RECAST探究文档

*日期：2021.02.18*

*作者：recast小分队*

目录

[RECAST探究文档 1](#_Toc16409)

*[日期：2021.02.18](#_Toc6413)* [1](#_Toc6413)

*[作者：recast小分队](#_Toc26822)* [1](#_Toc26822)

[一、 Recast生成 3](#_Toc4167)

[二、 基础路径求解算法 4](#_Toc8819)

[1． 迪杰特斯拉 4](#_Toc11221)

[2． 最短优先算法 4](#_Toc555)

[3． A-Star算法 4](#_Toc20774)

[三、 优化算法 5](#_Toc12782)

[（一） String Pulling 5](#_Toc7004)

[1． 拉绳算法概述 5](#_Toc7407)

[2． 算法详解 6](#_Toc31504)

[3． 算法详细步骤 10](#_Toc8908)

[4． 算法思考 10](#_Toc25899)

[（二） Raycast 11](#_Toc12679)

[1． 算法说明 11](#_Toc4471)

[2． 算法流程详解 12](#_Toc25674)

[3． 算法核心 13](#_Toc3805)

[4． 数据结构参考 23](#_Toc30005)

[5． 算法思考 24](#_Toc25191)

[四、 寻路算法 25](#_Toc721)

[（一） STRAIGHT 25](#_Toc20353)

[1． 方法概述 25](#_Toc28121)

[（二） FOLLOW 25](#_Toc6923)

[1． 基础寻路 25](#_Toc31344)

[2． 找引导点 26](#_Toc26687)

[3． 找目标点 26](#_Toc30730)

[4． “沿着表面移动” 27](#_Toc1395)

[5． 路径拼接迭代 28](#_Toc3466)

[6． 小结 29](#_Toc28735)

[（三） SLICED 30](#_Toc29541)

[1． 方法概述 30](#_Toc21396)

[2． 算法优化 30](#_Toc29644)

# Recast生成

# 基础路径求解算法

### 迪杰特斯拉

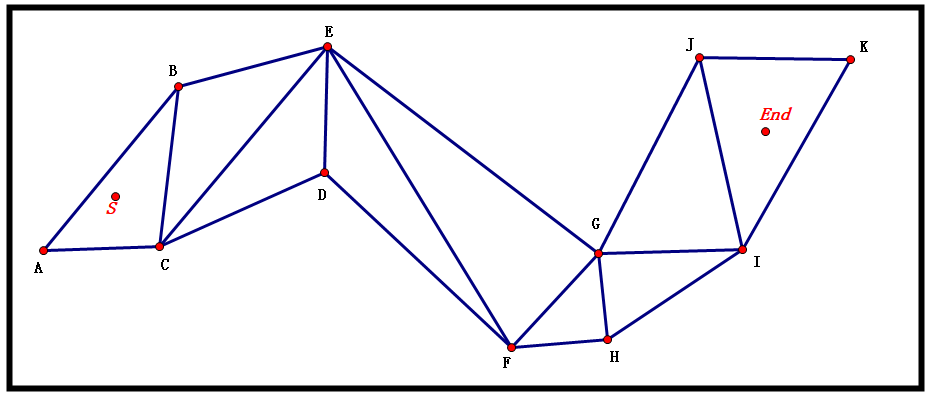
### 最短优先算法

### A-Star算法

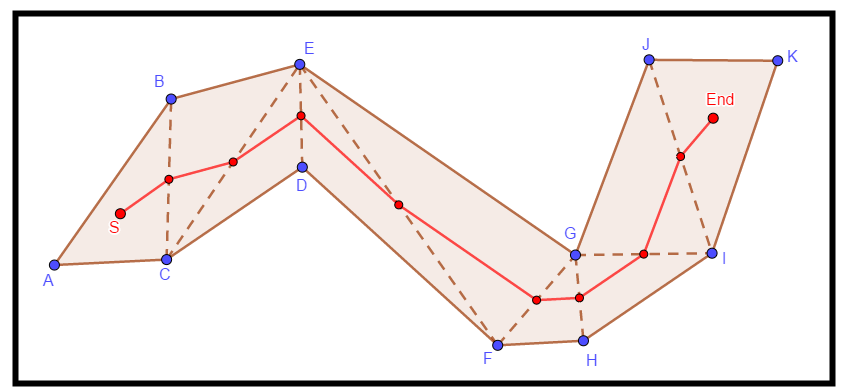
# 优化算法

## String Pulling

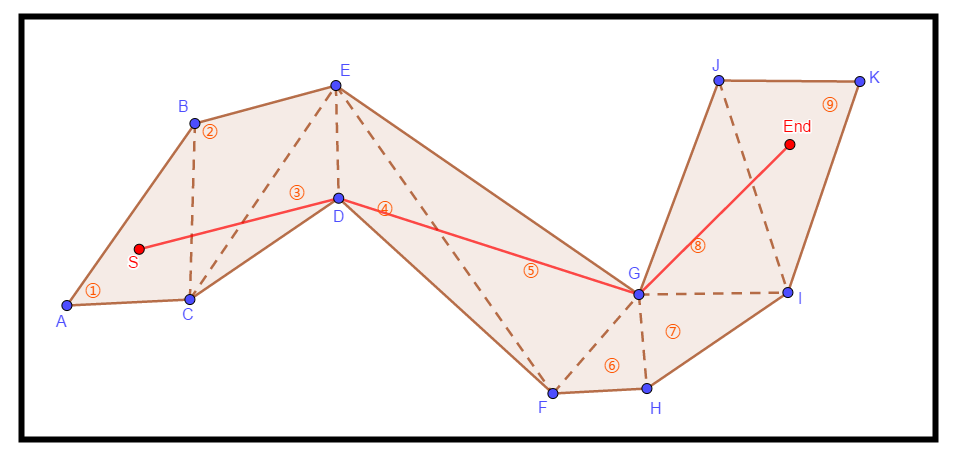
### 拉绳算法概述



上图为三角形导航网格，S为寻路起点，End为寻路终点。



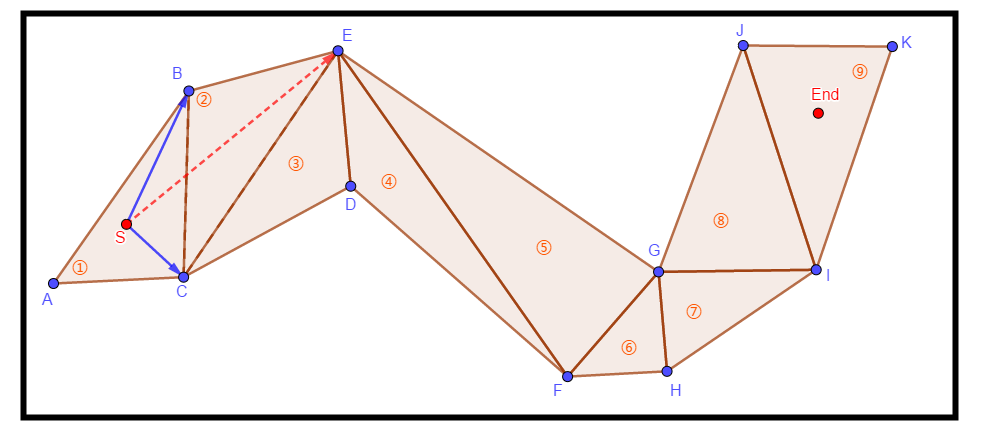
该图为基础A星算法寻路后的多边形路径，其中A星算法是以三角形边中点作为距离运算的迭代点的。



