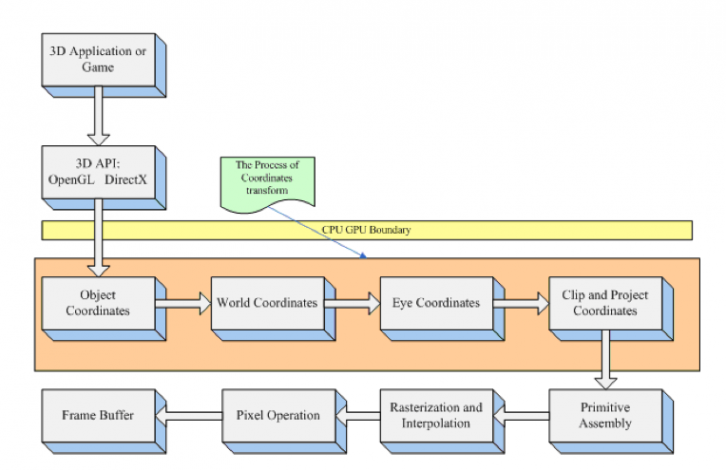
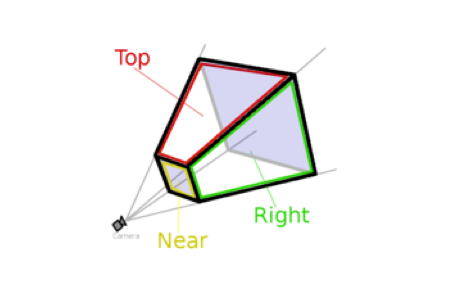
**引言**   
shader到底是干什么用的？shader的工作原理是什么？   
其实当我们对这个问题还很懵懂的时候，就已经开始急不可耐的要四处搜寻有关shader的资料，恨不得立刻上手写一个出来。但看了一些资料甚至看了不少cg的语法之后，我们还是很迷茫，UNITY\_MATRIX\_MVP到底是个什么矩阵？它和v.vertex相乘出来的又是什么玩意？当这些问题困扰我们很久之后，我们才发现，原来我们是站在浮沙上筑高台，根基都没有打牢当然不可能盖得起高楼大厦了。   
那根基是什么呢？大牛曰，计算机图形学。   
shader中文名叫着色器，顾名思义，它的作用可以先简单理解为给屏幕上的物体画上颜色。而什么东西负责给屏幕上画颜色？当然是GPU，所以我们写shader的目的就是告诉GPU往屏幕哪里画、怎么画。说到这其实大家应该很明白了，如果我们连GPU的工作原理都不知道，何谈指挥它？   
说到计算机图形学，包括我在内很多同学都非常害怕它，因为里面包含了各种艰深的理论、变换，大量的公式什么的。其实我们大可不必一开始就吓倒自己，先从基本概念开始，慢慢来，总有一天我们也会成为大牛~！   
最后，这篇文章不算是原创，最多算是摘要+读后感，很多概念性文字都是我从书里搬过来后再加上自己的理解，算是和大家一起学习，有理解不当之处还请多多指教。   
废话不多说，让我们来进入第一章的学习，GPU的渲染管线。 

**正文**   
所谓GPU的渲染管线，听起来好像很高深的样子，其实我们可以把它理解为一个流程，就是我们告诉GPU一堆数据，最后得出来一副二维图像，而这些数据就包括了”视点、三维物体、光源、照明模型、纹理”等元素。   
在各种图形学的书中，渲染管线主要分为三个阶段：应用程序阶段、几何阶段、光栅阶段。   
  
1，应用程序阶段。   
这个阶段相对比较好理解，就比如我们在Unity里开发了一个游戏，其实很多底层的东西Unity都帮我们实现好了，例如碰撞检测、视锥剪裁等等，这个阶段主要是和CPU、内存打交道，在把该计算的都计算完以后，在这个阶段的末端，这些计算好的数据（顶点坐标、法向量、纹理坐标、纹理）就会通过数据总线传给图形硬件，作为我们进一步处理的源数据。   
  
