课4

课4：透视投影

目标

上一堂课里我们通过简单的忘记Z轴的方法用正交投影来渲染我们的模型。而今天的目标是学习怎么用透视投影的办法来进行渲染：

2D 几何学

线性变换

一个在平面上的线性变换可以用矩阵表示。如果我们有一个点(X,Y)这个点的位移可以表示为：

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

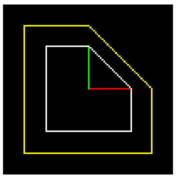
最简单的（而不是退化）移动是恒等式，它不移动任何点：

$$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}x\\y\end{bmatrix}$$

矩阵的对角能对坐标轴进行缩放。让我们来距离说明，如果我们进行如下的位移：

$$\begin{bmatrix}3/2 & 0\\0 & 3/2\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}3/2x\\3/2y\end{bmatrix}$$

下图中白色的对象（切掉一个角的白色矩形）将会转变为黄色那个样子。红色和绿色的线段分别表示一个单位向量的X.Y长度：



本文中运用的图片都能在这些代码文件里面找到

我们为什么会因为矩阵而烦恼？因为它很难处理。首先，在一个矩阵中我们可以对表达式做这样的转变：

$$\begin{bmatrix}3/2 & 0\\0 & 3/2\end{bmatrix}\begin{bmatrix}-1 & 1 & 1 & 0 & -1\\-1 & -1 & 0 & 1 & 1\end{bmatrix}=\begin{bmatrix}-3/2 & 3/2 & 3/2 & 0 & -3/2\\-3/2 & -3/2 & 0 & 3/2 & 3/2\end{bmatrix}$$

这个矩阵表达和前一个很像，只不过用了2X5的矩阵来表示我们的矩形对象。我们用一个简单数组来表示所有的顶点，用变换矩阵乘以自身，得到变换后的对象。这难道不酷么？

真正的原因在这：我们总是需要用连续多次的位移来得到转换后我们想得到的对象。想象一下你的位移源码写成如下这样：

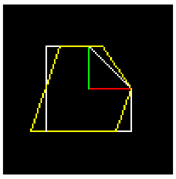
```
vec2 foo(vec2 p) return vec2(ax+by, cx+dy);
vec2 bar(vec2 p) return vec2(ex+fy, gx+hy);
[..]
for (each p in object) {
    p = foo(bar(p));
}
```

代码的意思是对所有的顶点进行两次线性变换，而这些变换我们计算起来通常是上百万的量级的。而一般来说连续几十次这样的变换也不是什么稀罕事，而这种上千万级的计算量太耗性能了。在矩阵计算中，我们可以预乘以所有的位移变换最后在对我们的对象进行一次操作。我们能够将只有乘法表达式的方程的括号放在我们想要放的地方么？

好的，让我们继续。我们知道矩阵对角的系数能够对我们的世界坐标进行缩放。那么其他的系数能够干什么呢？让我们考虑一下如下的矩阵运算：

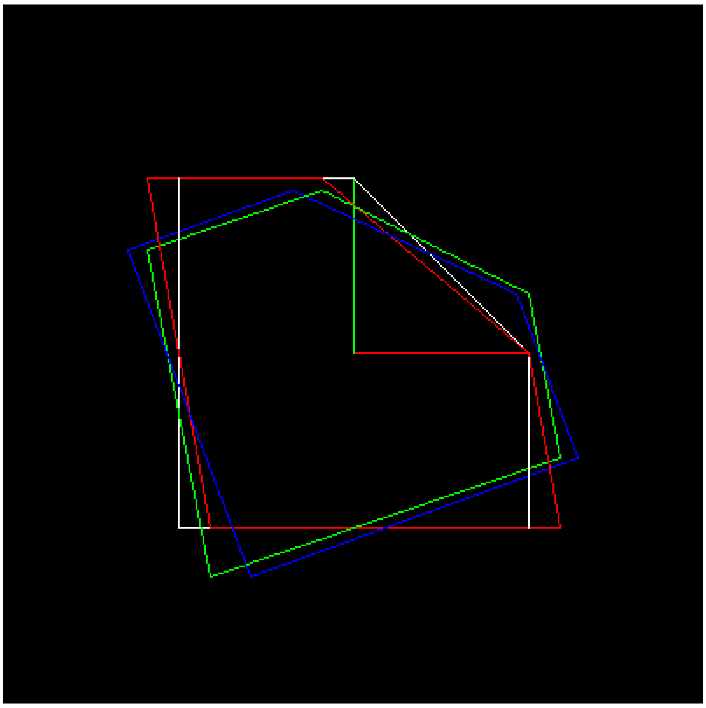
$$\begin{bmatrix}1 & 1/3\\0 & 1\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}x+y/3\\y\end{bmatrix}$$