该图为漏斗算法优化后的路径点，最终为S，D，G，End。可以明显的看出基础A星在非规整导航网格中的路径结果是不平滑的，有种左右摇摆的感觉，而漏斗算法就是解决路径平滑问题的。

### 算法详解

所谓漏斗，就是设定两个边界包围向量，然后去迭代路径多边形，如果迭代的过程中，这两个边界包围向量可以不断缩小，证明这些路径多边形是可以直线联通的，这就是所谓的拉绳。而一旦出现了边界向量的交叉，则证明出现了拐角点。

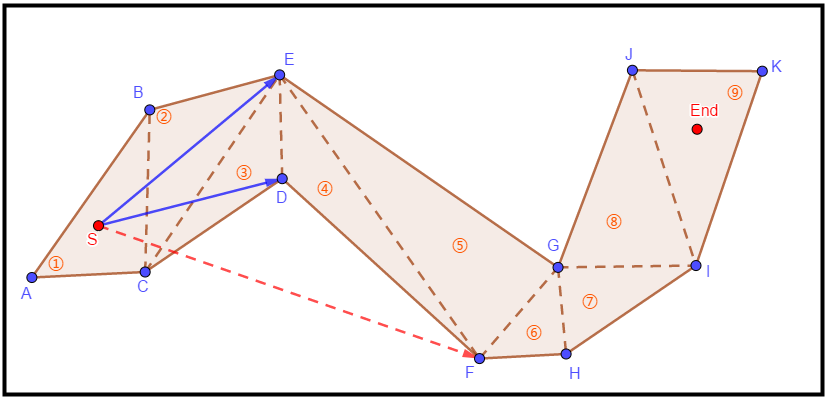


* S为起始点，序号①到⑨为A星寻路的结果多边形路径。End为终点。
* 初始多边形为①，下一个路径为②，①②之间的连接边是BC。
* 初始两个空间范围向量，图中的SB，SC。为了方便，下面称SB为“范围左边界”，SC为“范围右边界”。
* ②和③的临边是CE，构造SE，发现SE在“范围左边界”的右侧，且同时SE在“范围右边界”的左侧，此时更新“范围左边界”为SE。

#### 判断左右边界

“范围左右边界”并不是随意定义的，是需要遵循规则的。在DETOUR中，一个多边形的边采用的逆时针存储方式，例如多边形①，三点的存储方式为，CBA，这是严格控制的。而两个边界向量，在区分左右边界的时候，是根据两个相邻多边形公共边的方向顺序来定的。

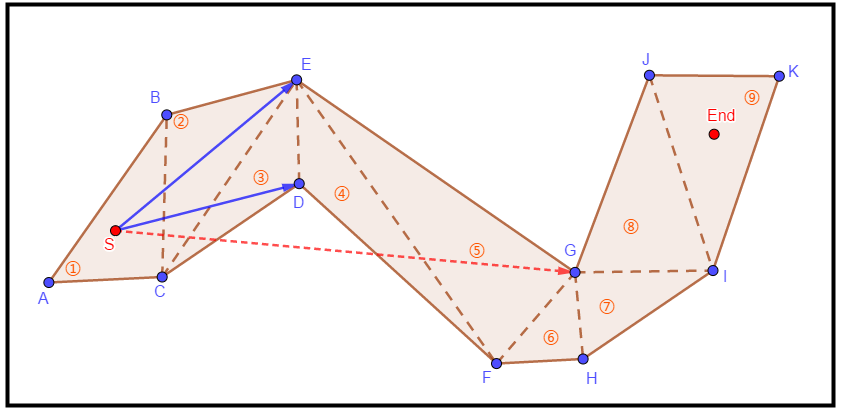
* 确定左右边界的时候，是根据公共边在“目的多边形”中的边端点顺序来的。
* 从①到②，①就是“起始多边形”，DETOUR中称为“From”，②就是“目的多边形”，DETOUR中称为“To”。
* 例如，①②的公共边为BC，BC在②中的定义顺序，按照逆时针来，B就是左边界点，C就是右边界点。同理，②③的公共边为ED，ED在③中的逆时针顺序为，E就是左边界点，D就是右边界点。



* 上图是处理多边形④到⑤的漏斗情况。
* 根据之前的规则，公共边为EF，且E为左边界点，F为右边界点。
* 在经过③到④后，此时的左边界为SE，右边界为SD
* 所以此时漏斗算法，比较SE和SE，SD和SF

#### “更新左右边界”规则

* SE和SE是重合的，不需要任何修改，这个很容易理解。
* SF在SD的右侧，同时F也是右边界点，测试不需要更新。

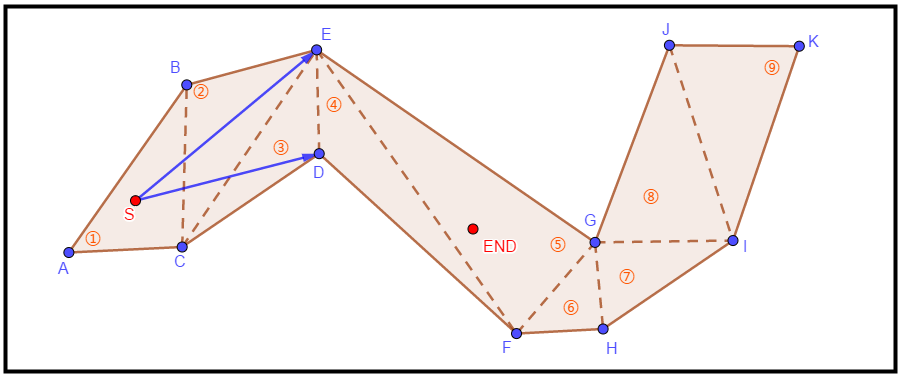


* 此时是⑤到⑥的漏斗处理：公共边为GF，且G为左边界点，F为右边界点。
* 此时对比SE和SG，发现SG在SE的右侧，且已经越过了右边界SD的范围，那么说明在A星结果路径的约束下，S到G之间是不能直线联通的，此时就产生了拐角点D。

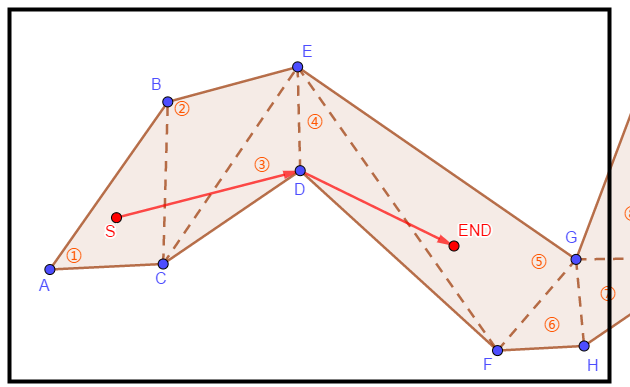
当找到拐角点之后，将拐角点加入到结果路径点集中，且以当前拐角点为起始点，重复上述过程，直到找到终点。总结规则如下：

