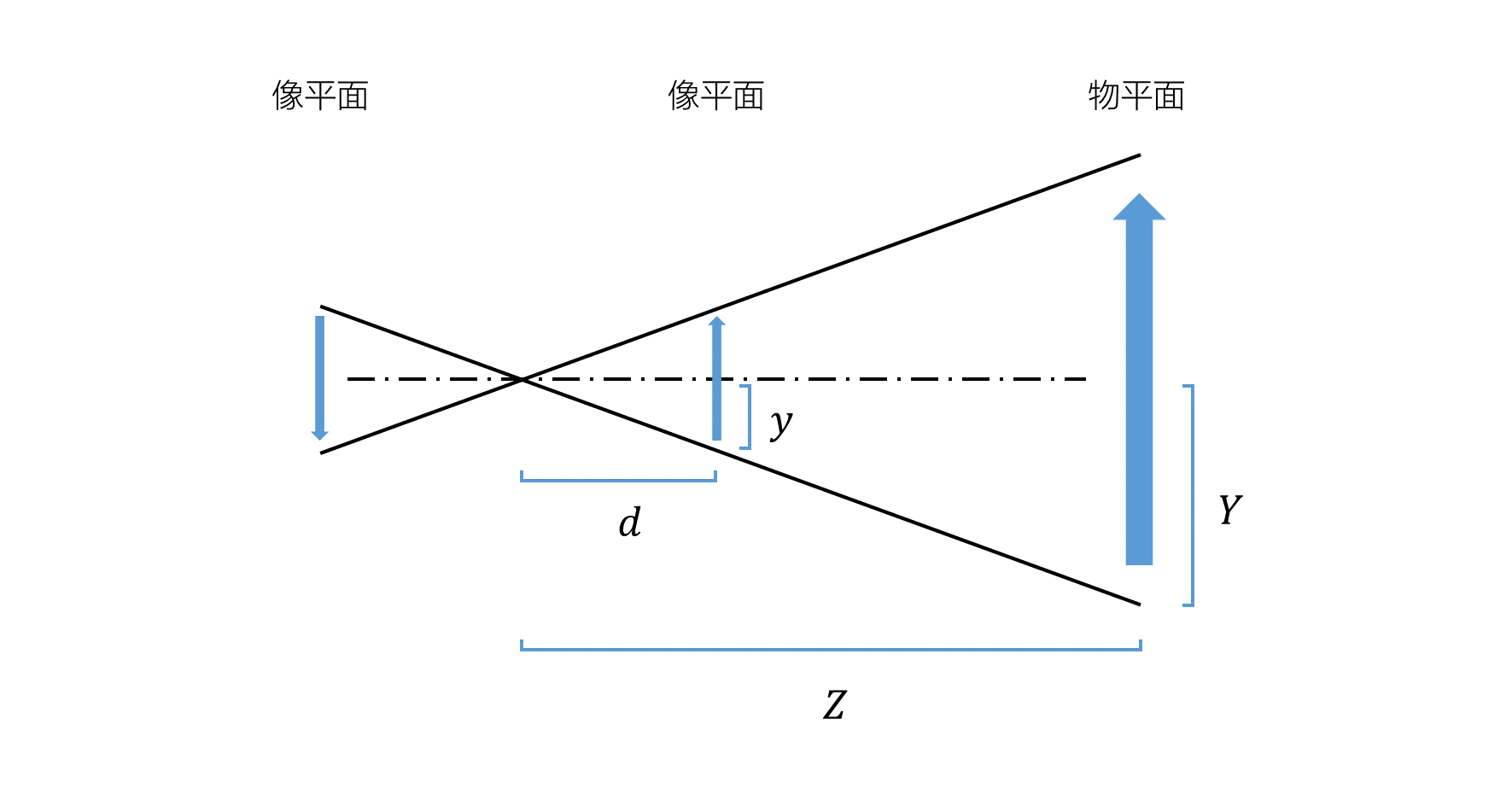


透视投影矩阵：

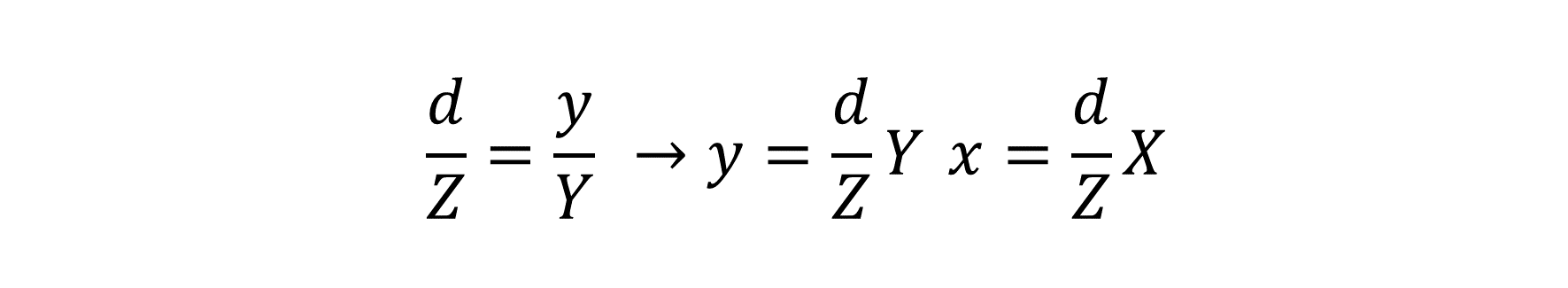
透视投影符合人眼和照相机的成像规律，如图所示：



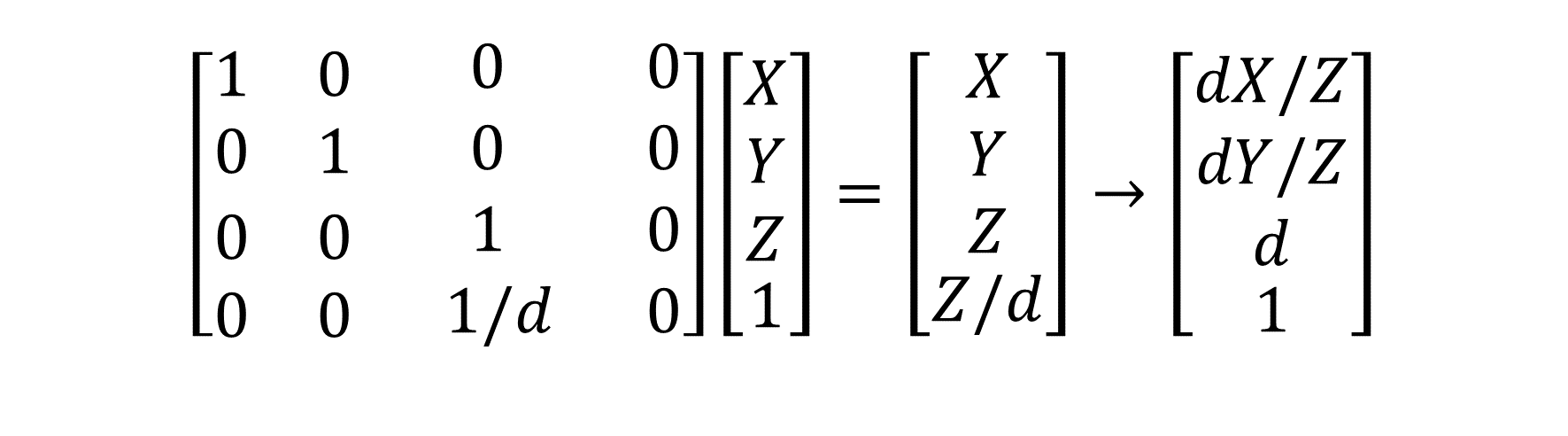
抽象这两种成像过程可得到：（简单起见，把物平面和像平面放在同侧）：



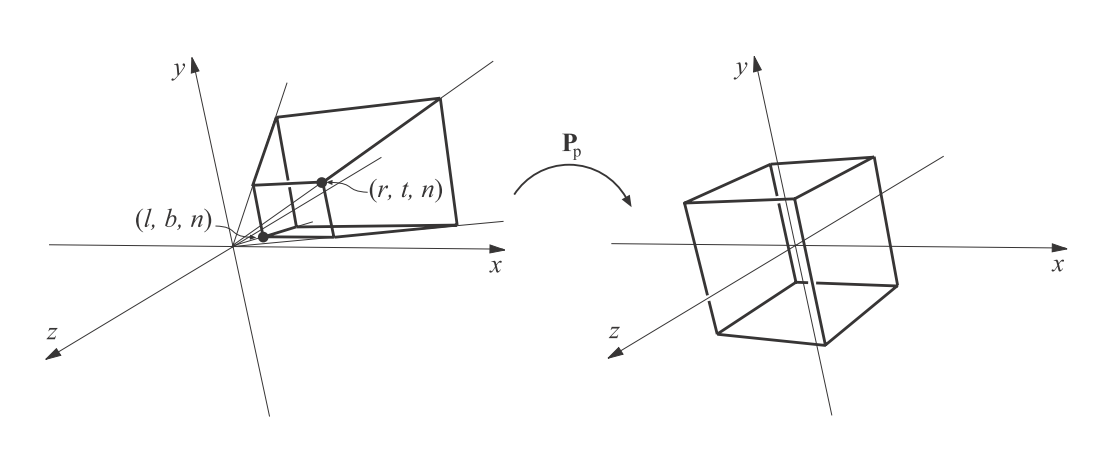