如下是我们对象的转换后的样子：



这是简单的对X轴进行了裁剪。另一个反对角的矩阵在Y轴进行剪切。因此，有两种不同的基本线性变换在这个平面上：缩放和裁剪。很多读者就会问了，等下，那么旋转呢？

事实证明，任意一个渲染（围绕原点的）都可以表示为三个裁剪的共同作用，如下白色的对象转变到红色的，然后转变为绿色的，最后变成蓝色的。



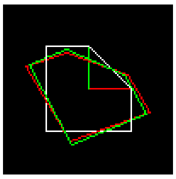
而这些都是一些错综复杂的细节，为了让事情变得简单，一个旋转矩阵能直接写出来（还记得之前的直接相乘的方程式么？）：

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos(\alpha) - y\sin(\alpha) \\ x\sin(\alpha) + y\cos(\alpha) \end{bmatrix}$$

我们可以用任意的顺序乘以矩阵，但是请记住，矩阵的乘法不满足交换律：

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \times \begin{bmatrix} 1 & 1/3 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 1/3\cos(\alpha) - \sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) + 1/3\sin(\alpha) \end{bmatrix} \neq \begin{bmatrix} 1 & 1/3 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} = \begin{bmatrix} \cos(\alpha) + 1/3\sin(\alpha) & 1/3\cos(\alpha) - \sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

这就像是：先裁剪一个对象再旋转它和先旋转它再裁剪是不同的两个概念！



2D 仿射变换

所以，任意一个在平面上的线性变换都可以归纳于缩放和裁剪。这意味着我们想做的任何线性变换，原点是永远不会移动的。如果我们不能简单的表示转换，那么计算过程就会变得很恶心了，这是很可能发生的情况，我们能让它变成这样吗？如果一个位移不是线性的，没有关系，我们可以先增添它的线性部分：

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

这个表达式太酷了，我们可以旋转，缩放，裁剪和位移。然而，我们让我们回想一下我们对复杂转换方程的兴趣点。如下是一个两次位移的复杂计算（记住，我们需要编写几十次这样的计算）：

$$\begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} \left(\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_1 \\ f_1 \end{bmatrix} \right) + \begin{bmatrix} e_2 \\ f_2 \end{bmatrix}$$

这看起来也太丑了，而且大概率会出错。

齐次坐标

好的，是时候展现一下黑魔法了。想象一下我添加一行一列到我们原本的位移矩阵中（让它变成3X3）并添加一个增加了值1的坐标轴向量到我们的位移矩阵中：

$$\begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \\ 1 \end{bmatrix}$$

如果我们将这个矩阵和这个增加了1值的向量相乘我们就会得到另一个带有1值的向量，但是另外两部分正是我们想要的参数，多神奇啊。
实际上，这个点子非常简单。在2维空间中平行移动不是一个线性的过程。故此我们需要将2维空间坐标转换到3维空间中（通过简单添加一个单位向量轴）。这意味着我们将2维的空间投影到了一个Z轴=1的三维平面上。这样我们就能在3维空间中进行线性位移并实际作用到我们的2维物理平面上。平行位移没有变成线性的，但是位移的过程变得简单了。
那么。我们怎么把3维的平面投影到2维平面上呢？简单除以一下第三个参数就好：