* 已经确定好的左右边界向量，称为“已确定左右边界向量”。
* 算法迭代过程中，需要新对比的左右边界点所形成的向量，称为“对比左右边界”。
* 当“对比左右边界”在“已确定左右边界”包围范围内的时候，更新“确定左右边界”为“对比左右边界”。
* 当“对比左边界”在“已确定左边界”的左侧。或者，“对比右边界”在“已确定右边界”的右侧时，此时，无需更新。
* 当“对比左边界”在“已确定右边界”的右侧。或者，“对比右边界”在“已确定左边界”的左侧时，此时，产生拐角点。

#### 终点特殊处理



* 图中END为终点，在多边形⑤内。
* 在经历多边形④到⑤的漏斗计算后，此时的“确定左右边界”为SE，SD。



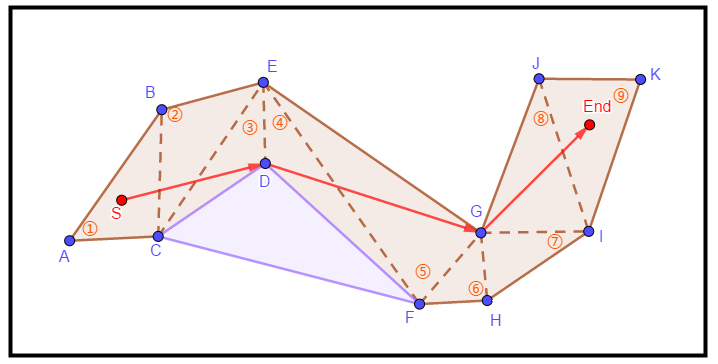
* 如图，最终结果，D肯定是一个拐角点，因为不存在S到End之间有联通直线路径。
* 但是，上图说明从④到⑤漏斗计算后，上下边界仍旧是SE和SD。
* 此时，算法在多边形⑤中找到End终点的时候，会同时设定End点为新的“对边左边界点”和“对比右边界点”。即SEnd向量同时为“对比左右边界向量”
* 按照漏斗算法流程，SEnd作为“对比左边界”时，其在“确定左边界”SE的右侧，且在“确定右边界”SD的右侧，所以D被解析为拐角点。

### 算法详细步骤

* 依赖上层寻路的多边形结果集合。例如例子中的多边形①到⑨。
* 得到①②的公共边，进而得到“左右边界点”。以当前起点为向量起点，“左右边界点”为向量终点，构造出两个“左右边界向量”。
* 继续向下迭代，得到②③的公共边，进而得到新的“左右边界点”，仍旧以起点作为向量起点，构建两个新的“对比左右边界向量”。
* 对比最新的两个“对比左右边界向量”和之前的两个“左右边界向量”，去更新迭代到当前多边形后确定最新的漏斗的两个边界向量。同时可能会产生拐角点。
* 一旦出现拐角点，则将当前拐角点加入到结果路径点集中，同时以当前拐角点作为新的迭代路径起点，重复上述的过程。
* 所有细节规则参考上述的算法详解。

### 算法思考

从算法的角度思考，漏斗平滑路径是完全基于上层寻路算法的，且依赖于上层寻路算法的多边形寻路结果。如下图所示：

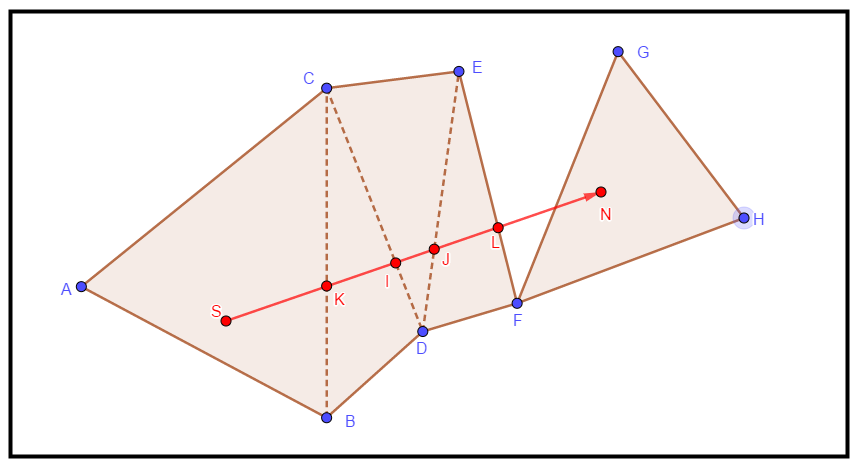


* 假设局部导航网格中三角形CDF也是可以直接通过的。
* 但是上层寻路算法从S到End的寻路路径仍然是①到⑨。
* 寻路之后的“漏斗算法”平滑依然是：SD，DG，GEnd。
* 但是，可以直观的看出，从S到G是完全可以直线通过，而不需要拐角点D的。

造成上述问题的原因，并不是“漏斗算法”本身有问题，而正是证明了“漏斗”平滑是完全依赖于上层寻路结果的。多边形CDF可以通过，但是上层寻路的结果集中没有该多边形，所以“漏斗”也不可能去主动搜索上层寻路中没有出现的多边形。由此可见，“漏斗”平滑在广义上是有局限性的。

## Raycast

### 算法说明

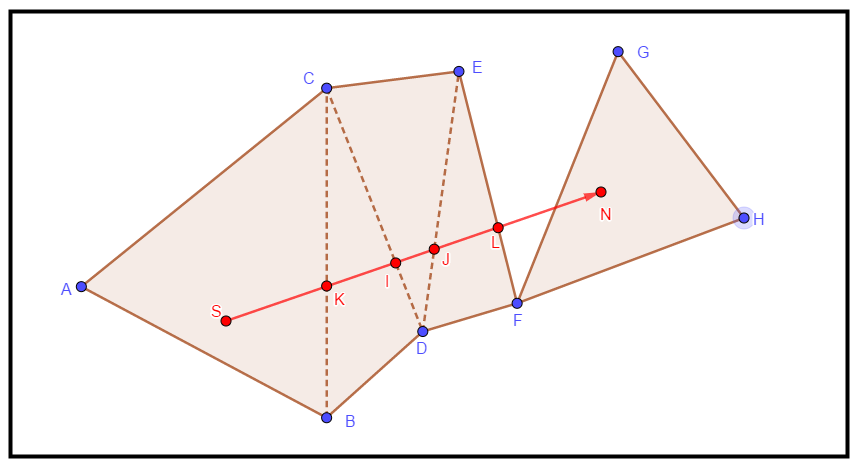


该图为俯视图，所有信息均为三维空间在底面上的二维投影

算法的目的是检测起始点S到终止点N之间是否可以直线通过，并且求出射线SN所经过的多边形（下述称“路径多边形”），进而得到射线SN与路径多边形的相交边（相交边分为：“进入边”和“穿出边”），并且求出与每一条相交边的交点坐标（由于某些需求使用，该交点需要得到其三维信息）。如果点S到点N之间不存在直线通过，则需要求解出第一个不可通过的相交点。