根据相似三角形，可以得到以下式子：



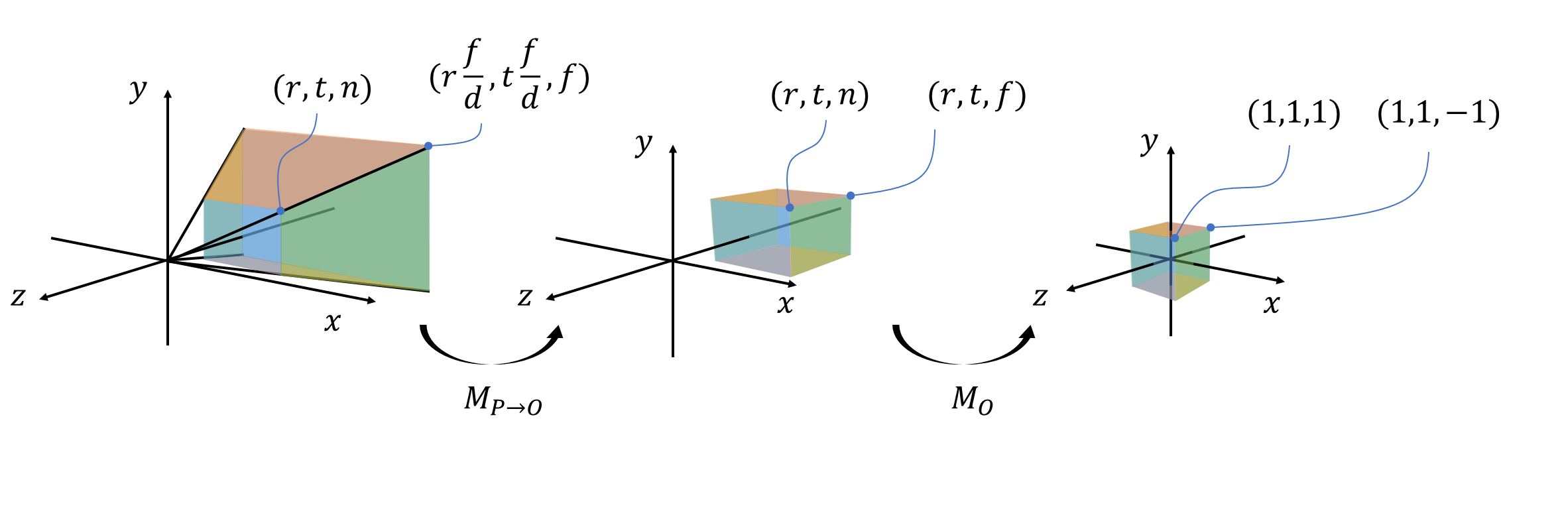
可以看到，此式子并不符合线性变换的两条规则，所以仅用三维的矩阵是不可能表示透视投影变换的，此处就体现了齐次坐标系的优势，考虑以下变换：



但这个矩阵也暴露出了和我们最开始给的简单正交投影矩阵相同的问题：不可逆，丢失了z，没有考虑多个点的重合情况。所以并不可用，一个常用的方法也和之前类似：将一个包围盒转换到规范视域体，但此处的包围盒是一个视锥体(view frustum)，即：

