上一篇我们学习了一些数学知识，包括矩阵，这一篇我们重点讲矩阵的几何意义：空间变换。

1、变换

变换（transform），指的是我们把一些数据，如点、方向矢量、甚至是颜色等，通过某种方式进行转换的过程。

1）线性变换和仿射变换

能满足下面公式的变换就是线性变换：

IMG_256

缩放和旋转都是线性变换，对于线性变换，一个3x3的矩阵就可以表示对一个三维矢量的线性变换。

平移不是线性变换，比如 f(x) = x + (1, 2, 3)，如果我们令x = (1, 1, 1)，那么

f(x) + f(x) = (4, 6, 8) f(x + x) = (3, 4, 5)

显然不符合上面的公式，所以不能用3x3的矩阵表示一个平移变换，那怎么办，所以有了仿射变换。仿射变换就是合并线性变换和平移变换的变换类型。仿射变换用一个4x4的矩阵来表示，为此，需要把矢量扩展到四维空间，这就是齐次坐标空间。

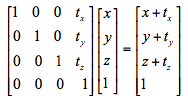


2）齐次坐标

3x3的矩阵不能表示平移操作，我们扩展到了4x4的矩阵。为此，我们还需要把原来的三维矢量换成四维矢量，这个四维矢量就是齐次坐标。那如何把三维矢量转换成齐次坐标呢，很简单，对于点，把w分量也就是第四维设为1，对于方向矢量，w分量设为0。也就是对于一个点，我们用4x4矩阵对其平移、旋转、缩放都可以。但对于方向矢量，平移效果没用，因为方向矢量我们只关心其方向和大小，位置没有意义。

3）几种基础变换矩阵

a、平移矩阵

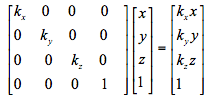


从上面可以看出我们用的是齐次坐标表示法，把点(x, y, z)在空间中平移了个单位。

平移矩阵的逆矩阵就是反向平移得到的矩阵，即：

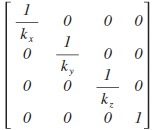
可以看出，平移矩阵不是正交矩阵。

b、缩放矩阵



如果三个缩放系数k都相等，我们称为统一缩放，否则是非统一缩放，因为非统一缩放会拉伸或挤压模型，所以会改变与模型相关的角度和比例，这在法线变换时很重要，如果非统一变换，直接使用变换顶点的变换矩阵就会出错。

缩放矩阵的逆矩阵很容易求解：

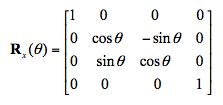


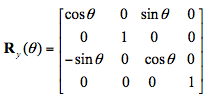
可以看出不是正交矩阵。

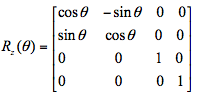
c、旋转矩阵

这里的旋转是围绕空间的x，y，z轴旋转。

绕x，y，z轴旋转的矩阵分别是：







旋转矩阵的逆矩阵是旋转相反角度得到的变换矩阵，你可以试试把上面的用-代替求出的矩阵就是其逆矩阵，然后你会惊奇的发现和其转置矩阵一样，所以旋转矩阵是正交矩阵。

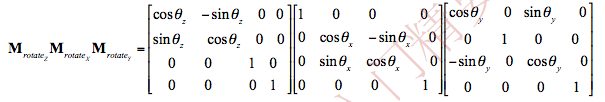
d、复合变换

我们可以把平移、旋转和缩放组合起来，形成一个复杂的变换过程。这个过程可以用下面公式计算：

IMG_256

还记得上一篇最后我们说过我们会采取列矩阵的方式，从右往左逐一变换，即按照缩放，旋转，平移的顺序。注意矩阵乘法是不满足交换律的，所以变换顺序很重要，不同的顺序结果会不一样。在大多数情况下，我们采取上面的顺序。原因自行看书吧，这里不解释。