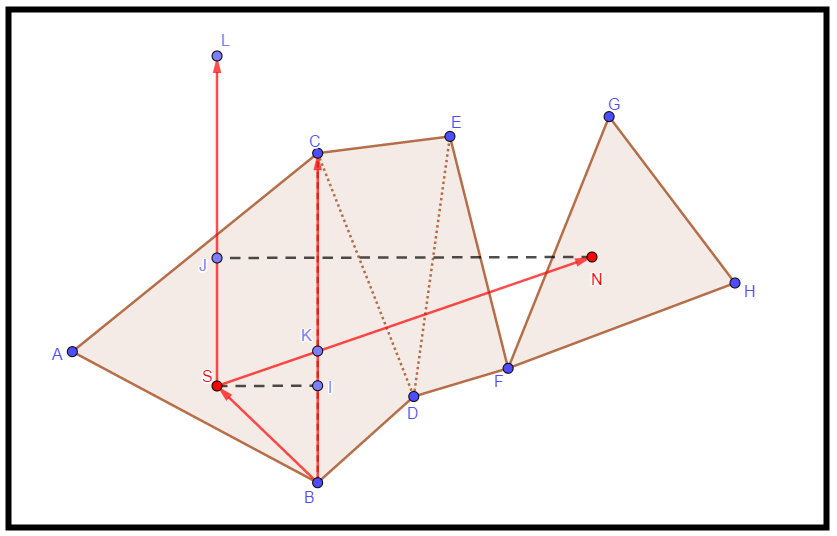
例如，三角形ABC、BCD、CDE、DEF均为路径多边形，边BC、CD、DE、EF均为相交边，且EF为边界边（非连通，没有邻居），点K、I、J、L为相交点，且L为射线探测的终止点。其中对于射线SN和三角形BCD，边BC为进入边，边CD为穿出边。

### 算法流程详解



* 算法开始的时候，起点S所在的多边形为“初始多边形”，以该多边形作为第一次算法的“迭代多边形”。
* 遍历“迭代多边形”的所有边，分别计算每一条边的与“目标射线”的交点情况。
* 上述计算后，会得到射线SN与当前多边形最多两个交点，而沿着射线方向，进入多边形的那条交点所在的边称为“传入边”，另一条边就是“传出边”。例如：SN与多边形BDC相交于边CB和边DC，沿着SN的方向，CB就是“传入边”。
* 此时射线SN与当前迭代多边形的运算就结束了，继续寻找下一个需要迭代计算的多边形。寻找的方式很简单，上一个多边形“传出边”所连接的多边形，就是下一个要迭代进入的多边形。例如，计算完SN与多边形BCA的交点情况，会得到一条“穿出边BC”，而多边形BCA以BC作为连接的多边形就是BDC，所以BDC就是下一个迭代多边形。我们称BDC为BCA以BC为公共边的“邻居多边形”。
* 寻找“迭代多边形”会有一种特殊情况，就是当前多边形的“穿出边”为公共边没有邻居多边形。例如，多边形EDF在边EF上没有多边形，而EF又是射线SN在多边形EDF上的穿出边，此时说明从起点到终点没有射线可以直接到达，算法结束。
* 还有一种情况，也会结束算法，即线段SN与当前多边形的交点计算结果后，发现，只有“传入边”，没有“传出边”。此时表明找到了终点N，因为只有射线SN不断迭代进入到终点N所在的多边形，才会出现这种情况。
* 以下一个迭代多边形重复上述过程。

### 算法核心



* 该图为俯视图，所有信息均为三维空间在底面上的二维投影
* 点S、N分别为射线求解的起始点和终点
* 向量SL是向量BC沿BS的平移结果，二者空间意义一致。
* SL是点S向BC做的垂线，NJ是点N向SL做的垂线
* 核心计算如下：

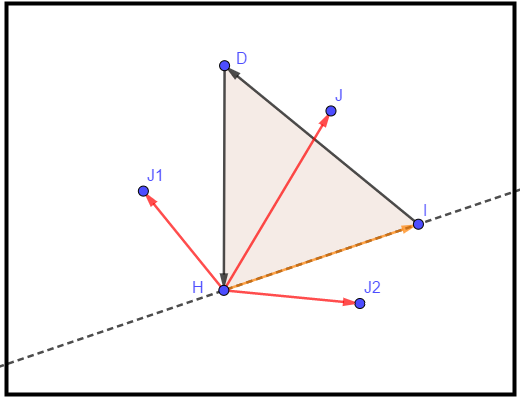
****

#### d值的作用

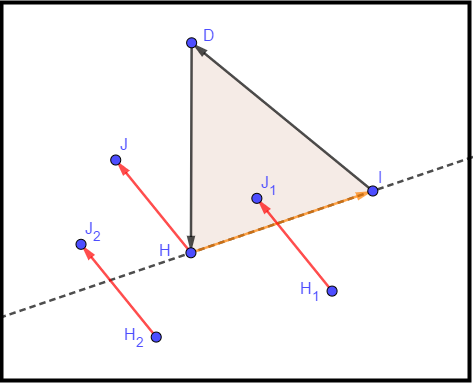
* d小于0，当前边要么为“进入边”，要么无交点。
* d大于0，当前边要么为“穿出边”，要么无交点。
* d等于0，当前边与射线SN平行，直接忽略。

##### 为什么d值可以决定“传入边”和“穿出边”：

* 多边形是以逆时针存储顶点信息和边信息的。
* d值是“目标向量”SE和“边的逆时针向量”的叉乘结果。数学意义上，叉乘结果的正负关系，代表了两个向量的空间位置关系。

