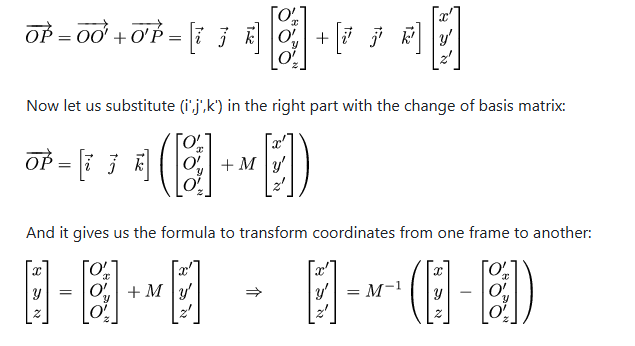
# C++软光栅渲染

1. 使用布雷斯汉姆算法实现线段绘制以优化性能，主题思想是提前乘除法，以近似减少浮点数运算。如果x增加1那么y会增加y1-y0/x1-x0，但是此时我们不立刻进行增加而是让这个值进行累积当这个值大于0.5那么y+1。
2. 判断点是否在一个三角形内部的方法有两种，首先非图形学领域中常用的矢量叉积法，直接判断向量叉积的符号是否相同如果相同那么视为在三角形内，其次是现代渲染管线中使用的重心坐标法，计算目标点的重心坐标，如果重心坐标的值均大于0那么则在三角形内部，这个方法在整个过程中只计算了一次叉乘，效率更高的同时还能得到用于插值的参数。
3. 光栅化常见的方法有两种，一种是优于CPU的扫描线，另一种是利于GPU并行计算的包围盒。
4. 背面剔除，如果光线和法向量的乘积为负数那么表明光线来源于模型的背面此时我们可以剔除这些不可见的三角形以优化性能
5. Z-buffer的作用在于判断像素的覆盖关系，其值是通过三角形顶点插值来进行计算的。其中会记录所有坐标的当前距离相机的距离。
6. 纹理也是通过重心坐标插值来进行计算的。注意重心坐标计算的结果
7. 缩放和切变都是线性变换直接通过矩阵乘法来获得，旋转也是线性变换，任意的旋转都可以通过三次切变来实现，平移通过在线性变换处理完成后加上一个矩阵来实现。但是这样的做法并不优雅如果我们希望将所有的变换都放到一个一个矩阵中进行呢？引入齐次坐标在原本的坐标向量中增加一个1。齐次坐标除了用来统一计算方式外
8. 世界空间移动到观察空间：在OpenGL的渲染的过程中相机的位置通常是不会发生改变的,但是我们需要在相机的视角下进行光照裁剪投影之类的操作如果不进行移动那么后续的每一次计算都需要考虑摄像机的视角问题，所以我们可以直接将模型移动到摄像机空间以优化后续计算。

如何确定摄像机坐标：将相机与原点的连线作为v轴，取世界坐标的vector.up，与其叉乘得到r轴，在将v轴与r轴叉乘得到u轴。

通过计算可知实际上每一个点的移动都只需要乘轴移动的逆矩阵。



1. 三种最经典的着色模型

平面着色：整个图元都填充相同的颜色，效率最高但是效果最烂

高洛德着色：片元的所有数据都插值进行计算

冯氏着色：只插值计算片元的法向量，其他数据都通过计算得出，效果最好但是效率比较低。

10.反射向量的计算公式：入射光线 - 2\*（入射光线 \* 法向量）\*法向量

11.法线贴图用于采样得到片元具体的法线以便于实现计算每一个片元的实际的光照等信息，只有在冯氏着色中会使用，前面的两种着色方式都不需要使用片元的法向量，只需要在顶点处使用法向量来计算颜色，然后在片元处进行插值即可。

12.遍历三角形中的片元的方法。使用扫描线算法，将三角形分为上下两个部分，分别进行填充，先计算左右的两个点然后填充中间的像素。使用包围盒算法，先计算包围盒然后通过重心坐标判断点是否被三角形覆盖。计算重心的方法有两种思路，一种是公式推导出重心坐标构成的向量垂直于AB，AC，PA的x，y分量构成的向量，另一种是使用面积，而面积可以通过叉乘得到。

13.