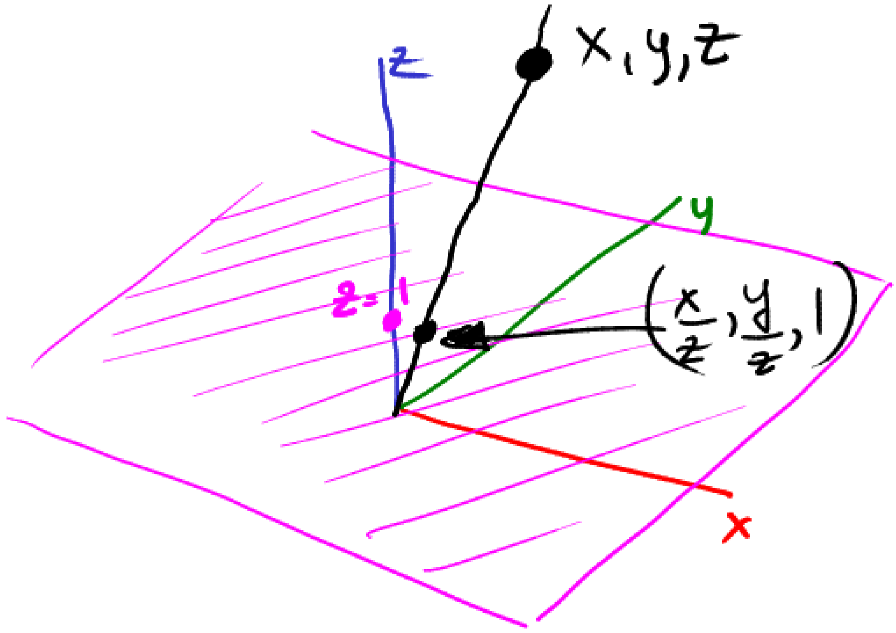
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x/z \\ y/z \end{bmatrix}$$

等一下，除数千万不能是0！

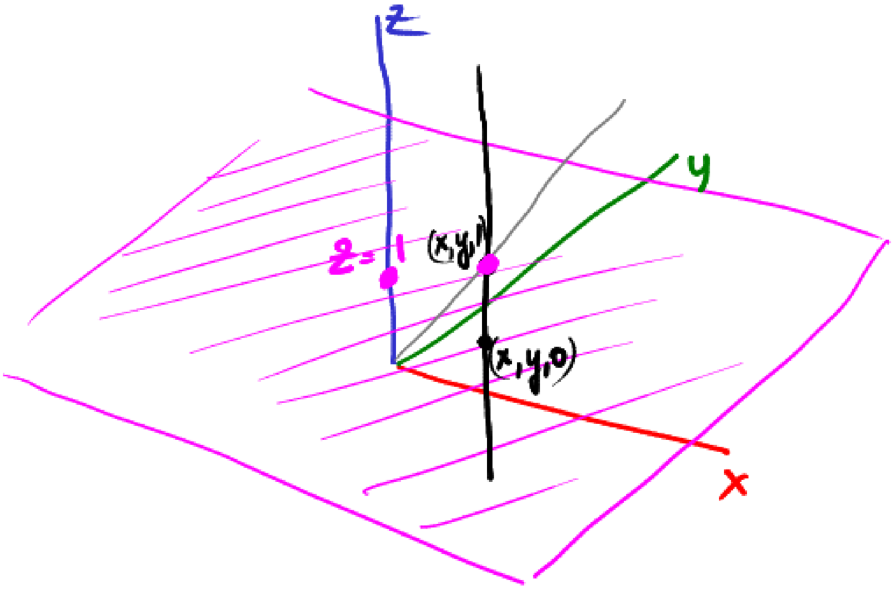
这是什么意思呢？让我们回想一下管线：

- 我们将2维平面投影到3维平面通过添加Z=1的轴
- 我们能在3维空间中得到任何我们想要的
- 对每一个我们想从2维投影到3维的点，我们都要在原点和投影点之间画一条线，然后在Z轴=1的平面上找到交点