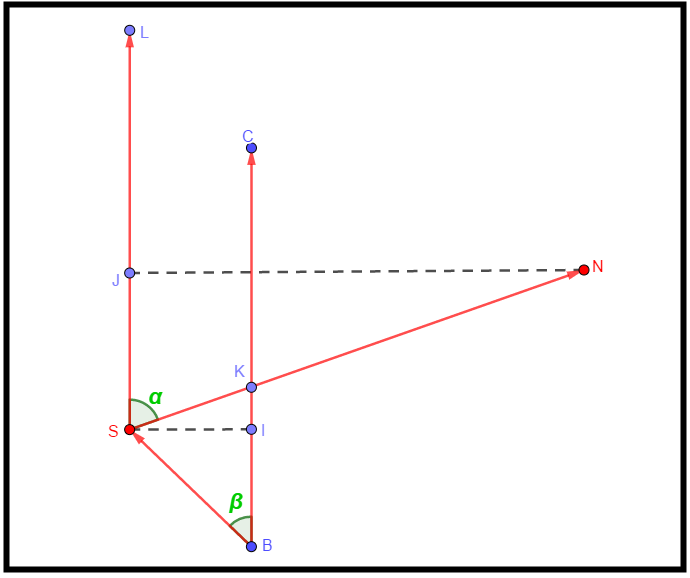


* 根据上图来判断空间中任意向量（下述简称“向量X”）和多边形任意一条边的“传入穿出关系”（图中以边HI作为例子，其他边理论一样）。
* 可以明显的看到，其实多边形自身的所有范围区域一定是在某条具体的边向量的一侧。并且，“传入边”的空间意义，其实就是从不属于多边形范围的空间，进入到属于多边形范围的空间。
* 根据上述的前置条件，会发现如果向量X与多边形的某条边是“穿入关系”。那向量X平移到与多边形边起点相同的时候（向量在空间中的任意平移，不会影响向量间的位置关系和数学运算结果），其向量X的终点指向一定是在边向量的同一侧。这就证明了，任意与当前多边形边HI构成“穿入”关系的空间向量，一定是在该边向量的同一侧，而这个空间关系恰好就是向量叉乘计算的空间意义。
* 如图中，很清晰的就可以看到。对于向量HJ和HJ1来说，边HI就是“穿入关系”。同理，对于HJ2来说，边HI就是“穿出关系”。



* 图中，向量HJ，H1J1，H2J2是同一个向量，可以相互平移得到。
* 构成“穿入穿出关系”不一定就是“穿入穿出边”。因为“穿入穿出边”还需要一个非常重要的信息，就是两个线段在空间中还必须有交点。
* 例如向量H2J2，其和向量H1J1是一样的，但是以线段的角度来看，H2J2与HI压根都不相交，所以也就不是“传入边”了。
* 到此，证明了运算值d在实际算法中的具体空间意义。

#### t值的作用

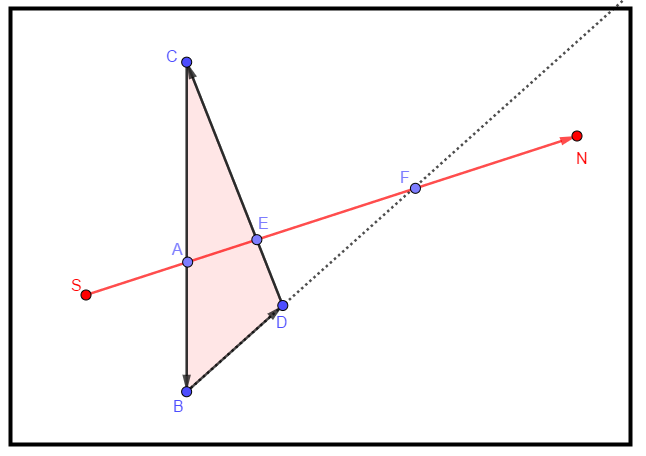


求解SK占SN的比例系数：



到此可以看出：t值代表的交点的一种特殊比例关系。

#### t的特殊情况



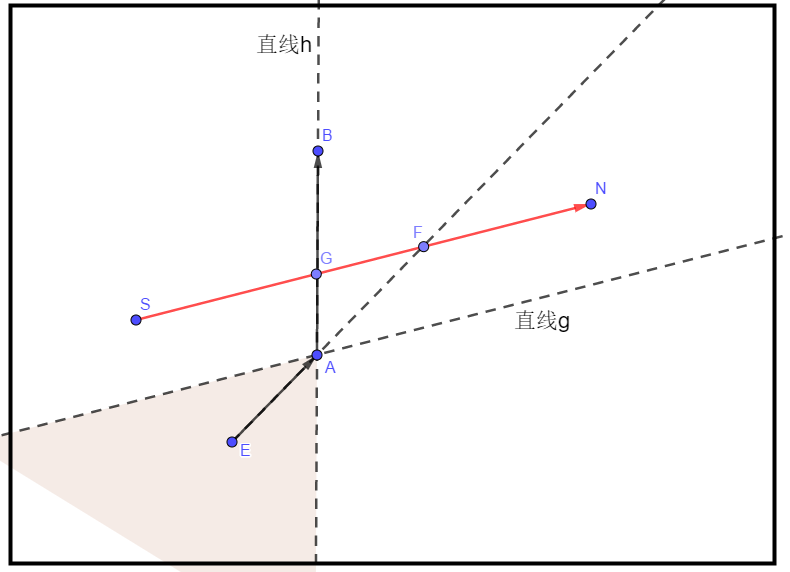
* 上图求解SN与多边形DCB的交点情况。S起点，N终点。
* 从实际情况来看，期望的算法结果为：边BC是“进入边”，点A是与其的交点。边CD是穿出边，点E是与其的交点。
* 上面对t的计算，实际空间意义是：两条向量所在两条直线的交点。例如，实际边BD与射线SN是没有交点的。但是直线BD与直线SN会交于点F，此时t值的实际意义就是SF占SN的比例。

##### 多个“穿入穿出关系边”获取“穿入穿出边”

多边形出现多个边向量与射线向量有交点的时候，其t值大小决定了哪一个是真正的交点边。（这一步就把上面根据d值判断“穿入穿出边”时候无交点的边排除掉了）

其实上图就能明确的看到，在计算SN和多边形BDC的穿出边的时候，DC和BD两条边都具有“穿出关系”。且直线DC与SN实际交点与点E，直线BD与SN实际交点于点F，但是点F不在边BD上。

* 对边DC计算t值，结果是SE长度占SN长度的比例
* 对边BD计算t值，结果是SF长度占SN长度的比例
* 可以证明，对于多个“穿出关系”边，t值小的那个是真正的有交点的“穿出边”
* 同理，对于多个“穿入关系”边，t值大的那个是真正的有交点的“穿入边”

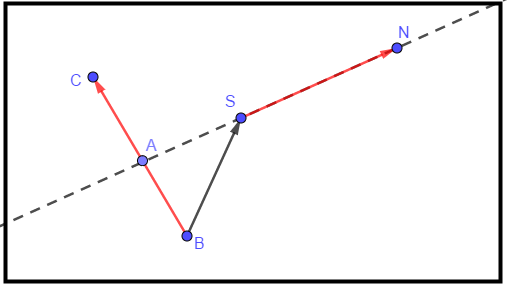


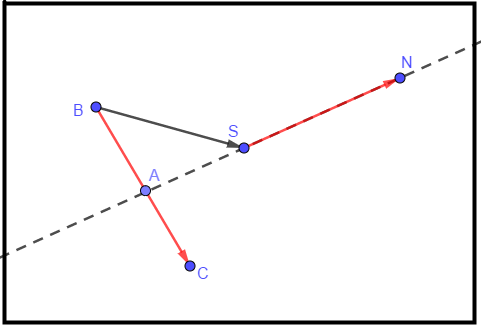
* SN是射线算法的起点和终点。
* AB是一条实际的穿出边，与SN交点与点G。
* 如果在多边形中还有其他边与SN有“穿出关系”。那么这条边一定满足以下条件：（例如，图中的边EA）
  + 边是逆时针排序，所以EA一定在AB所在直线h的左侧。（此处是以EA去逆时针连接AB为例，AB逆时针连接BE证明思路一样）
  + EA与射线SN存在“穿出关系”，所以EA一定由直线g的下侧穿向直线g的上侧。直线g是射线SN的平行线。因为：既然SN与EA存在“穿出关系”，那么SN一定是由EA的内侧穿向EA的外侧。反过来，EA就是由SN的外侧穿向SN的内侧。
  + 所以证明任意“穿出关系”边EA，（注意，是任意的，图中只是EA做举例）。其起点，一定是在图中E所在的橙色区域内，这个区域是上述两个条件的空间交集区域。
* 而当在橙色区域内的任意点E向A所做的射线，与SN交点F，这个F在SN上的位置一定在点G后面。因为点F一定在直线h的右侧。
* 所以SF占SN的比例一定大于SG占SN的比例。