还有个要注意的是旋转的变换顺序。如果要同时绕三个轴进行旋转，在Unity中，这个旋转顺序是zxy，这意味着组合旋转变换矩阵是：



这里要注意一点，这里的旋转过程中我们的坐标轴不变，不是那种旋转一个轴后由于自身方向的改变再按新的坐标轴的旋转的顺序。

2、坐标空间

通过上一篇和上面有关矩阵变换的相关知识的学习，到现在我们终于进入我们的核心主题了，所有的这一切都是为了坐标空间。

1）坐标空间的变换

在渲染流水线中，我们往往需要把一个点或方向矢量从一个坐标空间转换到另一个坐标空间。而定义一个坐标空间，需要指明原点和3个坐标轴方向。而这些数值实际上是相对另一个坐标空间的。举个例子，我现在坐在屋子的东南角，这是以屋子中心为原点，方位朝向为坐标轴的。但我的屋子在大楼的四层西北方向，那如何以大楼为空间描述我的位置呢。所以坐标空间会有一个层次结构，每个空间都有一个父空间。对坐标空间的变换实际上就是在父空间和子空间之间对点和矢量进行变换。

假设有父空间P和子空间C，我们往往需要把子空间下的点或矢量转换到父空间下表示的，或者反过来，把父空间的点或矢量转换到子空间下的，可以使用下面公式表示这两种需求：

IMG_256

其中表示从子空间变换到父空间的变换矩阵，是其逆矩阵。那么如何求解这些变换矩阵呢，当然，只要求解其一就可，另一个为它的逆矩阵。我们来求。

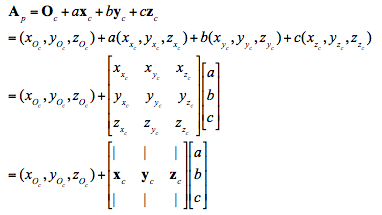
假如我们已知子空间C的3个坐标轴在父空间P下表示为，以及原点，当给定一个子空间下的坐标 =

(a, b, c)，我们求其在父空间的坐标，我们可以得到公式：

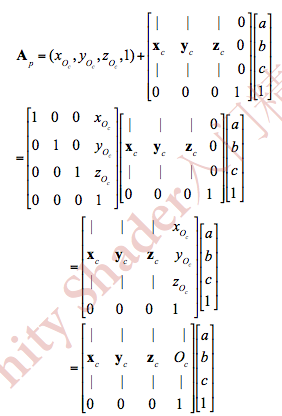
IMG_256

原理很好解释，我们从出发，沿x走了a，沿y走了b，沿z走了c，就得到了其在父空间的坐标。

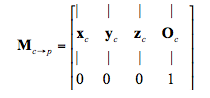
下面就是见证奇迹的时刻了：



最后分别代表他们所在的列，这个公式还存在着加法表达式，即平移变换，还记得平移变换的矩阵表达式吧，3x3的矩阵无法表示平移变换，我们先把他们转换到齐次坐标空间。



第一行到第二行的转变就是平移矩阵的表示方法，所以我们得出了最终结论：

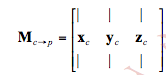


好了，从这个里面我们可以得到相当大的信息量。

首先有了，就是它的逆矩阵；

其次，如果我们知道了这个转换矩阵，那么提取它的前三列就是子空间在父空间下的坐标轴，第四列就是原点。

再次，在对方向矢量的空间变换中，我们不关心其位置所在，所以原点没有意义，三维表达式就够了，所以方向矢量的变换矩阵为：



最后，如果这个转换矩阵是正交矩阵的话，那么其逆就是其转置，那么它的行列分别可以表示其在父子空间的坐标轴了，非常方便。

2）顶点坐标空间变换过程

下面我们要说说，在渲染流水线中，顶点在各个空间的变换过程。

a、模型空间 (model space)

也叫对象空间(object space)或局部空间(local space)。每个模型都有自己独立的坐标空间，当它移动或旋转时，模型空间也会跟着移动和旋转。在模型空间中，我们常使用一些方向概念，例如“前（forward）”、“后（back）”、“左（left）”、“右（right）”、“上（up）”、“下（down）”。Unity在模型空间使用的是左手坐标系，所以+x、+y、+z轴分别对应模型的右、上、前。而模型的右上前是由我们的美工人员制作模型时定好的。

b、世界空间（world space）

相对于模型空间，世界空间是模型所在的最外层的父空间。Unity中，世界空间同样是左手坐标系，原点是游戏空间的中心，x、y、z轴固定不变。顶点变换的第一步，是将顶点从模型空间变换到世界空间。这个变换叫模型变换（model transform）。在Unity中，我们看到模型的Transform组件里的位置旋转和缩放，都是基于它的父节点（parent）的，当模型没有父节点时，那这个Transform就是基于世界空间的。变换过程用下面的公式就可：

