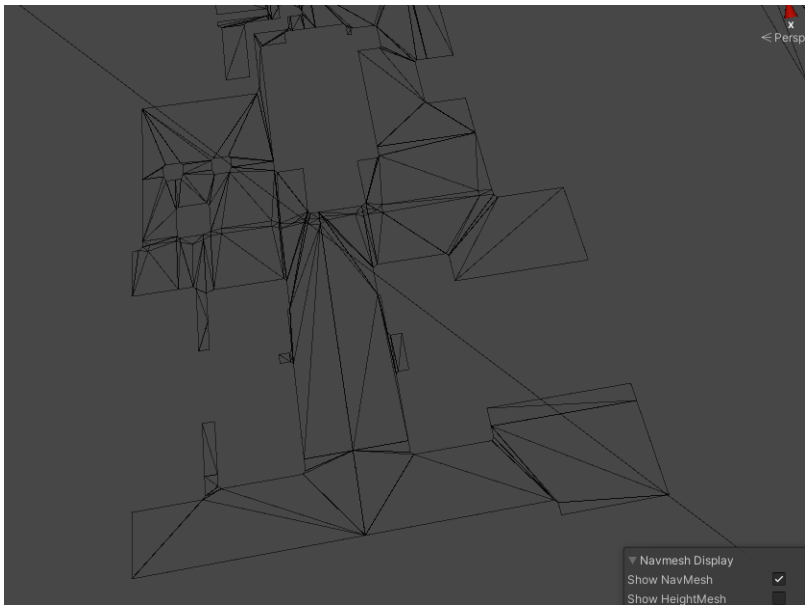
NavMesh

- **导航网格**（NavMesh）
  - 一种数据结构，用于描述游戏世界的可行走表面，并允许在游戏世界中寻找从一个可行走位置到另一个可行走位置的路径
  - 从关卡几何体自动构建或烘焙的
- **导航网格代理 (NavMesh Agent)**
  - 帮助您创建在朝目标移动时能够彼此避开的角色
  - 主角
- **网格外链接 (Off-Mesh Link)**
  - 允许您合并无法使用可行走表面来表示的导航捷径。例如，跳过沟渠或围栏
- **导航网格障碍物 (NavMesh Obstacle)**
  - 描述代理在世界中导航时应避开的移动障碍物
- UnityEngine.AI.NavMesh.CalculateTriangulation()
  - 计算并返回当前导航网格的简单三角形剖分 - 包含顶点、三角形索引和导航网格层

areas	导航网格三角形剖分的导航网格区域索引。
indices	导航网格三角形剖分的三角形索引。
vertices	导航网格三角形剖分的顶点。

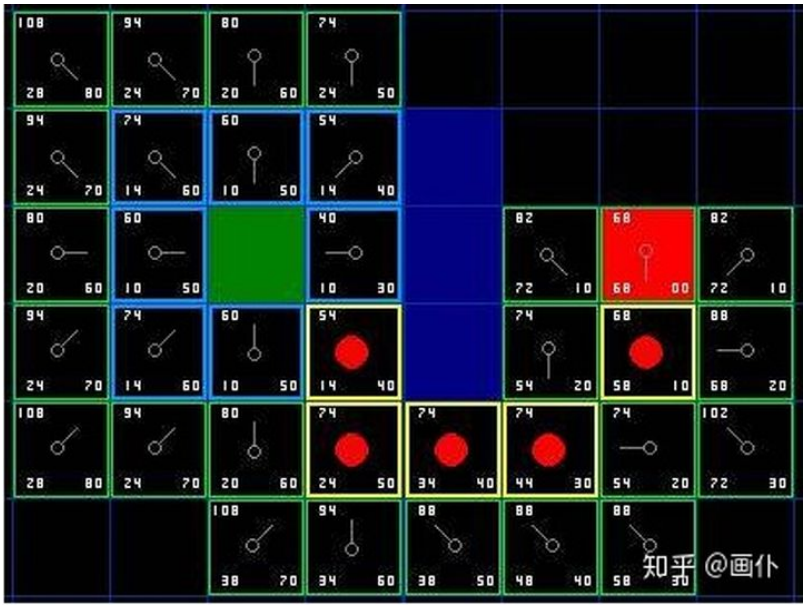


  - 上面是unity bake出来的navmesh的图，在wireframe模式下的网格
  - 接下来，**优化顶点**：
    - 计算每个顶点的sqrMagnitude，然后存到作为key存到表里
    - 遍历每个顶点，然后取出key值一定范围内的其他顶点
    - 判断其他顶点与当前顶点的范围是否小于一定值，如果小于，则进行合并

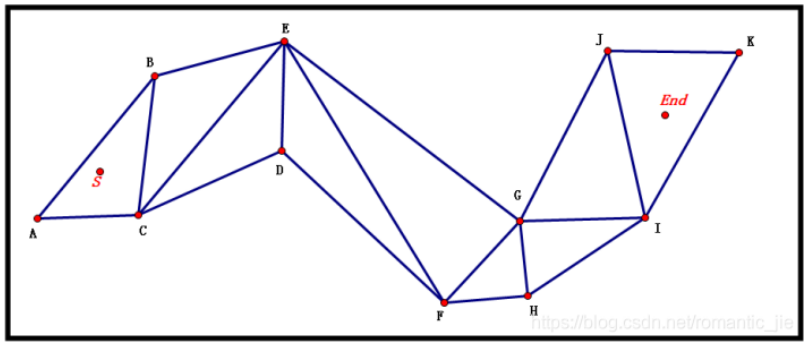
    - 然后把pathTriangles里面的顶点进行替换
- 还得**获取障碍物信息**：
    - 先获取所有的三角形信息