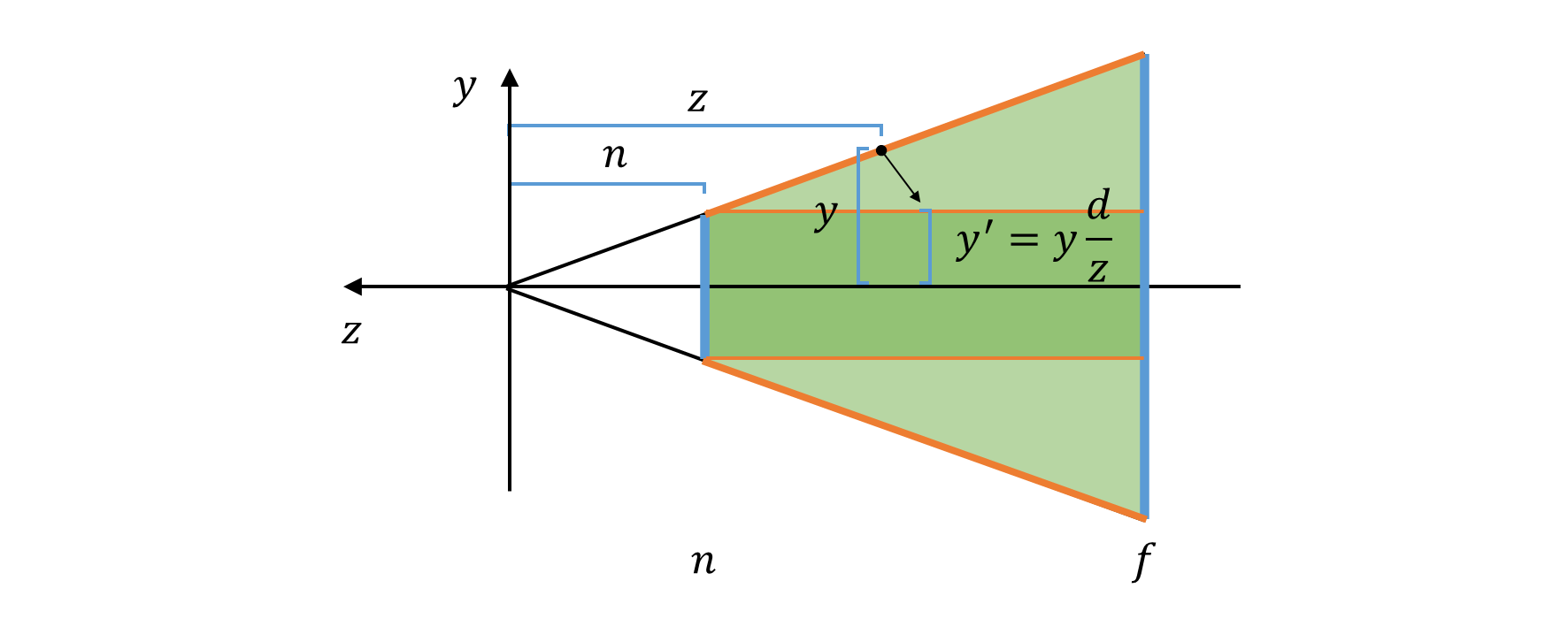


透视投影矩阵的推导较为复杂，此处借鉴闫令琪老师的思想：先将视锥体“原地”变换为AABB包围盒，再经过平移和缩放变换到规范视域体，即：



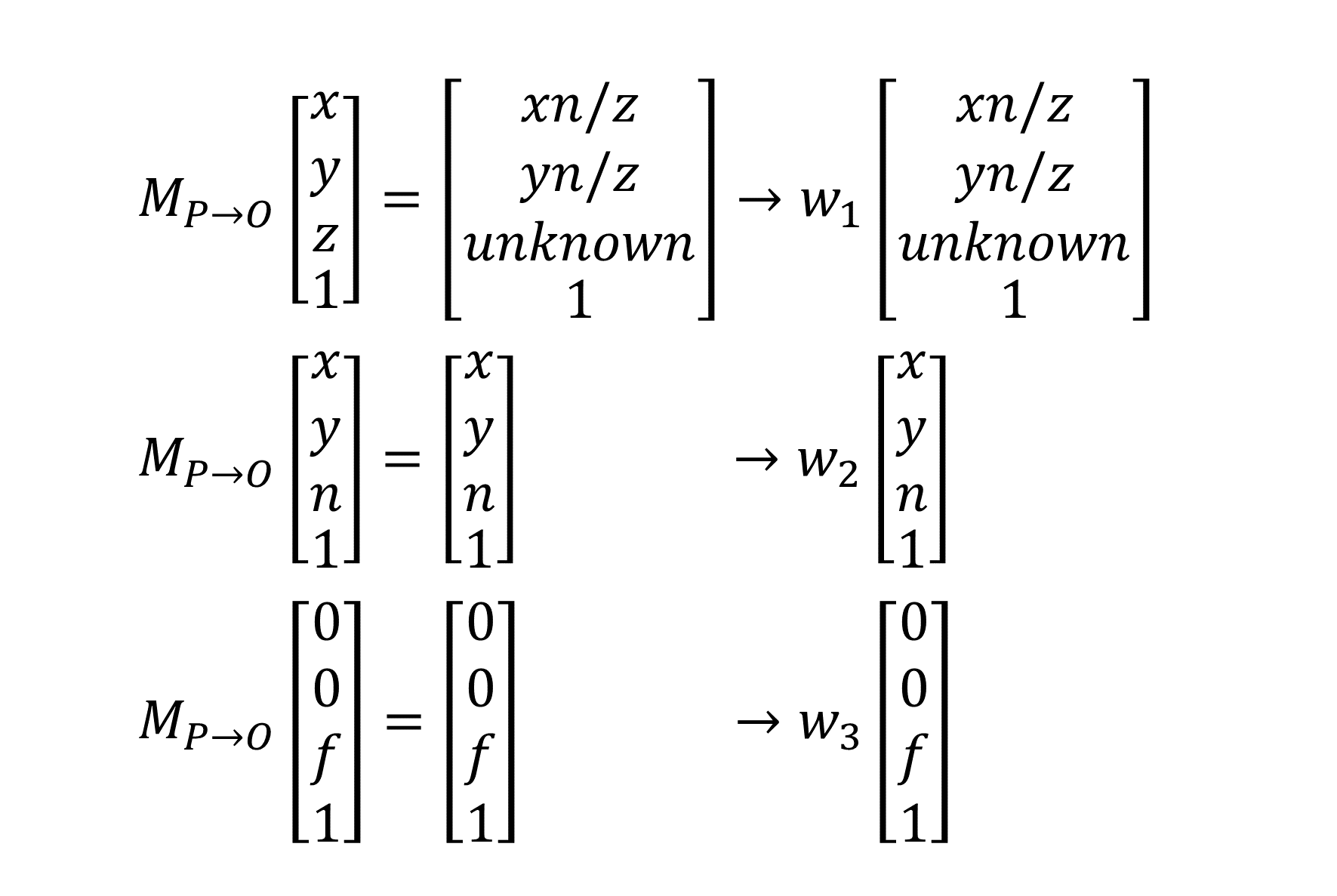
其中第二个矩阵就是前面所述的正交投影矩阵，所以此处推导第一个矩阵，观察这个变换过程，我们可以得到以下几个规律：

1. 视锥体内所有点的xy坐标，都经过了同样的缩放，且缩放因子为n/z，即(x,y,z,1)变换后变成(xn/z,yn/z,unknown,1)，可参照下图理解：