IMG_256

有关平移旋转缩放的矩阵上面都介绍过了，把Transform组件的信息代入进去就可求得，这里不多介绍。

c、观察空间（view space）

观察空间也被称为摄像机空间，在观察空间中，摄像机位于原点，它决定了我们渲染游戏所使用的视角。前面说过，观察空间采用的是右手坐标系，所以+z轴指的是摄像机后方。顶点变化的第二步，就是将顶点坐标从世界空间变换到观察空间中。这个变换叫观察变换（view transform）。

为了得到顶点在观察空间的位置，我们可以有两种方法。

一种方法是计算观察空间的三个坐标轴在世界空间的表示，然后按照上面“1）坐标空间的变换”的方法算出观察空间到世界空间的变换矩阵，再求逆得住世界空间到观察空间的变换矩阵。

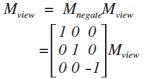
第二种方法是平移整个观察空间，让摄像机原点位于世界空间原点，坐标轴与世界空间坐标轴重合。两种方法得到的变换矩阵是一样的。

这里我们用第二种方法，有一点很重要，我们上面说过世界空间的变换顺序公式是先缩放，再旋转，再平移，而这里我们为了把摄像机移回世界坐标原点，我们需要逆向变换，所以是先平移，再旋转，再缩放。

IMG_256

注意我们是从右往左，矩阵乘法满足结合不满足交换，所以乘法顺序可以从左往右。这里可以把摄像机的Transform组件数据逐一代入。

还有一点要注意，因为观察空间是右手坐标系，与世界空间的左手坐标系z轴相反，所以z分量要取反操作：



最后乘以经过上一步世界变换的位置就可以了

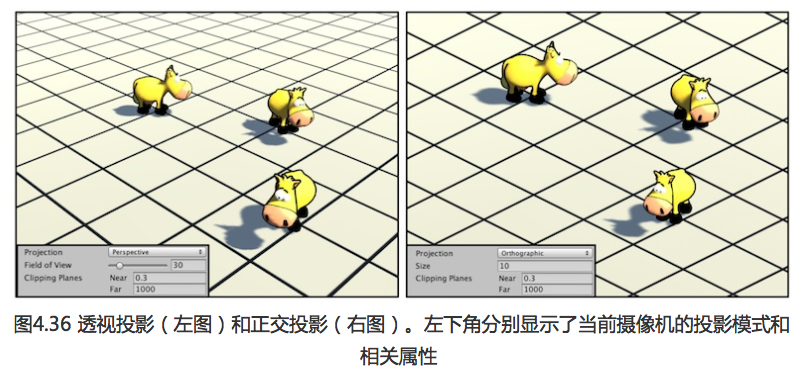
IMG_256

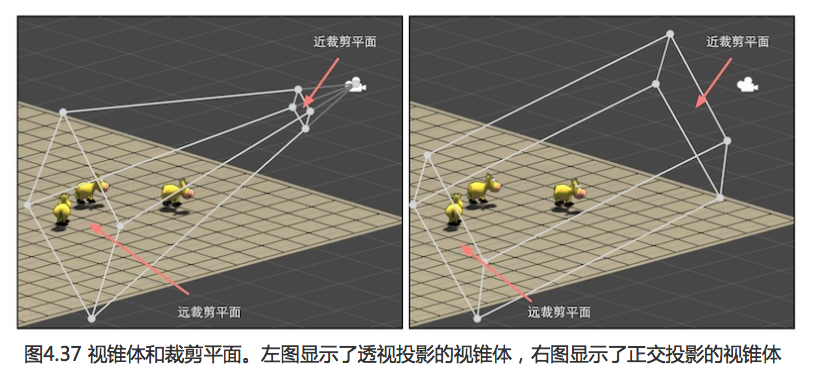
书中有关于一个农场的例子，有详细的计算过程，有兴趣的可以按着算一遍。

d、裁剪空间（clip space）

几种空间转换中最复杂的一个空间。裁剪空间，也被称为齐次裁剪空间，用于变换的矩阵叫裁剪矩阵，也叫投影矩阵。在渲染流水线中说过裁剪，完全位于裁剪空间的被保留，完全位于外面的被剔除，与这个空间相交的会被裁剪。那么这个空间是如何决定的，答案是视锥体（view frustum）。

视锥体决定了摄像机可以看到的空间。视锥体由六个面组成，这些面也叫裁剪平面。视锥体有两种类型，涉及两种投影，正交投影（orthographic projection）和透视投影（perspective projection）。这里用一下书上的图：