##### t值范围过滤“无效边”

多边形边向量与射线向量产生交点，t值的有效范围必须是有约束的。

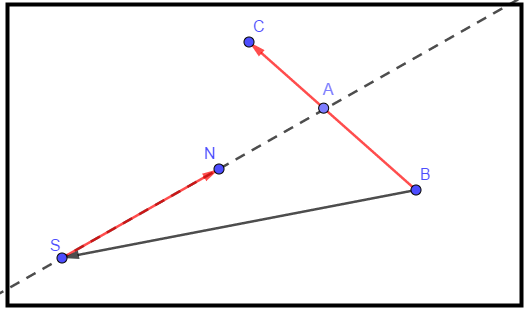
交点不在线段SN上，且交点在SN的反向延长线上





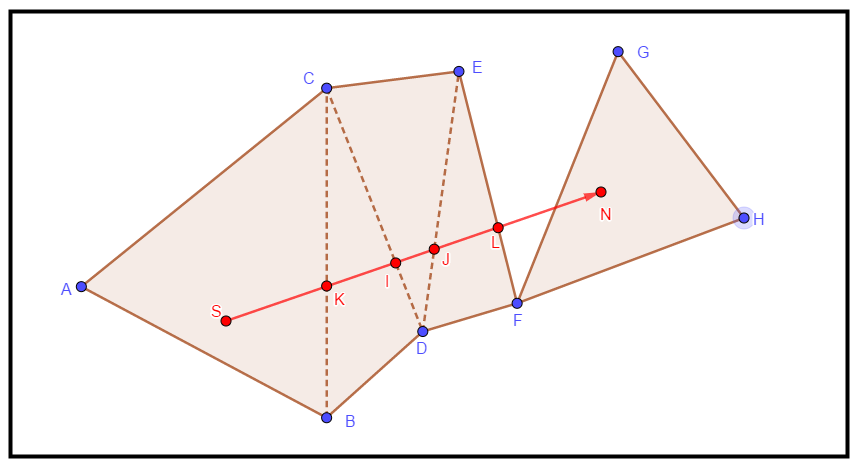
* 上图中，BC是多边形的一条边，SN是射线的起点和终点。
* 这种计算BC和SN的t值，会得到负数。也就是说，如果交点不在线段SN上，且交点在SN方向的反方向上，得到的结果就是负数。（直观理解：就是中间向量BS和向量SN一定在向量BC的同一侧。因为BC在SN的反方向上，所以构建BS的时候，BS一定是顺着SN的方向）
* 射线算法中，具有“穿出关系”的边，不会出现t小于0的情况。除非整个多边形根本与SN无交点（这种情况被射线算法本身排除了，射线算法在迭代的过程中不会处理任何与射线没有交点的多边形。且射线算法中采用的逆时针存储节点的凸多边形）。反证法很容易证明这一点。

交点不在线段SN上，交点在SN的正向延长线上



* 这种情况t值计算之后为正值，因为，SN和BS相对于BC的空间位置正好相反，所以t值表达式的分子分母的正负关系相同，其结果必为正值。
* 因为交点在SN正向延长线上，相当于BC在SN方向的前方。而BS的方向，又是以BC的起点为起点，SN的起点为终点，相当于从BC往SN的方向，这个方向必然与SN方向相反。
* 虽然t值为正值，但是，可以发现此时SA与SN的比例关系大于1。这也就说明计算结果t值大于1时，也是需要过滤的无效值。
* 同样的，射线算法中，具有“穿入关系”的边也不会出现t大于1的情况，除非整个多边形与SN无交点。反证法很容易证明。

射线起点和终点的特殊情况：

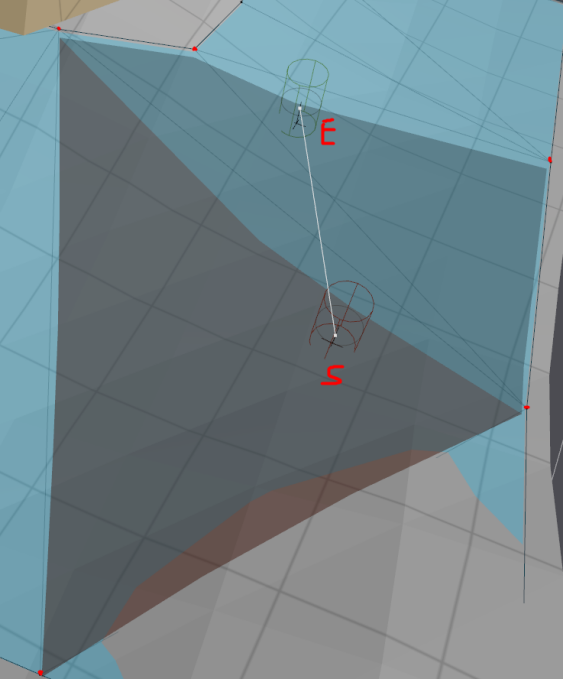
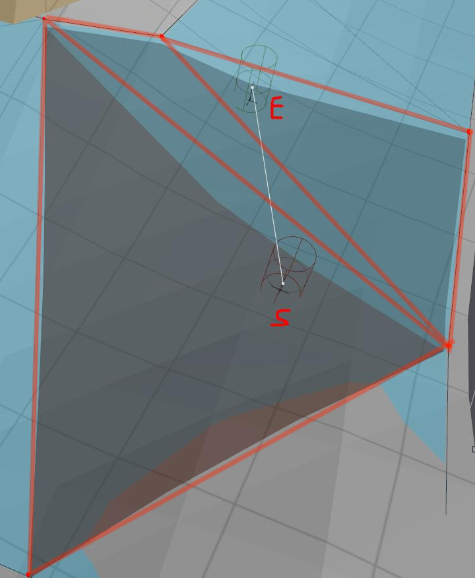