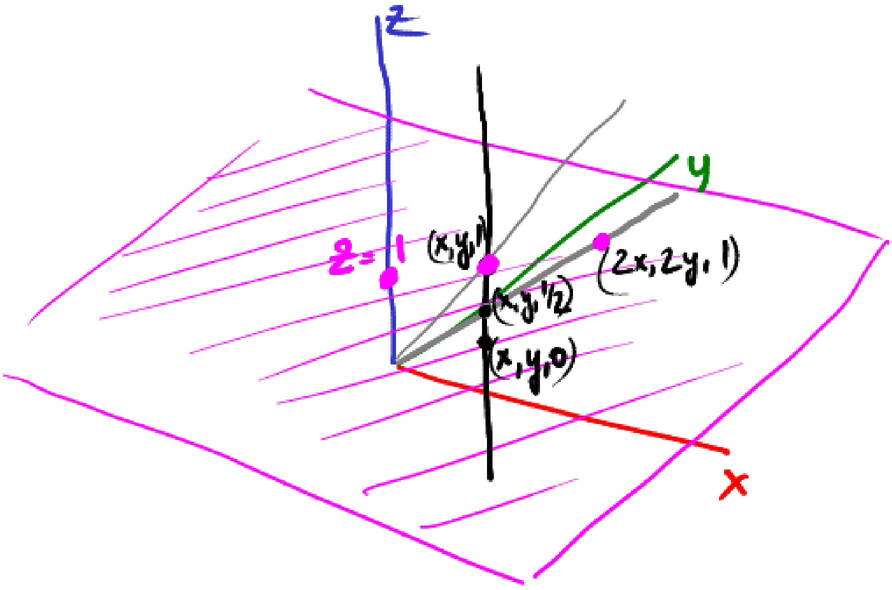
如下图所示我们的2维平面是洋红色的，点（X,Y,Z）投影到点(X/Z,Y/Z)



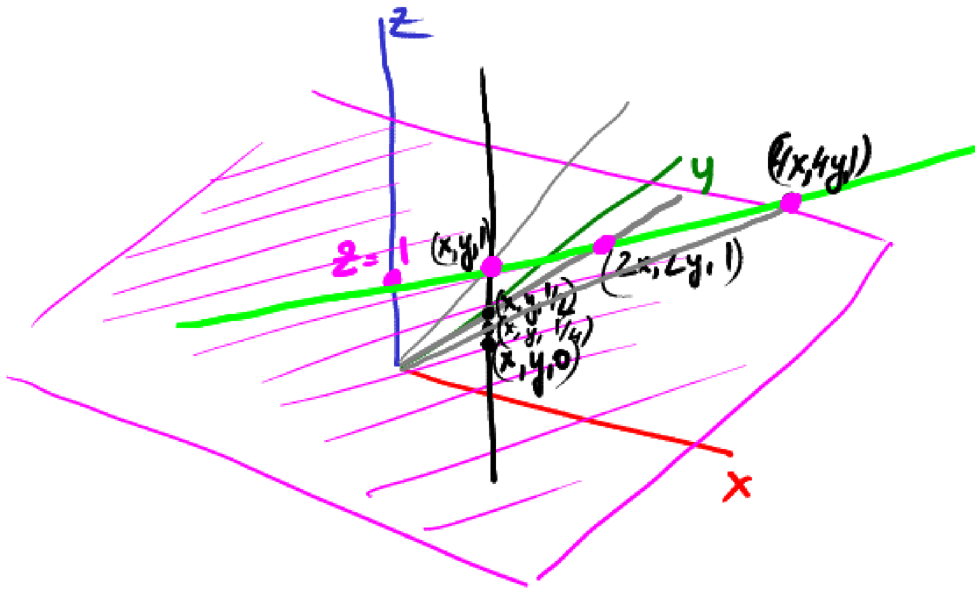
想象一个横穿过点（x，y，1）的线段。点（x，y，1）应该投影到那？当然是（x，y）上：



然后我们将这段线减少一半，点（x，y，1/2）投影就是（2x，2y）：



让我们继续，点（x，y，1/4）会变成（4x，4y）：



如果我们继续这个过程，知道接近Z=0的时候，投影将会离原点越来越远。换句话说就是点（x，y，0）将会被投影到距离点（x，y）无穷远的地方，这叫什么？对，这就叫做向量。

齐次坐标允许区分向量和点。如果一个程序写作vec2（x，y）它是向量还是点？很难说。在齐次坐标中Z=0的所有东西都可以看作向量。其他都被称为点。看：向量+向量=向量。向量–向量=向量。点+向量=点。这太棒了，不是么？

混合变换

如同我之前所说的，我们很可能要同时进行几十个坐标变换。为什么呢？让我们想象一下如果我们需要让一个2维对象绕点（x0，y0）旋转。应该怎么去做呢？我们或许可以在某处得到一些公式方法，也可以手撕，或者使用一些我们需要的工具。