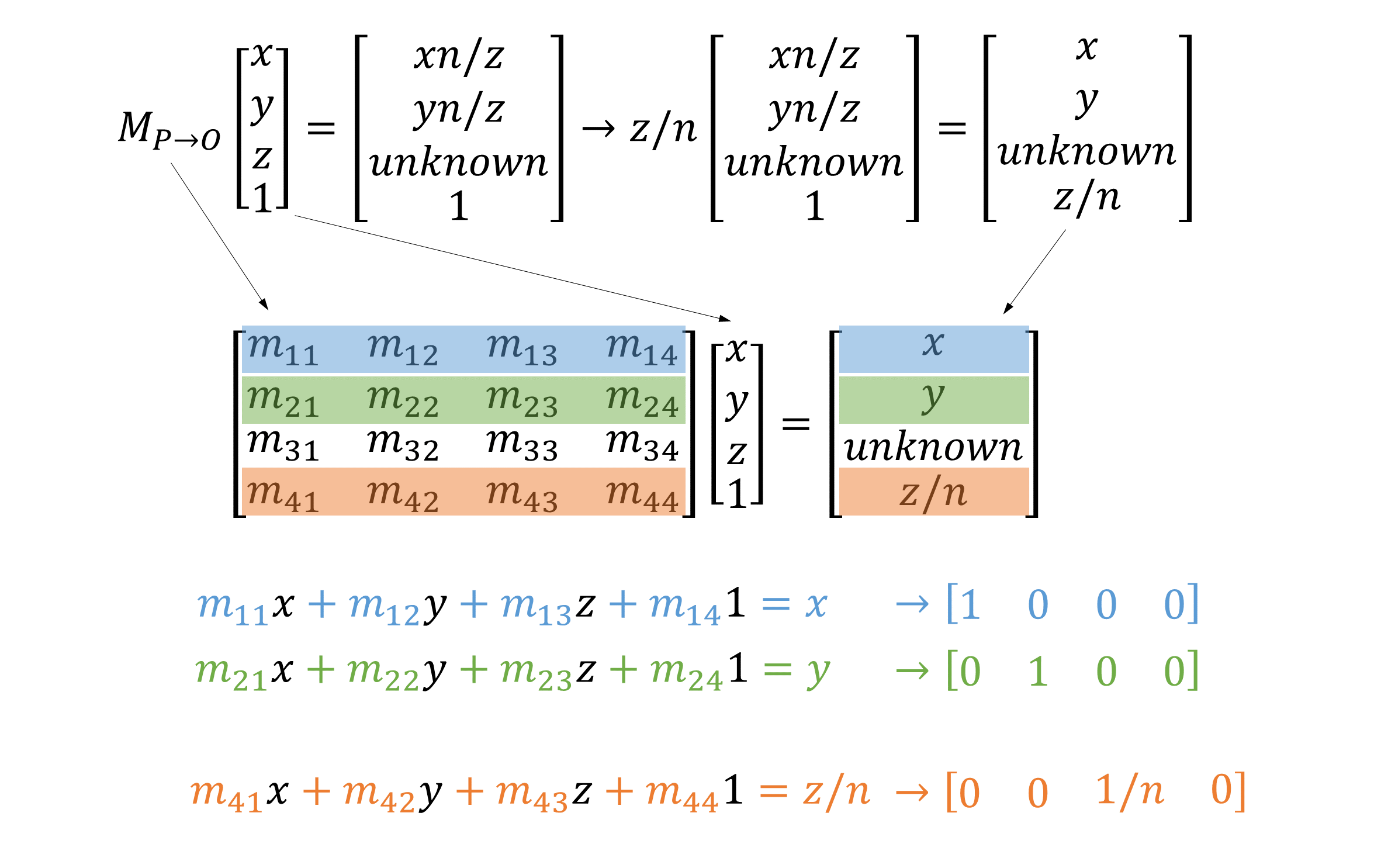
1. 近裁剪平面上所有点的坐标不变，即(x,y,n,1)变换后还是(x,y,n,1)
2. 所有在z轴上的点，变换后仍然在z轴上，特别的，z轴与远裁剪面的交点(0,0,f,1)变换后仍然是(0,0,f,z)

