但是，试着想一想，多边形在内侧还是外侧的判断，要如何进行呢？

实际上，就是通过刚才说的顶点的连接顺序为判断基准的。顺时针连接顶点的多边形是在外侧，而逆时针连接的多边形在内侧。

顶点就是至少包含了坐标情报的三维空间上的任意的一个点，三个点用线连接起来，表现为一个三角形的多边形。而多边形根据顶点的连接顺序不同，分为内侧和外侧，根据这个可以进行遮挡剔除。

再简单一点说明的话，顶点着色器就是处理顶点相关的信息，片段着色器就是处理画面上的颜色信息。

基本上，用顶点着色器来做什么是比较自由的，但是在WebGL程序中，首先生成模型，视图，投影的各个矩阵，然后进行合并，最后将得到的坐标变换的矩阵传给顶点着色器，这是一般的做法。

顶点着色器和片段着色器，都可以通过GLSL来书写，基本上它们算是一个组合。着色器的内部，必须要定义一个main函数，在这个函数里面添加自己的处理。而且，要从WebGL一侧向着色器传递数据的时候，需要用到一些特殊的修饰符所定义的变量。

要向着色器传递各个顶点的不同的信息的时候，使用attribute修饰符声明变量，要向着色器传递对所有顶点来说都一样的信息的时候，使用uniform修饰符声明变量。

另外，从顶点着色器向片段着色器传递数据的时候，使用varying修饰符声明变量。

顶点着色器中内置的变量gl\_Position必须赋值，而片段着色器的内置变量gl\_FragColor虽然不是必须赋值的，但是一般情况下都会赋值。

定点群放到什么位置，就表现为坐标，一般叫做

坐标。局部坐标就是模型的各个顶点相对于原点（x，y，z都为0）的坐标。  
比如，一个局部坐标为（1.0,0.0,0.0）的顶点，x轴方向距离原点的距离是1.0。同样，各个顶点都依次定义了局域坐标，这样顶点的位置就形成了。

WebGL中的顶点缓存叫做VBO（vertex buffer object）。

顶点缓存的作用，不光是保存顶点的位置，位置以外跟顶点相关的信息都可以用顶点缓存来保存。  
比如，顶点的法线，颜色，文理坐标等所有跟顶点相关的信息都可以用顶点缓存来保存和管理。但是需要注意一点的是，向顶点信息中追加信息的时候，需要使用相应的VBO。

其实，担任向这个attribute变量里传入数据的任务的就是VBO。WebGL的程序中，先把顶点的信息保存到VBO中，接着，通知着色器哪个VBO和哪个attribute变量相关，然后顶点着色器就可以正确的处理这些顶点了。  
根据前面的内容，顶点缓存相关的处理的具体流程如下。  
・顶点的各种信息保存到数组里  
・使用WebGL的方法生成VBO  
・使用WebGL的方法将数组中的信息传给VBO  
・顶点着色器中的attribute函数和VBO结合  
VBO的生成过程中，首先在最初的时候必须把数据保存到数组中，因为顶点的信息（位置）中必须有x，y，z，所以数组的长度必须是顶点数x3，这个时候需要注意，数组不可以使用多维数组，VBO的生成需要使用一维数组。  
准备好保存顶点信息的数组之后，使用WebGL的context的方法生成VBO，当然生成的时候VBO是空的，然后将顶点信息的数组传给它。  
然后，比如把顶点着色器中的attribute函数和VBO关联起来。上面也说了，VBO中不是只能保存一种信息，位置情报以外的法线和颜色等信息存在的时候，要准备合适的VBO，然后通知WebGL哪个VBO和哪个attribute变量相关联。

其实，实现从一个着色器向另一个着色器传递数据的，不是别的，就是程序对象。程序对象是管理顶点着色器和片段着色器，或者WebGL程序和各个着色器之间进行数据的互相通信的重要的对象。

顶点属性和VBO一对一进行分配，然后传给顶点着色器。

第一个是模型变换矩阵，DirectX中叫做世界变换矩阵。模型变换矩阵影响的是所绘制的模型，模型的位置，模型的旋转，模型的放大和缩小等相关的情况。  
第二个是视图变换矩阵，简单来说，就是定义拍摄3D空间的镜头（摄像机），决定了镜头的位置，镜头的参考点，镜头的方向等。  
第三个是投影变换矩阵，这个坐标变换定义了屏幕的横竖比例，剪切的领域等，另外获取远近法则的效果也需要用这个变换矩阵。

不管在片段着色器中有没有做什么特殊的处理，首先要将precision相关的设定写上，否则在编译着色器的时候会出错。这就像魔法的咒语一样，逃不掉的。

>>遮挡剔除的内侧和外侧切换的方法

多边形的内侧和外侧是根据顶点的连接顺序来判断的，而这个判断基准反过来的情况也是有的，形成多边形的顶点的连接顺序是顺时针的时候是外侧，逆时针的时候为内侧，想要反过来判断的话，顺时针就变成了内侧。

顺时针统称为CW，因为［顺时针］的英语是ClockWise，CW就是它的头文字。而逆时针统称为CCW，因为［逆时针］的英语为CounterClockWise。改变WebGL中遮挡剔除的内侧和外侧的判断标准的函数是frontFace，参数就是刚才提到的CW和CCW。

将顺时针设置为［外侧］的代码：gl.frontFace(gl.CW);

将顺时针设置为［内侧］的代码：gl.frontFace(gl.CCW);

但是有时你可能需要自己控制转换。你想要自己控制转换的一个原因是这些常用变换必须按顺序分别调用，每次调用都是一次具有一定开销的矩阵乘法（稍后讨论）。如果你自己进行转换，你可以将多个转换组合成一个矩阵，从而减少每帧都要进行的矩阵乘法操作。

由于你可以向量化你的矩阵乘法调用，所以你可以通过定义自己的矩阵来获取最大性能。据我所知，关于此点iPhone 并无文档记录，但作为基本规则OpenGL ES将对向量间或顶点和矩阵间的乘法进行硬件加速，但两个转换矩阵间的乘法并无加速。通过矩阵乘法向量化，你可以获得比让OpenGL进行矩阵乘法更好的性能。由于通常矩阵与矩阵的乘法调用的数量远小于向量/顶点间的矩阵乘法调用的数量，所以这并不会带来巨大的性能提升，但在一个复杂的3D程序中，每个方面小的额外性能提升都很有好处。

# Debug

## 可以引入webgl-debug.js文件，从而能在控制台中直接打印错误信息（而不会当glsl出错时，弹出对话框？）

## 使用webgl inspector工具，可查看glsl的状态（查看捕捉的时刻的状态）

有两种方式启用该工具：

1、在google web store上下载extensive，然后重启浏览器，允许GL（可能被阻止，需要点击地址栏右边，打开GL）

2、在页面中引入core/embed.js

## 参考资料

https://www.khronos.org/webgl/wiki/Debugging

<https://github.com/benvanik/WebGL-Inspector/blob/master/readme.md>

# 纪录

优化：

纹理压缩

对于要修改顶点的情况，可以将bindbufferdata的第三个参数改为dynamic\_draw，同时使用bindsubbuuferdata，从而直接替换原有buffer的顶点数据（不过貌似也会有问题：using fewer draw calls by changing buffer data with something like glbuffersubdata may actually hurt your performance more than help because gl is not allowed to make changes to your data before previous draw calls finish (and remember that the cpu and gpu run asynchronously). so you either force gl to stall and finish previous draw calls, or make a copy of your buffer so you can change it without affecting previous calls）

use dirty flag, only draw if dirty