我们知道怎么去移动坐标，怎么去旋转，这就是我们全部需要知道的了。移动（x0，y0）平移到原点，旋转，完成：

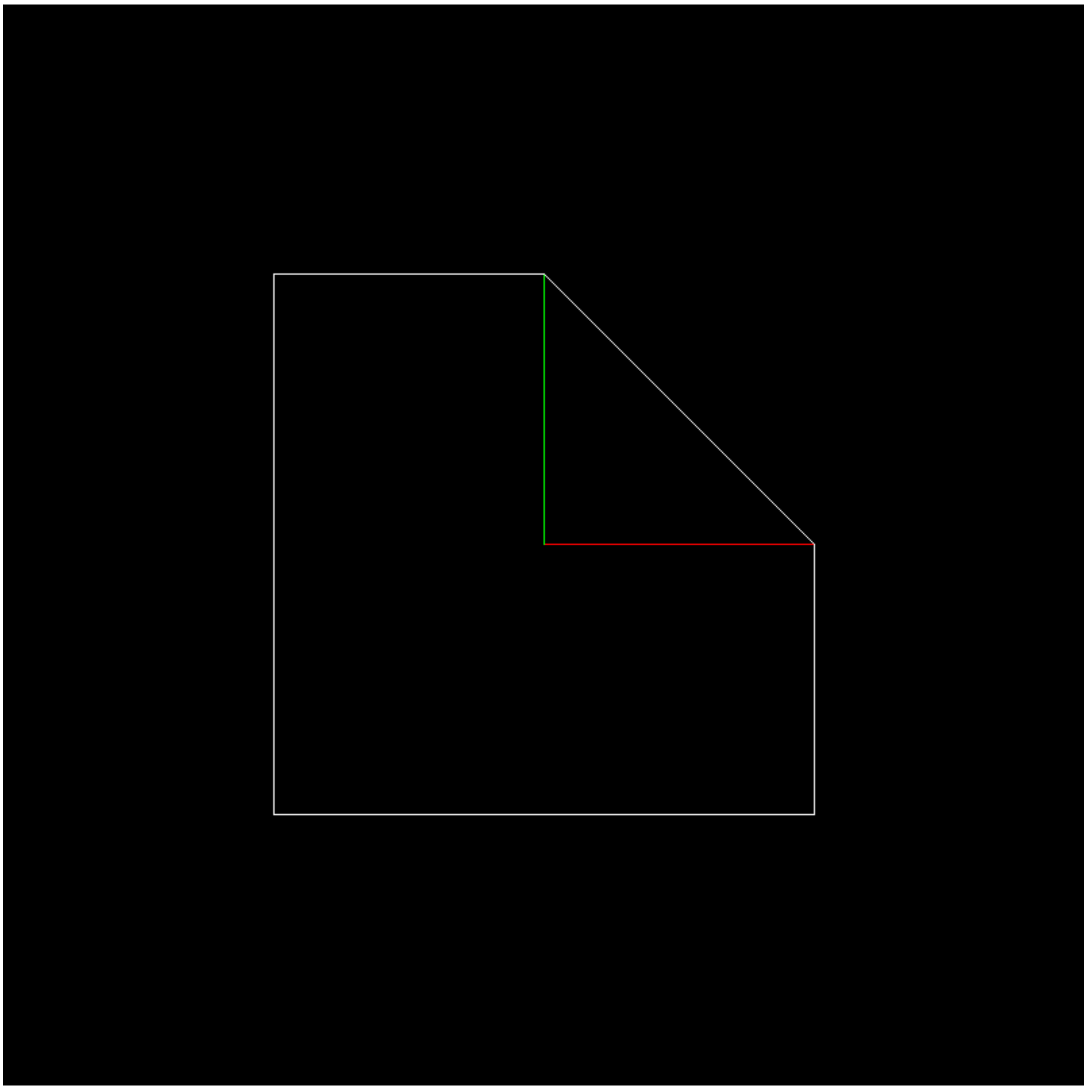
$$M = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