这三个规律可以写成以下式子，考虑到是齐次坐标，所以乘以任意不为0的倍数w都是可以的：

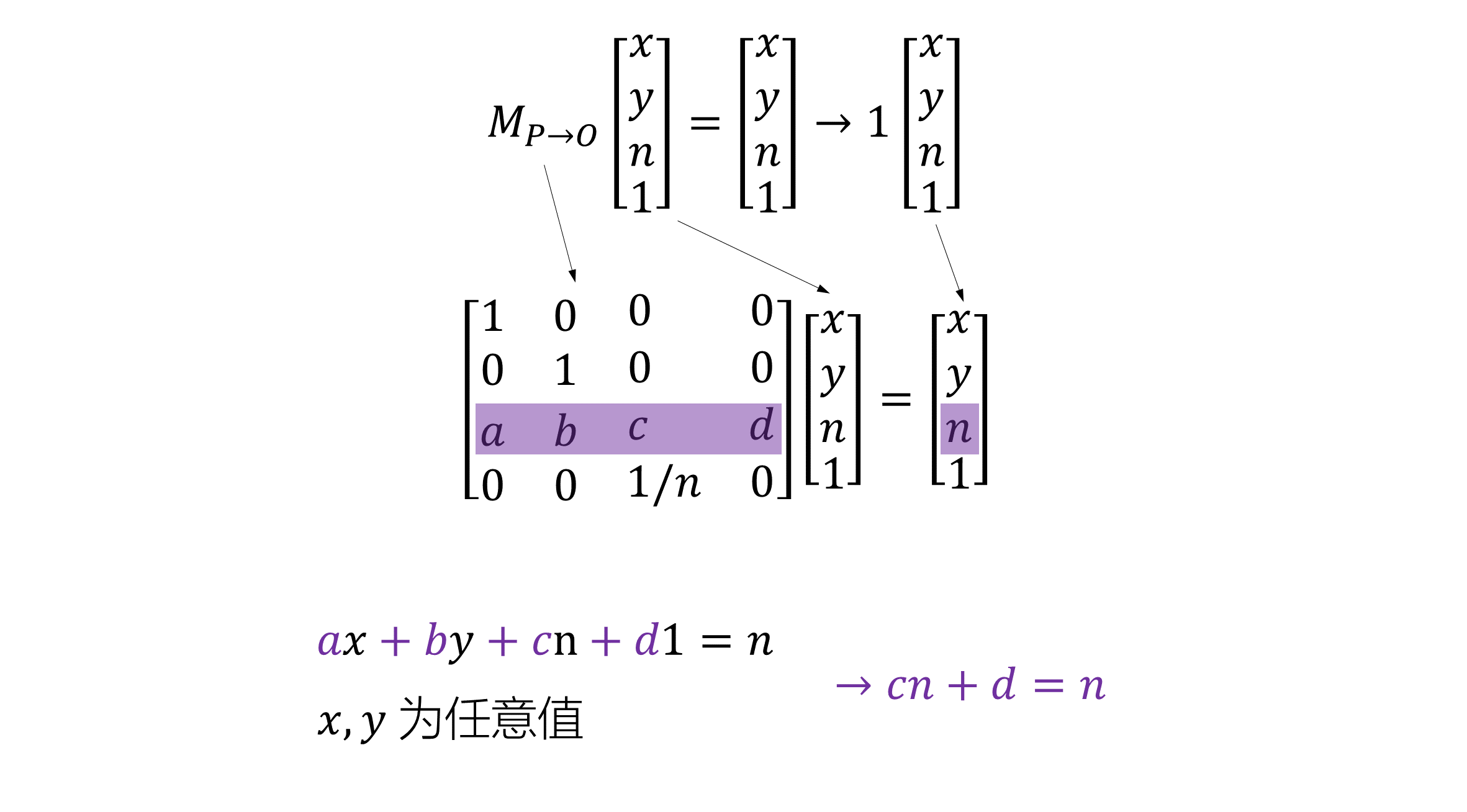


其中矩阵M是关于n和f的未知矩阵，且0>n>f，里面出现的xyz都是视锥体内部任意点，下面来进行推导：

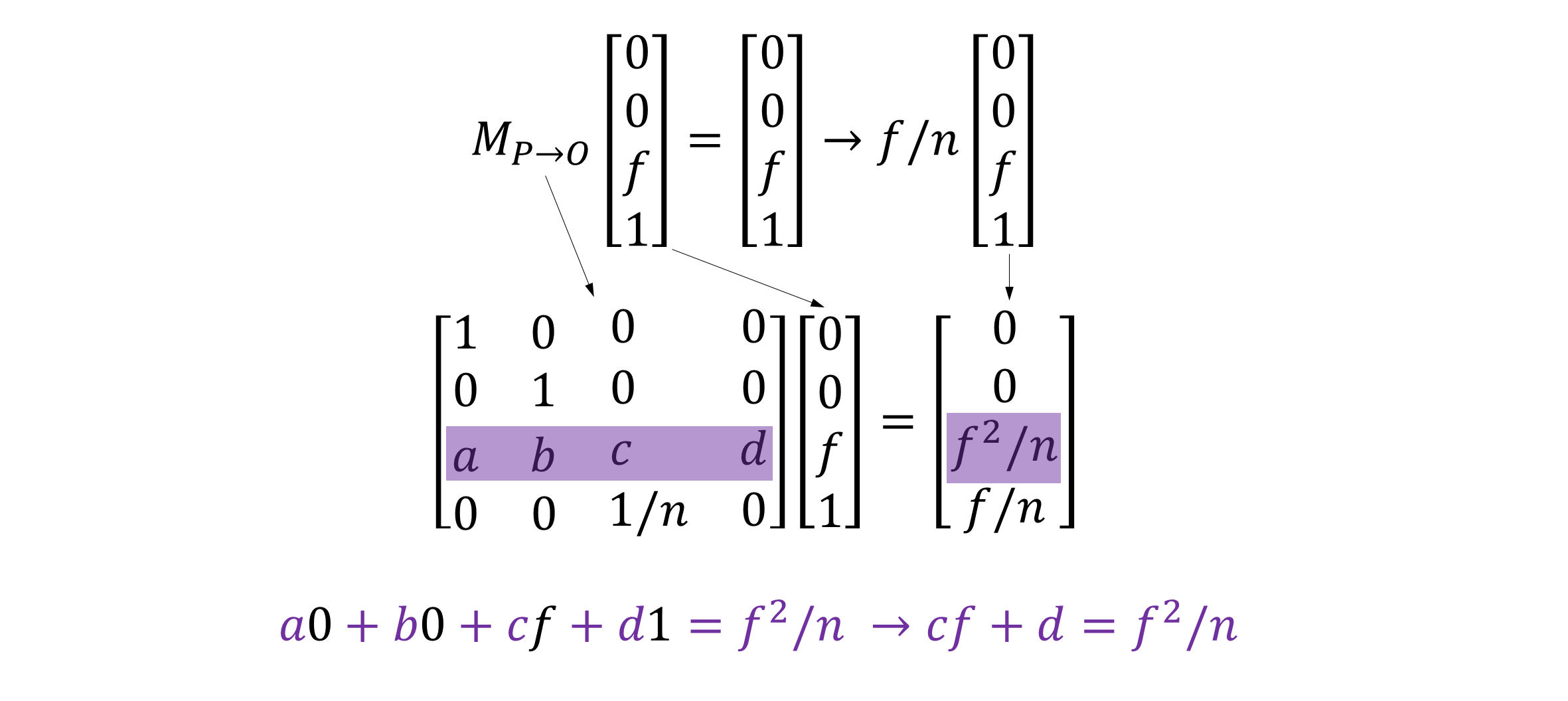
1、首先选取信息量最多的等式一，不失一般性，可以把这个任意的w取为z/n，以便xy凑成前后一样的。可推导出M第一二四行：