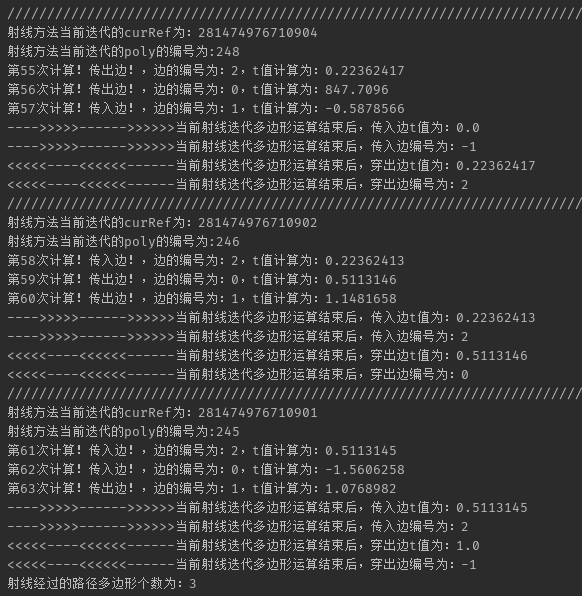
* 射线SN与初始多边形BCA无“穿入边”。此时默认穿入边t为最小值0。其实还有一种情况，就是S恰好在边BA上，此时t也是0。也就是将起点作为了一个初始交点。
* 而对于SN与最后多边形（假如上述多边形FEG是联通的）FHG无“穿出边”。此时默认穿出边t为最大值1。同样的，N恰好在HG上，t也是1。也就是将终点作为了最后一个交点。
* 而射线算法结果中还保留了相交边的索引，如果没有“穿入穿出边”，会有一个默认常量值标识这种特殊情况。

总结d值和t值的实际意义：

* d值首先得到射线SN与多边形每条边的“穿入穿出关系”
* t值首先需要过滤有效范围，[0,1]。小于0，大于1的都是无效值。
* 在得到有效t值后，对于多个具有“穿入关系”的边，t值最大为“穿入边”。对于多个具有“穿出关系”的边，t值最小的为“穿出边”。
* 确定“穿入穿出边”后，对应的t值就是一种线段比例关系。

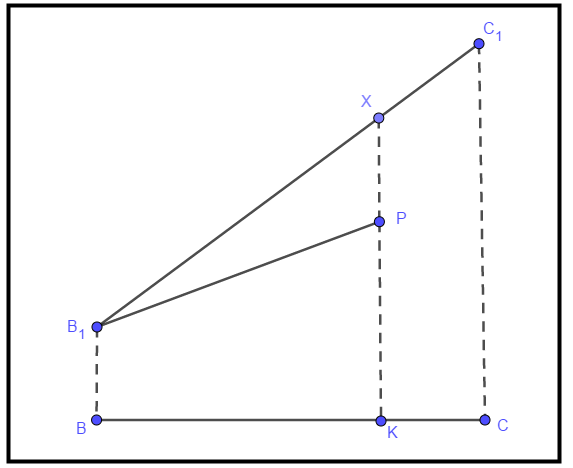
#### 根据Detour测试工具实际例子验证结果



#### 得到交点的高度信息

计算二维投影下的交点后，确定其在三维上的高度信息。这一步仅仅只是为了利用边交点的三维数据信息，来计算花销



解释说明：

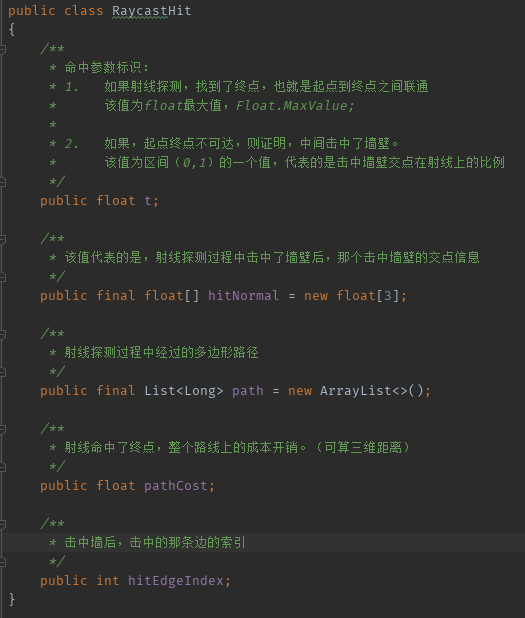
* 点B、C是二维投影下多边形的边的端点
* 点B1、C1是三维空间中多边形的边的实际端点
* 点K是二维投影下，SN(射线起终点)与边的交点(该点不需要计算)
* 点P是二维点K在三维SN上的垂直对应点(是之前已经求解的)
* 点X是最终求解，二维点K在三维边上垂直对应点(含高度信息)

求解点X：

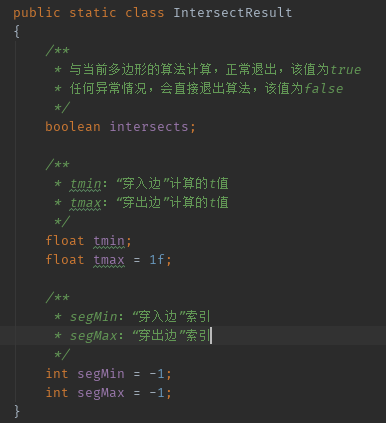


### 数据结构参考

#### 射线算法的结果数据



#### 射线与多边形计算相交关系的结果数据



### 算法思考

#### 有重复计算

很明显的，算法不断的往前搜索需要与射线计算交点关系的多边形。而往前迭代的连接逻辑就是：找到射线与当前多边形形成的“穿出边”，那么与当前多边形以“穿出边”构成邻居关系的多边形就是，下一个要迭代的多边形。

这就证明了：下一个搜索多边形的“穿入边”就是上一个搜索多边形的“穿出边”。且是同一个交点，那么相应的t值就是一致的。这就是重复计算

如果算法中是分开两次分别计算的这个t值。还会引起一个问题，t值本身就是一个比值关系，最后会被转化成小数，但是这个小数绝大多数情况下都是近似值，因为分数多数都是不能整除的。那么分开两次计算，很有可能造成两次结果有细微的不一致。

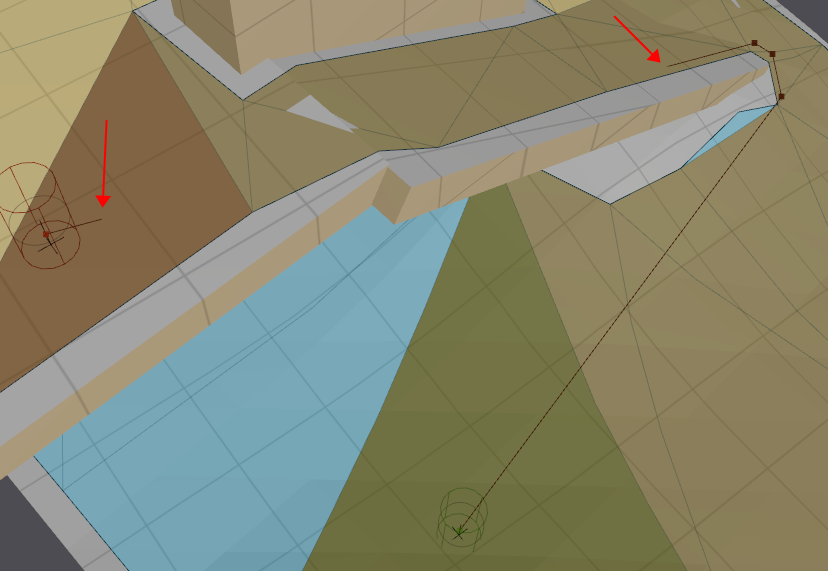
# 寻路算法

## STRAIGHT

### 方法概述

简单的说，该方法就是“拉绳”算法的应用。

上层基础寻路算法会获得寻路所得的多边形路径，该路径的计算都是以某一空间点来代替多边形进行数学抽象的，这就造成了最后的路径点极其“诡异”。而这种方式，就是上层寻路得到路径后，再进行一次完整的“拉绳”来实现一定的路径平滑。