可以看出，透视投影模拟了人眼看世界的方式，适合3D游戏，而正交投影则完全保留了物体的距离和角度，适合2D游戏。

在上图中我们可以看到两个特殊的面，是近裁剪平面和远裁剪平面。他们决定了摄像机可以看到的深度范围。和侧面的四个面决定了裁剪空间。如果直接用视锥体定义的空间来进行裁剪，那么不同的视锥体要不同的处理，而且透视投影的视锥体判断起来更麻烦。所以我们需要一种更通用的方式，通过一个投影矩阵把顶点转移到一个裁剪空间中。

投影矩阵有两个目的：

第一，为投影做准备

投影矩阵并没有进行真正的投影工作，投影是个降维的过程，从三维降到二维，真正的投影发生在后面的屏幕映射中，通过齐次除法获得二维坐标。经过矩阵变换后，顶点的w分量会有特殊的意义。

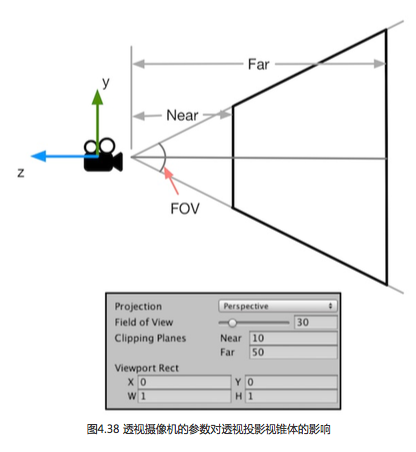
第二，对x、y、z分量进行缩放。

直接用视锥体的6个裁剪平面进行裁剪比较麻烦。经过投影矩阵缩放后，w分量会成为一个范围值，如果x、y、z都在这个范围内，就说明顶点位于裁剪空间中。

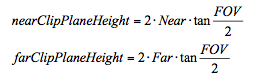
下面，我们分别看看两种投影类型的投影矩阵：

一、透视投影

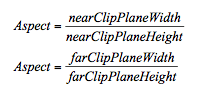
我们先看看透视投影的6个裁剪平面怎么决定的。在Unity中，它们由Camera组件中的参数和Game视图的纵横比共同决定。如图所示：



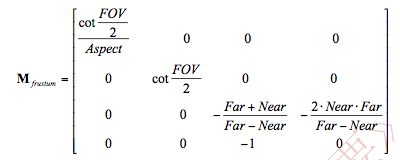
上图中Camera的Field of View(FOV)决定视锥体竖直方向的张开角度，Clipping Planes中的Near和Far决定视锥体的近裁剪平面和远裁剪平面距离摄像机的远近，这样就可以求出近和远裁剪平面的高度：



而横向信息由摄像机的纵横比决定。这个纵横比由Game视图的纵横比和Viewport Rect中的W和H属性共同决定（Unity中可以通过Camera.aspect获得）。假设纵横比为Aspect，则：