在三维坐标系中，这些运算会变得更繁琐，但我们需要知道的是，只要我们能将它拆分成基础的坐标变换，我们就能解决任何复杂的变换。

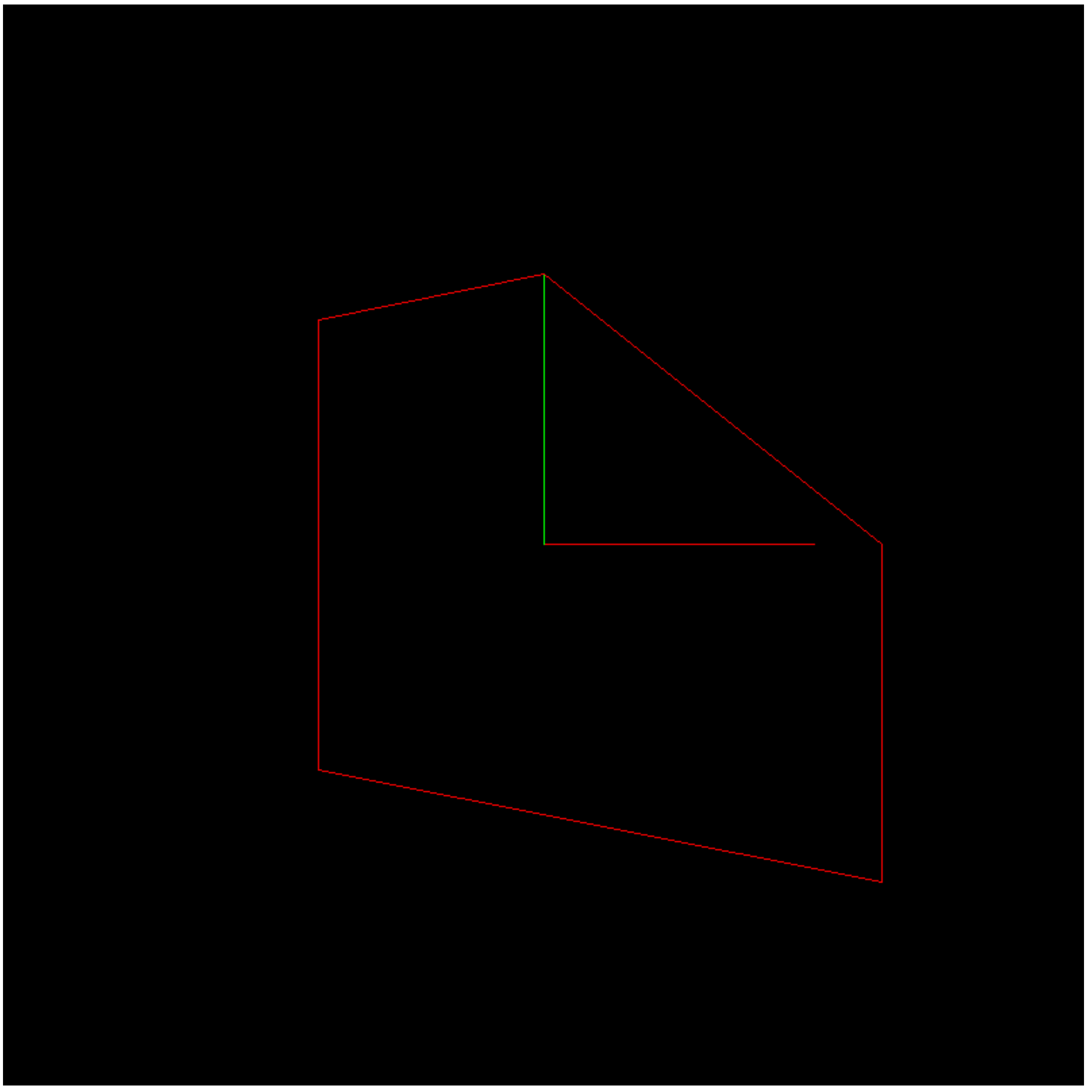
等一下，我能对这个神奇的3x3矩阵的底行做操作吗？

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/5 & 0 & 1 \end{bmatrix}$$

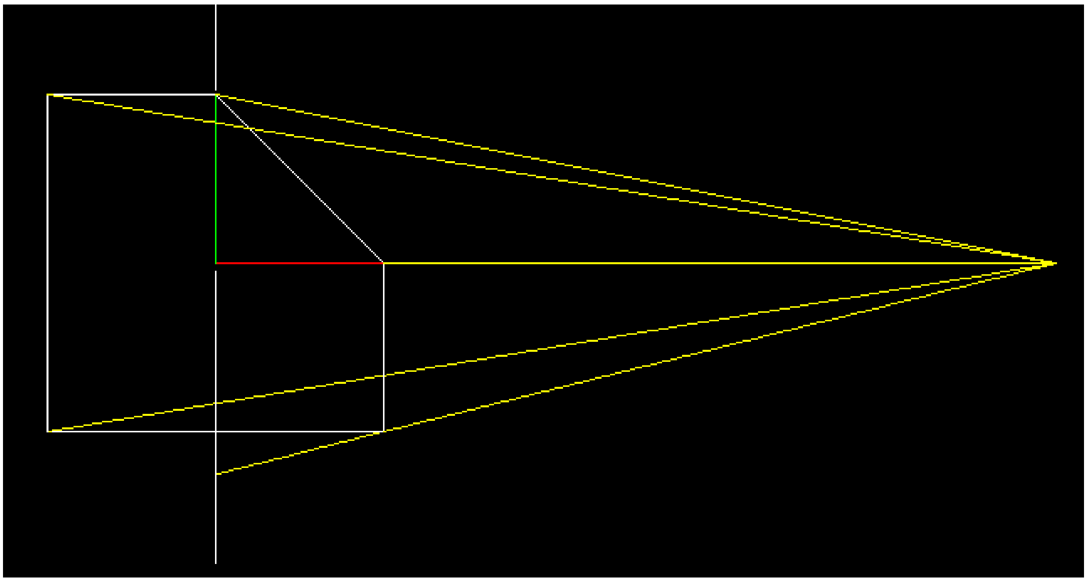
回想一下原始对象是白色的，单位向量轴是红色和绿色的：



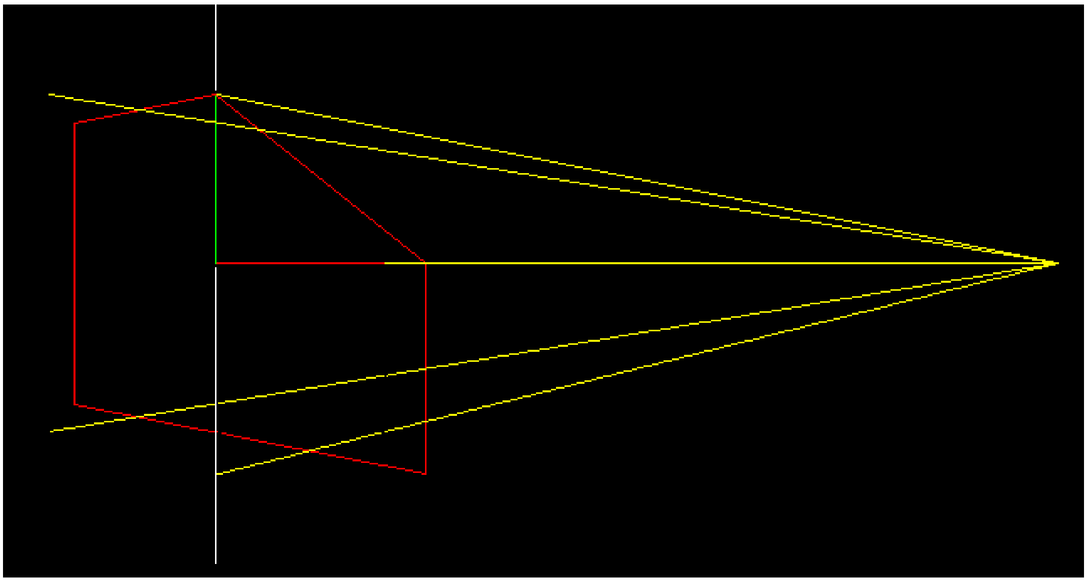
转变后的对象如下：



接下来是另一种神奇的现象。还记得我们的Y-buffer吗？我们将会对其做一样的操作：将我们的2维对象投影到x=0的线上。让我们强调一遍规则：我们使用中心投影，我们的摄像机位置在点P（5，0）并指向原点。为了找到投影我们得链接摄像机和投影在屏幕上的对象的（白色图形）连线（黄色）的交点坐标。



现在我们用之前的方式转换我们的投影对象，但是不去改变原本的黄色直线：



如果在屏幕上的红色对象使用标准正交投影，我们就能发现这两种方式会有相同的交点。让我们更近一点观察位移是怎么起作用的；所有的垂直线段没有发生改变，但是靠近摄像机的线段被拉伸，远离摄像机的线段则收缩了。如果我们选择了正确的缩放系数（在我们这个矩阵位移中是-1/5）我们就能得到透视中心的投影。