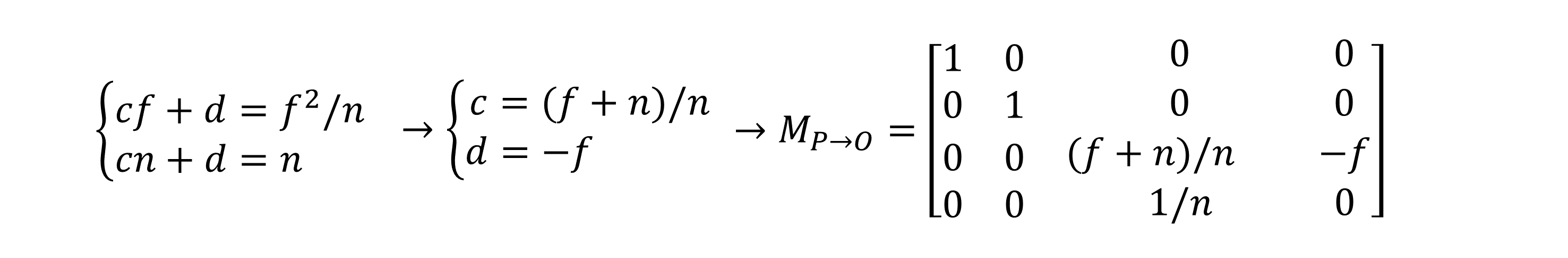
2、再使用等式二，注意此时选择任意因子w的时候，要同时第一二四行也满足关系。发现，w选择1就可以：