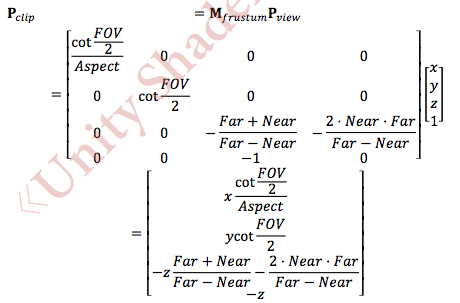


这样可以确定透视投影的投影矩阵：

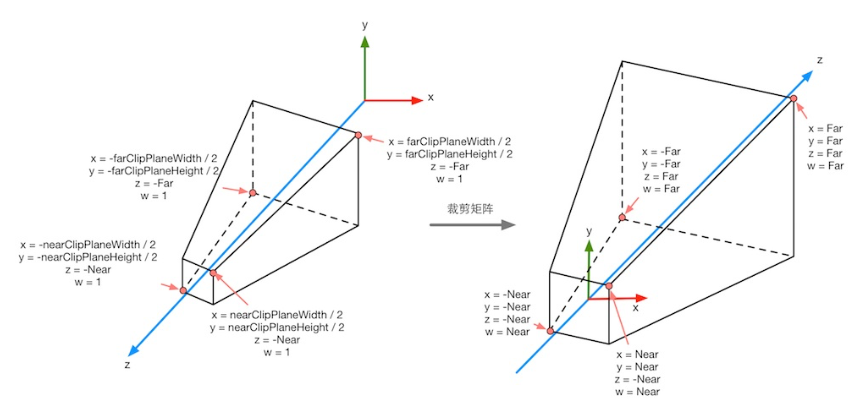


推导过程看书上的扩展阅读部分。这个投影矩阵是建立在Unity坐标系上，观察空间是右手坐标系，使用列矩阵右侧相乘，且变换后z分量在[-w, w]之间。但在DirectX中，z分量在[0, w]之间，上面的透视矩阵就要更改了。这里不讨论。

用上一步观察空间得到的坐标和投影矩阵相乘，就可以变换到裁剪空间中：



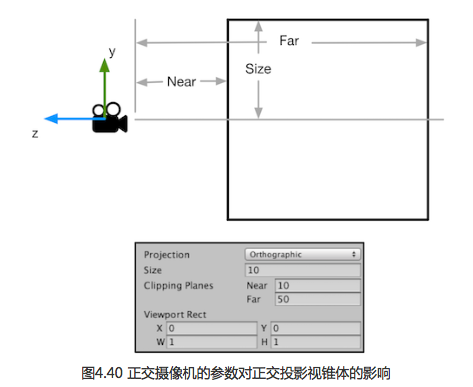
就如上面说过的，本质就是对x、y、z做了不同的缩放（z还有个平移）。w也不再是1，而是z取反。最后通过x、y、z是否在[-w, w]中判断是否位于视锥体内。不在其内的会被剔除或裁剪，这样通过投影矩阵后，视锥体变化如下：



读者可以把左边的各个顶点带入上面的公式，就会得到右边顶点，而且我们发现，裁剪矩阵使空间从右手坐标系换到了左手坐标系，z从向外为正变成了向里为正。

二、正交投影

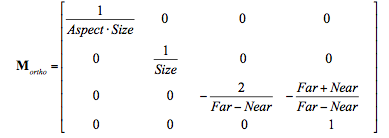
和透视投影类似，我们看看其Camera组件的属性：



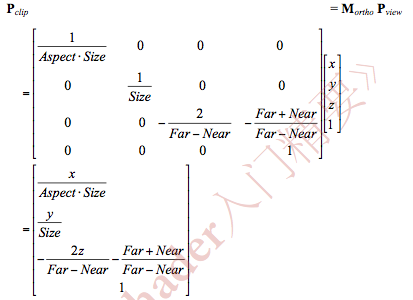
视锥体是个长方体，因此不需要FOV了，用Size代替了，Size是高度的一半。因此，我们得到公式：