# 鸟群算法

三条最主要的规则：

分离，对齐，聚集

分离规则：游戏物体避开彼此，远离拥挤处，四叉树判断

对齐规则：游戏物体向着整体的平均移动方向移动

聚集规则：游戏物体向着整体的中心进行移动，计算集群的质心

这三条规则会分别得到一个方向，将这些方向和原始目标方向进行加权就得到的了鸟群的效果。

本身没有向着目标方向的精确寻路，只会向着目标方向进行粗略的移动和面对障碍时的被动避障行为。

# SteeringBehaviour寻路算法

算法是对鸟群算法的优化，使用力来对游戏物体进行驱动，游戏物体实际的移动是多种力的叠加，每一种力进行单独的计算随后叠加。

常见的力有：

Seek：让游戏物体靠近目标的力。

Avoidance：让游戏物体远离障碍，越靠近障碍该力越大

Separate：让游戏物体原理彼此，游戏物体之间越靠近该力越大

# FlowField寻路

网格化地图然后通过从起点开始的BFS(使用队列实现，类似于二叉树的层次遍历)来得到所有网格到起点的距离，随后遍历每一个网格到临近的8个网格，将其指向更加接近起点的网格形成向量场，每一个点根据当前的网格的指向进行移动。只是依靠当前位于的节点来计算移动方向的话那么效果并不太好，所以考虑使用游戏物体覆盖的所有的四个网格来双线性插值计算移动的实际方向。

# NevMesh寻路

分为3步

1.构建导航网格：

使用德劳内三角剖分，其能实现生成的三角形每一个的外接圆都不包含其他顶点，将希望寻路的范围视为一个三角形，在这个三角形中不断插入障碍顶点，该顶点将当前三角形再分割为三个三角形，如果这三个子三角形的有邻接三角形那么判断该子三角形的外接圆是否包含其他顶点如果包含那么将临界边移动到该插入点和包含的其他顶点。

即插入顶点，检查（子三角形的临界三角形）是否满足空圆特性，翻转边

2.基于网格寻路

使用A\*等算法进行构造出来的网格寻路。

1. 平滑路径

不断移动交替移动顶点，如果出现顶点边交叉那么得到一个拐点，再从此拐点进行下一次移动，最后连接所有顶点得到最后的平滑路径。

# AOI算法

AOI算法即兴趣区域算法，让多玩家联机游戏的玩家只关注与自身有关的其他玩家。

主要的实现方法有两种：九宫格法和十字链表法

九宫格法：将地图分割为与玩家视角相同大小的网格，服务器只转发玩家周围九个网格的其他玩家的操作，玩家进入其他玩家的网格范围时向彼此发送进入信息，离开时发送离开的信息。

十字链表法：标定玩家的视野的上下限，横向链表保存玩家坐标以及视野的X坐标的上下限，纵向链表保存玩家以及视野的Y坐标的上下限，当玩家同时进入视野X和Y的视野范围时进行对玩家的检测。

# Floyd算法

Floyd算法用于多对多的寻路，使用矩阵来依次添加点看是否距离更短。迭代的算法，在每一轮迭代中选择一个点作为中间点。

# Dijkstra算法

Dijkstra算法用于一对多的寻路，一样是迭代的算法，每次选择当前所有点中距离最短的点来作为下一次的迭代的中间点。

# 柏林噪声算法

值噪声和柏林噪声都是基于网格双线性插值,但是值噪声是对网格的四周的点直接进行插值，而柏林噪声将点的位置原本的值修改为向量，将点本身的向量与点与目标点的连线向量进行叉乘得到点处的值进行插值，相较于值噪声更加自然。