1. 最后使用等式三，同样注意此时选择任意因子w的时候，要让第一二四行也满足关系。可以发现，由于第四行与列向量相乘得到f/n，故w因子选择f/n

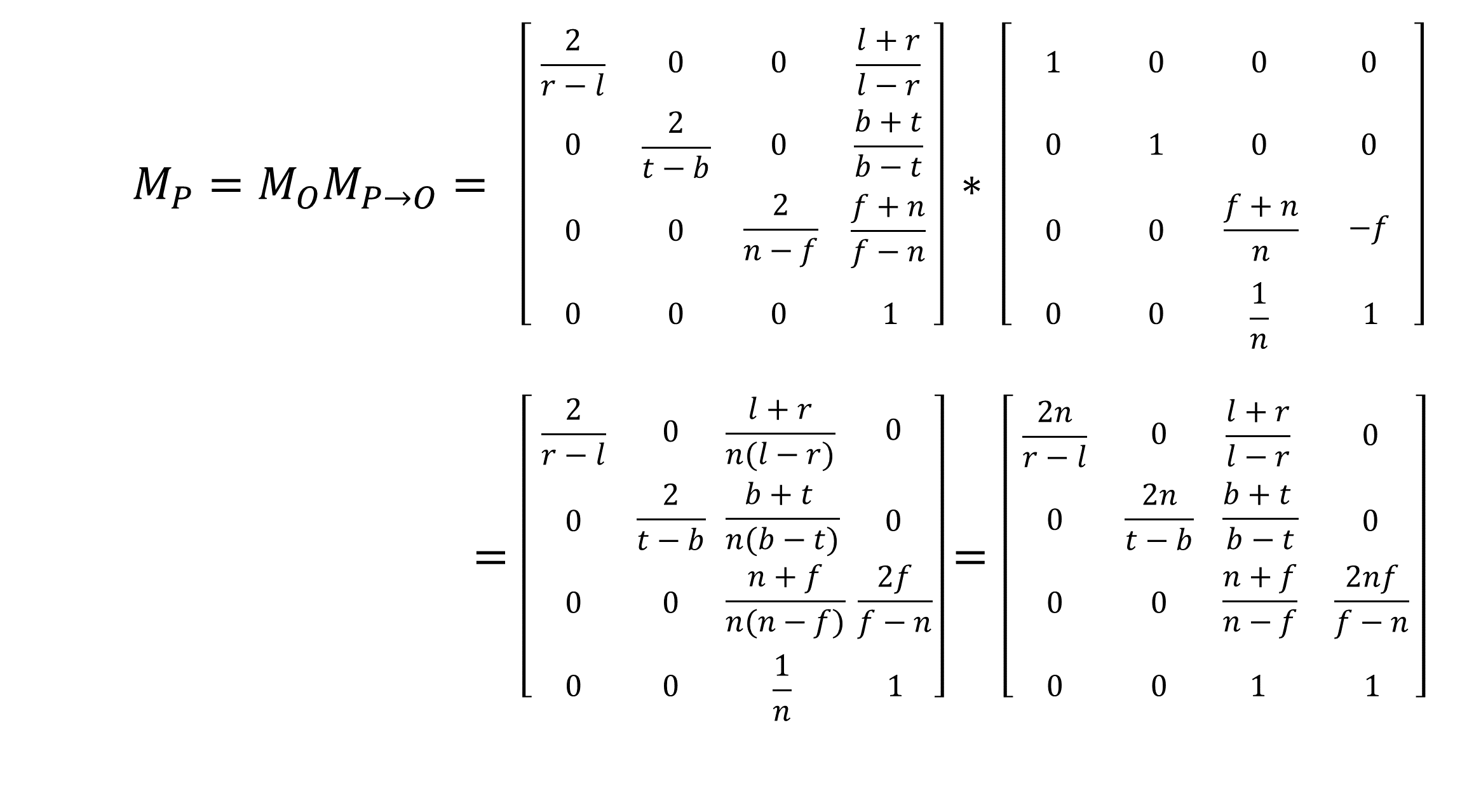


联系第2步和第3步中的结论，可解出c和d，也就解出了整个矩阵：



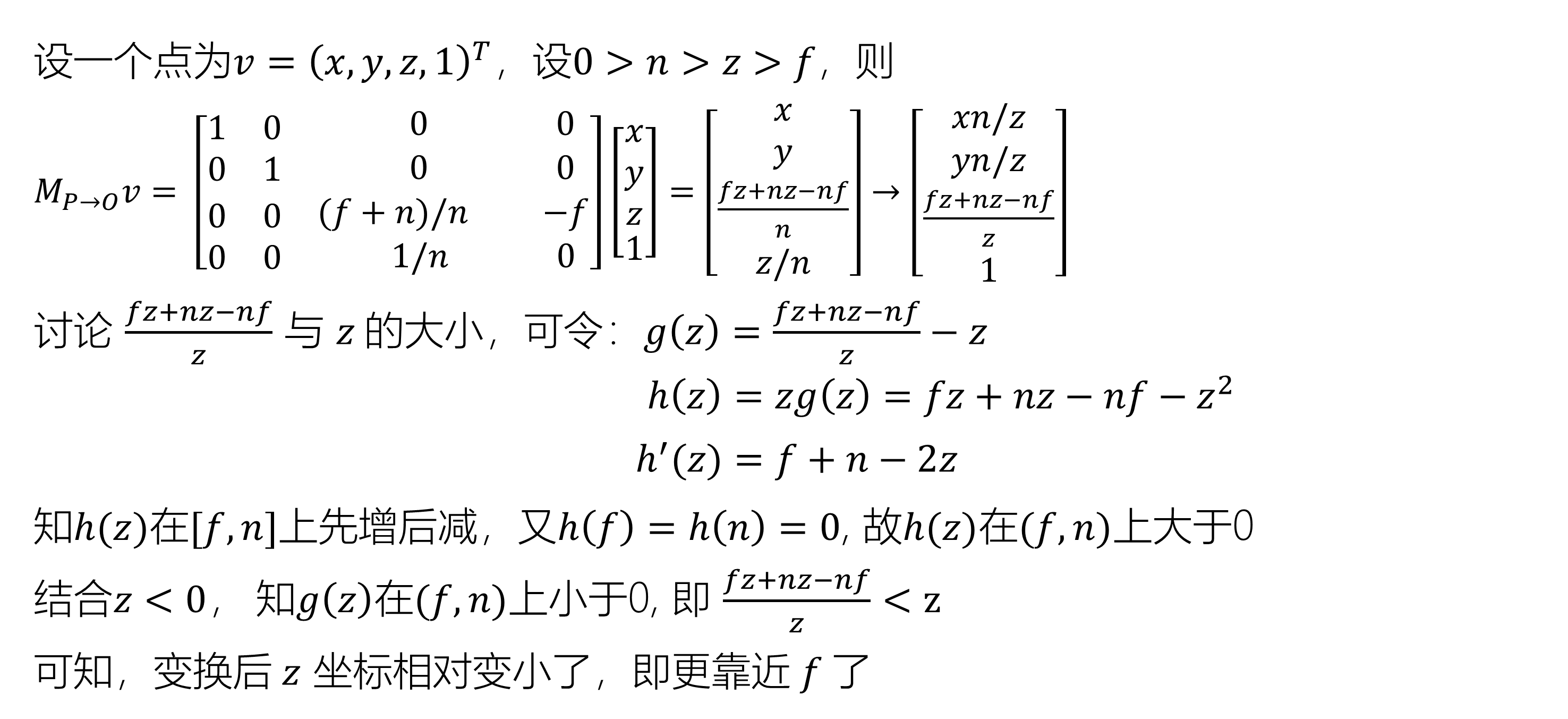
注意此矩阵形式上与闫令琪老师的版本不同，但只是相差了一个倍数而已，最终除以w后可以得到一样的结果。具体差异源自我在处理第一个等式的时候，w取的是z/n，而他取的是z

所以，再结合正交投影矩阵，即可得到整体的透视投影矩阵：



此处给的是OpenGL里面的透视投影矩阵，如果要转化为DirectX里的，按照之前类似正交透视矩阵的三点转换步骤转换即可

关于闫令琪老师在课上提出的问题：在n-f之间的点的z坐标变换后，是更靠近n还是更靠近f了？此处给出自己的一个解析过程以及用matlab进行模拟的结果：



下图给出了用matlab对一系列点进行变换前后的位置关系图，可以看到，视锥体内所有点都往远裁剪面靠，且同一z平面上的xy坐标确实也是同等缩放的（可体现为横向点仍然是等距的）

