很像基于重采样的切向畸变(tangential distortion)校正

pass的定义：xxxx

假设显示屏是一个矩形，那么可以想象一下，矩形背后的有一些物体，物体表面反射的光线会穿过矩形，最终射到视网膜的某一点上。但是**人眼无法分辨光线是从屏幕发出来的，还是屏幕后面的物体反射而来的**。这样子一来，屏幕就可以当成虚拟场景在真实世界中表现媒介，用屏幕发出的光来模型虚拟物体反射出来的光。

具体的实现思路简单易懂：

输入：

1. 一个**显示屏描述模型**（三角形网格），描述了物理屏幕的形状。例如现实世界的显示屏是矩形，那么其屏幕描述模型就由两个三角形组成。
2. 肉眼的位置描述：肉眼将在虚拟场景里面建模为一个针孔摄像机，而输入只需要摄像机相对于**显示屏坐标系下的坐标**。

Pass1：用正常的方法渲染场景，设camera(pos,posture)，渲染至RenderTargetA，这时候的图像是透视投影到传统的近平面(near plane)产生的。保持视点不变，进入post processing阶段。

Pass2：qwerty distortion可以看作是一个“渲染一个纹理”的过程，而这一张纹理会被贴在现实世界的物理屏幕表面上（一般是一个矩形，但理论上可以是其他形状的屏幕，只要能用三角形网格描述即可）。这张用于欺骗人眼的贴图可以用后处理（post processing）的方法生成，这需要对pass1的RenderTargetA的结果进行符合一定规律(pattern)的重采样。

一般的Post processing需要两个pass，pass1渲染原始图像，pass2里面画一个覆盖全屏幕的、有纹理坐标的矩形（两个三角形），那么在pass2里面的pixel shader就可以对pass1生成的原始图像进行采样和处理并输出颜色，从而实现一定效果的后处理。qwerty distortion的步骤跟一般的post processing比较像，但是qwerty Distortion的pass2的输入三角形的顶点属性需要有世界坐标(posW)。VS阶段输出的SV\_POSITION应该是显示屏描述模型顶点的纹理坐标，这是用来覆盖屏幕的。三角形经过光栅化后进入Pixel Shader阶段。对于每个像素，posW乘个perspective projection matrix (与pass1一致)，得到在RenderTargetA中采样的纹理坐标texcoord。

Pass1渲染的图像是提供足够的信息给pass2采样的，所以为了使pass2的分辨率足够高，**同时又保证pass1渲染效率和图像传输带宽利用率**（也即pass2不需要降采样太多倍，不然最终图像某些部分的分辨率会变低，总不能为了保证pass2生成的distorted image的分辨率而肆意提高renderTargetA的分辨率），pass1的camera的视角、朝向等参数需要斟酌一下。(2018.2.8)实验证明pass1 camera朝向对pass2最终生成图像形状几乎没有影响（只有采样的影响），位置才对最终显示图像有影响，这就对上了之前的猜测，也就是“定位”只需要**视点位置**即可。如此一来，可能要想想怎么设置摄像机的fov和lookat来最大化pass1里面虚拟显示屏的透视投影占RenderTarget的面积比例，从而pass2能有更高分辨率的输入用以（升）采样。

显示屏描述模型顶点的纹理坐标要和普通显示屏的纹理空间坐标系对上才行。

虽然用手机可能可以拍出不错的效果（2018.2.8是静态图片），但是如果屏幕小的时候近距离的的双目效应依旧不能轻易消除，看来还是大屏幕远距离才能把双目近似为单目啊。

分成两pass实现的优点：易于模块化，并在原来的渲染系统上加以修改。缺点：要达到较好的画质则需要稍微提高pass1渲染的分辨率。其实也可以考虑另一种的实现方法：**延迟透视投影**。也就是不在VS里面进行perspective projection，而是在PS的一开始进行。因为qwerty distortion最终成像的的每一个像素点一般都对应着不同的“近平面深度”。而近平面深度的参数一般是蕴含在projection matrix里面的，而Projection一般是一帧更新一次，在VS里面使用，透视除法也是由硬件实现。而要实现3D画的渲染，则透视除法就要延迟到Pixel Shader里面实现。在PS的一开始根据（GG，好像VS也要处理好顶点的成像位置才有救啊）