暂时渲染管道是分三个阶段

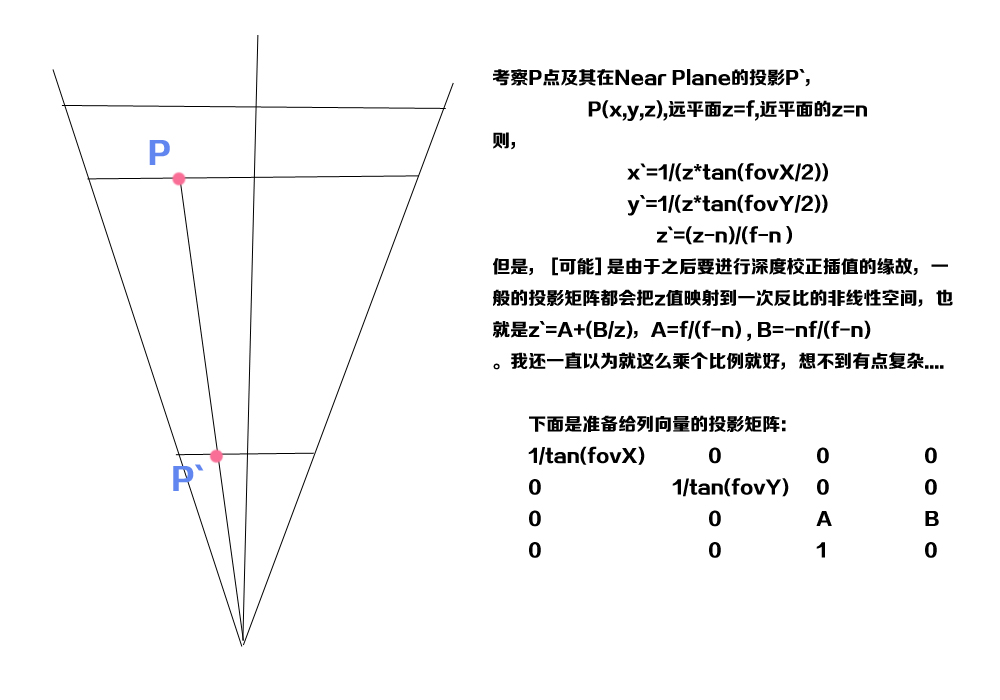
1，VertexShader （顶点变换等）

2，光栅化（Rasterization）

3，PixelShader

**1.VertexShader**

暂时呢，还是打算用顶点光照，Gourand Shading吧。（一开始还更懒打算直接flat shading）。顶点格式不使用tangent，因为这个控制台项目画质并不打算做的非常好，分辨率也低，用不上法线空间吧=。=



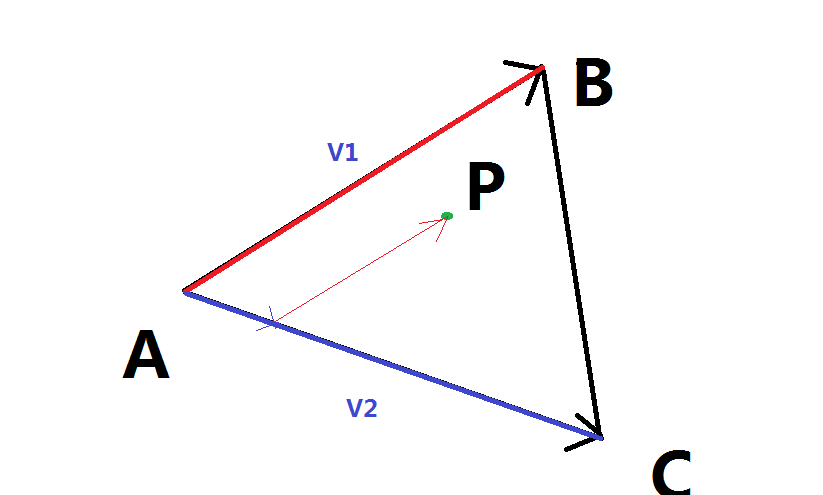
我自己现在实现的齐次裁剪空间的z是线性分布在[0,1]上的，不是国际惯例的1/z线性分布在[0,1]。投影操作国际惯例是同除保存在w坐标的view space z坐标，我的是直接搞个变量出来储存view space z坐标，然后同除x，y。

**2.Rasterization**

光栅器暂时不使用任何抗锯齿的措施，量化的方法直接使用截断取整(用UINT()强制转换一个float来截断)。画个线就相对好画，但是画三角形好像要麻烦许多，因为要填充。光栅化打算用扫描线（scan line），从上到下来一行行像素扫描，然后就可以一行行地来填充。一开始有想过先把三角形的三条边边光栅化先，再当成位图填充。但是好像先把边光栅化的话，当前行的x范围信息不太好储存。所以还是先用横着的直线和三角形求交，交得两个点再量化，然后把这一行填充，并且逐像素插值。

然后这个横着的直线和三角形求到刚好俩交点，也是有点麻烦的原来。但是大体思路和逻辑都是抄NoiseUtSlicer的平行于xz平面和空间三角形求交，先分类讨论有多少个点在想test intersection的y坐标上，然后一类一类讨论，最后返回true和直线与三角形的两个交点的x坐标，或者返回false表示求交失败。

然后逐像素插值的时候，发现要先根据在屏幕的位置找到当前像素的双线性插值系数（s,t）。一维线性插值系数就好求了，二维还要稍稍解个方程。



若已知三角形内的p点，还有两个基底向量

写成二元方程，

其中s,t是未知量，根据克拉默法则

有了双线性插值的比例系数之后，加上一些关键点的深度值，就可以对顶点属性进行逐像素的深度校正插值。

插值的时候可能不用向量，那直接用三角形三个顶点来表示插值公式，则有

顶点属性的普通双线性插值，就把上式的顶点换成对应的顶点属性。若是要进行深度校正插值，则要稍微调整公式。

**3.Pixel Shader**

= =暂时还没什么要干的，因为连纹理贴图都没打算弄...慢死了。但是纹理坐标这个顶点属性依旧是保留了，说不了过几天就反悔了，想实现纹理贴图了呢→\_→（结果还真的实现了。。。）