2，几何阶段。   
主要负责顶点坐标变换、光照、裁剪、投影以及屏幕映射，改阶段基于GPU进行运算，在该阶段的末端得到了经过变换和投影之后的顶点坐标、颜色、以及纹理坐标。简而言之，几何阶段的主要工作就是“变换三维顶点坐标”和“光照计算”。   
问题随之而来，为什么要变换顶点坐标？我是这么理解的，比如你有一个三维游戏场景，场景中的每个模型都可以用一个向量来确定它的位置，但如何让计算机根据这些坐标把模型正确的、有层次的画在屏幕上？这就是我们需要变换三维顶点坐标的原因，最终目的就是让GPU可以将这些三维数据绘制到二维屏幕上。   
根据顶点坐标变换的先后顺序，主要有如下几个坐标空间：Object space，模型坐标空间；World space，世界坐标空间；Eye space，观察坐标空间；Clip and Project space，屏幕坐标空间。下图就是GPU的整个处理流程，深色区域就是顶点坐标空间的变换流程，大家了解一下即可，我们需要关注的是每个坐标空间的具体含义和坐标空间之间转换的方法。



2.1，从object space到world space   
object space有两层核心含义，第一，object space中的坐标值就是模型文件中的顶点值，这些值是在建立模型时得到的，例如一个.max文件，里面包含的数据就是object space的坐标。第二，object space的坐标与其他物体没有任何参照关系，这是object space和world space区分的关键。world space坐标的实际意义就有有一个坐标原点，物体跟坐标原点相比较才能知道自己的确切位置。例如在unity中，我们将一个模型导入到场景中以后，它的transform就是世界坐标。   
  
2.2，从world space到eye space   
所谓eye space，就是以摄像机为原点，由视线方向、视角和远近平面，共同组成的一个梯形体，如下图，称之为视锥(viewing frustum)。近平面，是梯形体较小的矩形面，也是靠近摄像机的平面，远平面就是梯形体较大的矩形，作为投影平面。在这个梯形体的内的数据是可见的，超出的部分会被视点去除，也叫视锥剪裁。   
例如在游戏中的漫游功能，屏幕的内容随摄像机的移动而变化，这是因为GPU将物体的顶点坐标从world space转换到了eye space。



2.3，从eye space到project and clip space   
eye space坐标转换到project and clip space坐标的过程其实就是一个投影、剪裁、映射的过程。因为在不规则的视锥体内剪裁是一件非常困难的事，所以前人们将剪裁安排到一个单位立方体中进行，这个立方体被称为规范立方体（CCV），CVV的近平面（对应视锥体的近平面）的x、y坐标对应屏幕像素坐标（左下角0、0），z代表画面像素深度。所以这个转换过程事实上由三步组成：   
（1），用透视变换矩阵把顶点从视锥体变换到CVV中；   
（2），在CVV内进行剪裁；   
（3），屏幕映射：将经过前两步得到的坐标映射到屏幕坐标系上。   
  
2.4，primitive assembly（图元装配）和triangle setup（三角形处理）   
到目前为止我们得到了一堆顶点的数据，这一步就是根据这些顶点的原始连接关系还原出网格结构。网格由顶点和索引组成，这个阶段就是根据索引将顶点链接到一起，组成线、面单元，然后进行剪裁，如果一个三角形超出屏幕以外，例如两个顶点在屏幕内，一个顶点在屏幕外，这时我们在屏幕上看到的就是一个四边形，然后把这个四边形切成两个小的三角形。   
现在我们得到了一堆在屏幕坐标上的三角形面片，这些面片是用于光栅化的。   
  
3，光栅化阶段。   
经过上面的步骤之后，我们得到了每个点的屏幕坐标值，和我们需要绘制的图元，但此时还有两个问题：   
（1）屏幕坐标是浮点数，但像素是用整数来表示的，如何确定屏幕坐标值所对应的像素？   
（2）如何根据已确定位置的点，在屏幕上画出线段或者三角形？   
对于问题1，绘制的位置只能接近两指定端点间的实际线段位置，例如，一条线段的位置是（10.48， 20.51），转换为像素位置就是（10，21）。   
问题2，涉及到具体的画线和填充算法，有兴趣的话可以研究。   
这个过程结束后，顶点和图元已经对应到像素，之后的流程就是如何处理像素，即给像素赋予颜色值。   
给像素赋予颜色的阶段称为Pixel Operation，是在更新帧缓存之前，执行最后一系列针对每个片段的操作，其目的是计算出每个像素的颜色值。在这个阶段，被遮挡的面通过一个被称为深度测试的过程消除。   
pixel operation包含下面这些流程：   
（1）消除遮挡面；   
（2）Texture operation，纹理操作，根据像素的纹理坐标，查询对应的纹理值；   
（3）Blending，通常称为alpha blending，根据目前已经画好的颜色，与正在计算的颜色的alpha值混合，形成新的颜色。   
（4）Filtering，将正在计算的颜色经过某种滤镜后输出。   
该阶段之后，像素的颜色值被写入帧缓存中。

http://game.ceeger.com/forum/read.php?tid=10970&fid=2