IMG_256

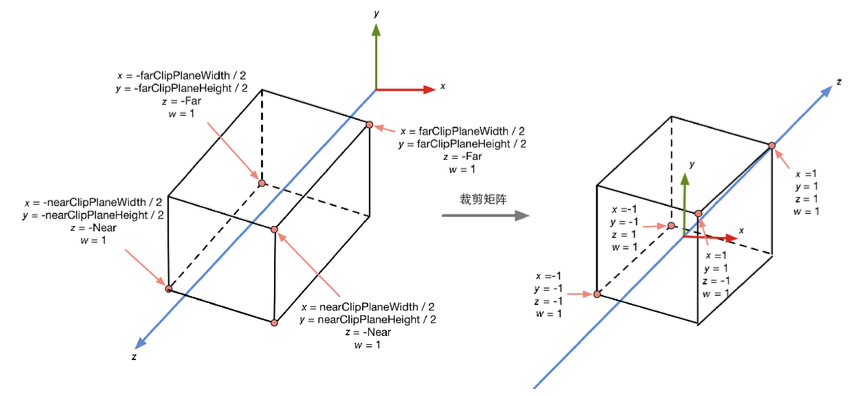
Aspect是横纵比。这样，可以得到正交投影的裁剪矩阵，如下：



然后观察空间的顶点与矩阵相乘，如下：



可以看出，w分量依然为1。判断是否位于裁剪空间内与透视投影一样，x、y、z是否在[-w, w]之间。通过投影矩阵后，视锥体变化如下：

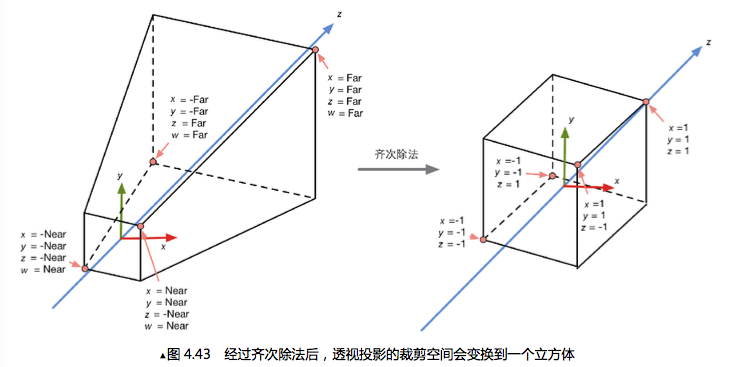


可以看出，变换后空间从长方体变成正方体了，范围是[-1, 1]。

e、屏幕空间（screen space）

终于说完裁剪空间了，经过投影矩阵变换后，我们完成了裁剪工作，开始正式投影了，把视锥体投影到屏幕空间。屏幕空间是个二维空间，投影的过程分为两步：

首先，要进行齐次除法，也被称为透视除法。就是用x、y、z分量除以w分量。在OpenGL中，这一步得到的坐标叫归一化的设备坐标（NDC，Normalized Device Coordinates）。经过这一步，我们把坐标从齐次裁剪空间转换到NDC中，你会惊奇的发现，这样会使透视投影的类似金字塔形状的空间变成正方体，并且和正交投影的一样：



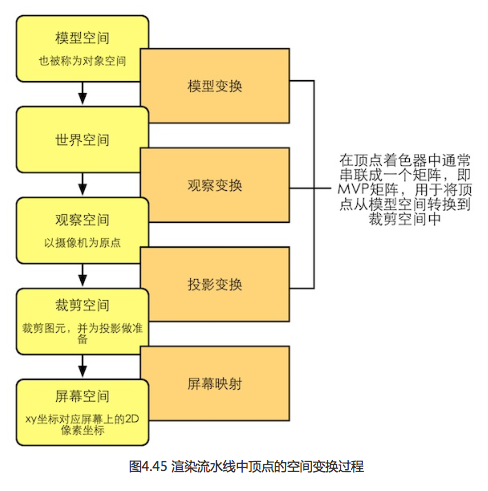
推导过程很容易，回头看裁剪空间那里的公式，透视投影坐标经裁剪矩阵变换后w是-z，所以坐标x、y、z都除以-z就得到了右边的样子。而正交投影变换后w是1，所以除以1没变化，这样两种投影方式就都是一样的正方体了。

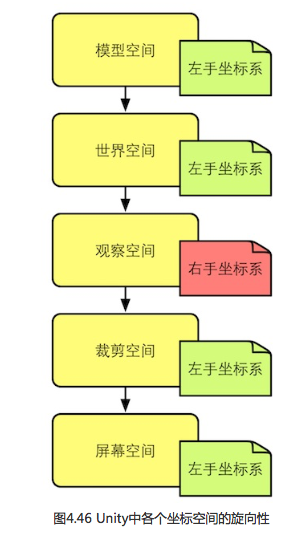
现在，我们开始屏幕映射了。Unity左下角坐标是(0, 0)，右上角是(pixelWidth, pixelHeight)，现在经过齐次除法后x、y的范围是[-1, 1]，所以这个过程就是个缩放的过程。首先把x、y变到[0, 1]，很简单，比如x = (x + 1) / 2，然后再乘以pixelWidth就是映射后的x了，当然这里的x，y都是裁剪空间的坐标除以w，总公式如下：

x、y被用作投影了，z分量会被用于深度缓冲，传统方式是z/w直接存进深度缓冲，但这不是必须的，驱动生产商会根据硬件来选择最好的存储格式。

f、总结

以上就是一个顶点如何从模型空间变换到屏幕空间的过程。顶点着色器的最基本的任务就是把顶点坐标从模型空间转换到裁剪空间中。也就是前三个顶点变换过程。后面一系列变换是自动完成的。然后在片元着色器中，我们就可以得到该片元在屏幕空间的像素位置了。最后上一张图总结一下：





---------------------

作者：Aimar\_Johnny

来源：CSDN

原文：https://blog.csdn.net/lzhq1982/article/details/73612170

版权声明：本文为博主原创文章，转载请附上博文链接！