是时候在3维坐标中工作了

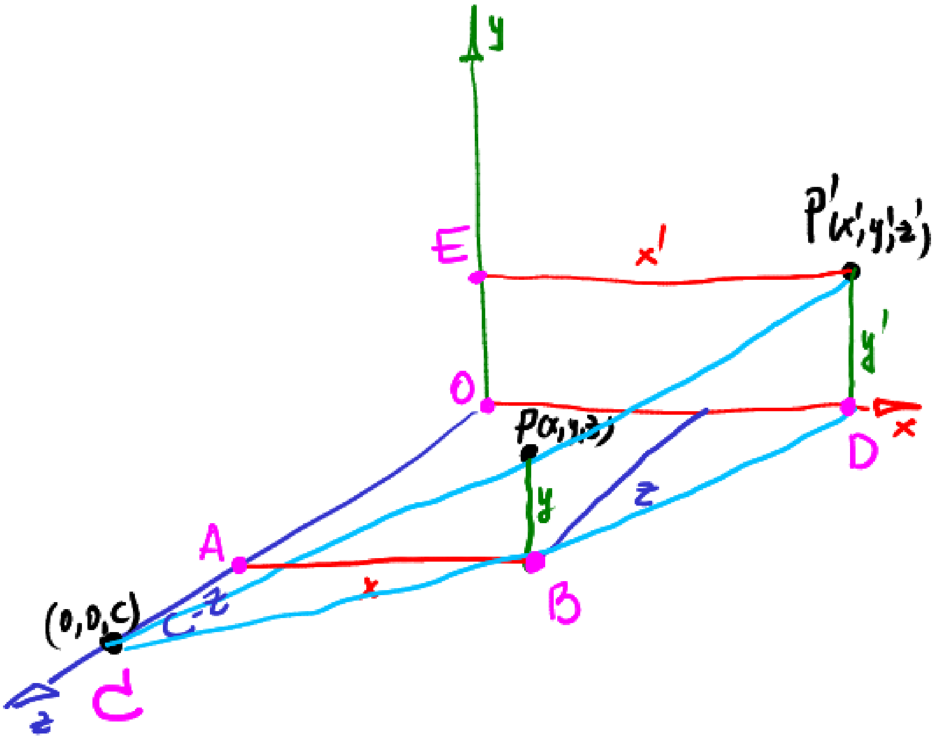
解释一下这个魔法。对2维和三维的仿射变换我们使用齐次坐标的方式：一个点（x，y，z）转换为（x，y，z，1)然后我们从4维投影回3维，比如，如果我们得到如下的变换：

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix}$$

反投影就会变成如下这样：

$$\begin{bmatrix} x \\ y \\ z \\ rz+1 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{rz+1} \\ \frac{y}{rz+1} \\ \frac{z}{rz+1} \end{bmatrix}$$

牢记这个结果，并把它放在一旁。回到标准的中心投影的定义，没有像4维坐标系转换这样花里胡哨的东西。给出一个点P=（x，y，z）我们希望将它投影在z=0的平面上，摄像机在z轴上的（0，0，c)点：



三角形ABC和ODC相似，这意味着我们能用如下的写法： |AB|/|AC|=|OD|/|OC| => x/(c−z) = x’/c.用别的话就是说：

$$x' = \frac{x}{1 - z/c}$$

同理对CPB和CP’D，也能如下表示：

$$y' = \frac{y}{1 - z/c}$$

这和我们之前放在一边的结果非常相似，但是我们是通过矩阵相乘的方式得到的这个结果，那么我们也能证得投影的系数规则是： r = −1/c.

来让我们总结一下，今天最重要的公式

如果你没有很好的去理解上述的原理，而只是复制粘贴这个公式，那么我会讨厌你。

所以，如果我想计算计算机在Z轴上（这很重要！）并跟原点有C距离的一个中心投影。我们先把点转变成4维的通过增加一个值为1的系数，然后将他乘以如下的矩阵，并反投影回3维坐标系。

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/c & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 - z/c \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 - z/c \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{1-z/c} \\ \frac{y}{1-z/c} \\ \frac{z}{1-z/c} \end{bmatrix} \tag{3}$$

我们将我们需要计算的对象进行了某种转换，只需要忘记Z轴坐标我们就能得到一张透视图。如果我们需要用到Z−buffer那就记得不要忘记Z轴坐标。代码你能在这里得到，结果你能在这篇文章的开头看见。