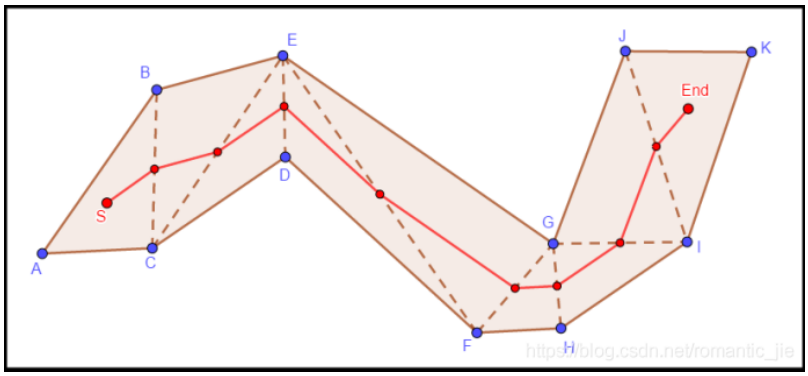
```
var vertexIndexs:int[] = _navMeshData.GetPathTriangles();
var vertices:MyVector3[] = _navMeshData.GetPathVertices();
int triangleIndex = 0; // 三角形下标
int length = vertexIndexs.Length - 3;
for (int i = 0; i <= length;) {
    int aIndex = vertexIndexs[i++];
    int bIndex = vertexIndexs[i++];
    int cIndex = vertexIndexs[i++];
    try {
        Triangle triangle = null;
        if (scale != 1) {
            triangle = new Triangle(a:vertices[aIndex], b:vertices[bIndex], c:vertices[cIndex], triangleIndex++,
                params vectorIndex:aIndex,
                bIndex, cIndex);
        }
        else {
            triangle = new Triangle(a:vertices[aIndex], b:vertices[bIndex], c:vertices[cIndex], triangleIndex++);
        }

        _triangles.Add(triangle);
    }
```

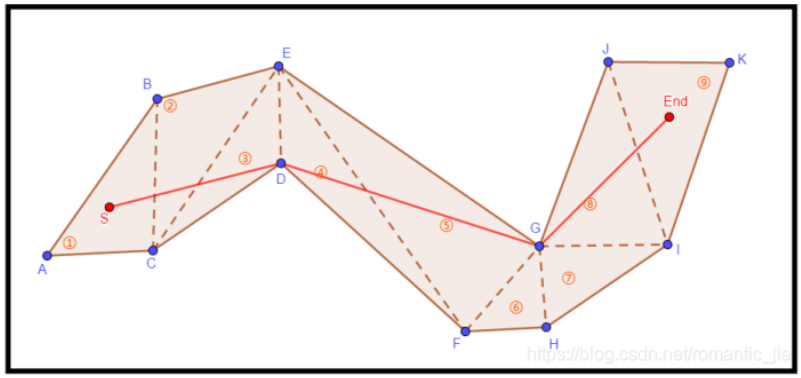
    - 然后获取所有的边
    - 只要有边跟其他的边完全重叠，那就把这个边跟所有重叠的全部去掉
    - 剩下的就是不重叠的边，也就是障碍物的边
    - 接下来生成polygon，根据边的顶点首尾相连去组成多边形
    - 获取每个多边形的面积，去掉面积最大的即可
- 构建**BSP树**
    - 又名**二叉空间分割**，存储的主要是一个前后的关系
    - 如何构建?
        - 1.遍历所有的边和所有三角形之前的关系，取得左右以及在平面的三种关系，然后累加，算出最合适的边作为分割线
        - 2.这样，一部分三角形在左边，一部分在右边，剩下的在分割线上，然后根据这个分割线，把剩下的三角形进行切割，分别放到左边和右边
        - 然后从左边和右边继续进行第一步，递归
        - 递归到左右各1个且分割线上没有的时候的时候结束
    - 构建BSP树的操作是离线的
    - 这种我们可以在运行时非常迅速的知道某个点在哪个三角形内
- 寻路
    - A*寻路
        - *A*算法总结(Summary of the A* Method)



- 1. 把起点加入 open list
- 2. 重复如下过程：
  - a. 遍历 open list ， 查找 F 值最小的节点，把它作为当前要处理的节点
  - b. 把这个节点移到 close list
  - c. 对当前方格的 8 个相邻方格的每一个方格?
    - 如果它是不可抵达的或者它在 close list 中，忽略它。否则，做如下操作
    - 如果它不在 open list 中，把它加入 open list ， 并且把当前方格设置为它的父亲，记录该方格的 F ， G 和 H 值
    - 如果它已经在 open list 中，检查这条路径 (即经由当前方格到达它那里) 是否更好，用 G 值作参考。更小的 G 值表示这是更好的路径。如果是这样，把它的父亲设置为当前方格，并重新计算它的 G 和 F 值。如果你的 open list 是按 F 值排序的话，改变后你可能需要重新排序
  - d. 停止，当你
    - 把终点加入到了 open list 中，此时路径已经找到了，或者
    - 查找终点失败，并且 open list 是空的，此时没有路径
    - 保存路径。从终点开始，每个方格沿着父节点移动直至起点，这就是你的路径
- 启发式函数
  - 2个三角形，取每个边的中点，然后取2个三角形的2个点之间最小距离作为最终值
- 最后可以找出起点到终点的一串相连的三角形
- 找最优路径：
  - 拉绳算法(漏斗算法)
  - 如下图







- 首先要严格分出左右边
- 然后再进行算法取路径点