考虑到三维空间问题：拉绳算法是以多边形路径的二维投影做的运算，得到路径点后直接获取其高度，会造成三维实际路径点的不合理，这是“拉绳”算法在三维空间上的一大问题。简而言之：二维运算两个拐角点间可以直线拉直，会直接忽略这两点之间的其他点。但是三维中有了高度信息，这些中间点就不应该被直接忽略。

## FOLLOW

### 基础寻路

上层寻路算法获得最基础的路径多边形信息。

### 找引导点

#### 部分“漏斗”

对当前迭代点（初始为起点）到终点，做“拉绳”算法。这一步，Follow中有一部优化，后续可以看到，其实并不是所有的“拐角点”都有用。所以Follow中这一步只是做了部分拉绳优化，默认是三次，也就是从起点开始，顺序得到三个拐角点，这一步就完成了。最终会得到三个拐角点信息。

#### 找“相对远”的点

在漏斗算法的结果点集（下面称之为拐角点）中，找一个距离迭代点“相对远”的引导点。“相对远”的定义条件如下，二者满足其一即可：

* 两点2D投影距离不小于一个设定好的距离常量（其实，这个设定好的距离常量是一个误差容错值，为0.01，即在误差容错距离内，认为迭代点和当前拐角点重合，那么去搜索下一个拐角点）
* 两点3D高度差不小于一个设定好的H高度差常量（其实，高度值是对上述距离误差容错的再一次容错，因为当前迭代点和拐角点在2D投影上再接近，其高度差过大，也就不能当做重合点看待了）

在漏斗算法结果点集合中满足上述条件之一的点有很多，所以顺序遍历结果点集，第一个符合条件的就是引导点的来源点A。不难发现，正常情况下，其实第一个拐角点就是引导点，除非第一个拐角点与迭代点在三维空间上几乎重合。

### 找目标点

Follow中有以下两个距离值：

* 一个设定好的默认常量值，STEP\_SIZE，每一步最大距离。
* 当前迭代点到引导点的距离Dist（三维距离）

目标点，就是这两个距离值比较后，距离较小的点。

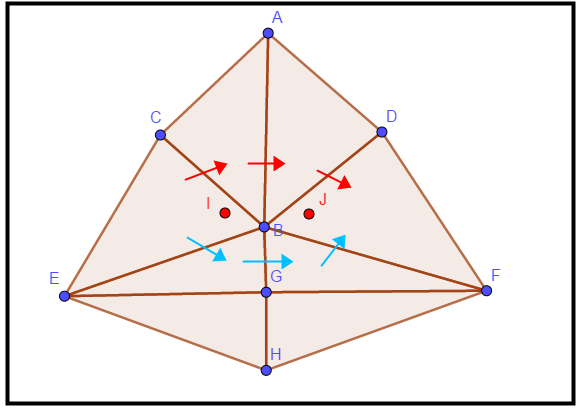
* Dist小于STEP\_SIZE，那么目标点就是引导点
* 否则，目标点为：迭代点沿着引导点方向走一步STEP\_SIZE之后的点

### “沿着表面移动”

#### 两种情况

* 迭代点（当前点）与目标点（走一步之后的那个点），同属于一个多边形网格内，那么不需要额外处理。
* 迭代点与目标点不属于同一个多边形网格中。此时做法是，根据边去搜索邻居多边形，确认是否可以从当前点经过联通的多边形路径到达目标点所在网格。如果存在这样的路径，那么也可以确认该目标点为结果路径中的一点。如果没有联通路径，算法会找一个距离目标点最近的边界边，且迭代点到该边界边是联通的。然后边界边上距离目标点最近的点作为结果点。

#### 局部寻路

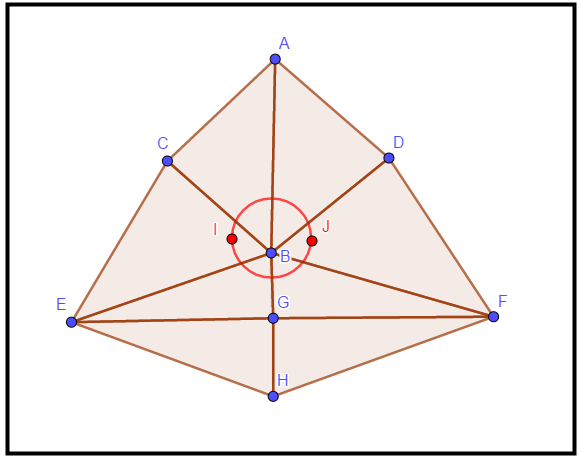


* 当前算法迭代点是I，J是确认的下一个目标点。
* 局部路径计算只关心I和J两点，不关心之前的所有计算。

算法思路：

* 维护一个多边形队列，一开始将初始多边形，也就是I所在的多边形加入队列。
* 从队列(先进先出)中取出多边形，遍历当前多边形的所有边所连接的邻居多边形，查看是否目标点J在邻居多边形里，如果在，算法结束，保存经过的邻居多边形的路径。如果不在，则将邻居多边形假如队列，作为下一次搜索的多边形。例如，BCA是初始多边形EBC的邻居，但是目标点不在BCA中，BCA加入到队列中，继续搜索邻居EGB，相同的处理。需要注意的是，将邻居加入到队列中的时候，是需要满足一定的条件的。
* 依照上述思路，由于采用了队列，所以最后搜索到目标点J的时候，返回的一条经过路径一定满足：其经过的多边形数量是所有路径中最少的。例如，图中红色和蓝色的指向，由于两者长度一致，所以谁先进的队列，最后计算的结果路径就是谁。

邻居多边形入队列的条件



* 算法是有优化的，并不是只要目标点不在邻居多边形中，就将邻居多边形加入到队列中。这一步有个限制，如上图所示：以IJ为直径做圆形区域，判断一个邻居多边形是否加入队列的条件，就是与邻居多边形的公共边，是否在这个圆的范围内。
* 例如：EBC的邻居BEG，公共边BE，BE在圆范围内，则EBG加入队列。假如当前算法从队列中取出了BEG，那么EGH是其邻居，公共边是EG，但是边EG不在圆圈范围内，则这个多边形直接忽略。
* 其实这个条件限制的做法是，取IJ的中点，计算该中点到公共边的距离xx，比较这个距离xx和IJ距离的一半，来验证边是否在圆圈范围内。
* 这一步限制的具体意义就是在迭代多边形加入队列的时候，过远的那些直接排除，忽略不做处理。

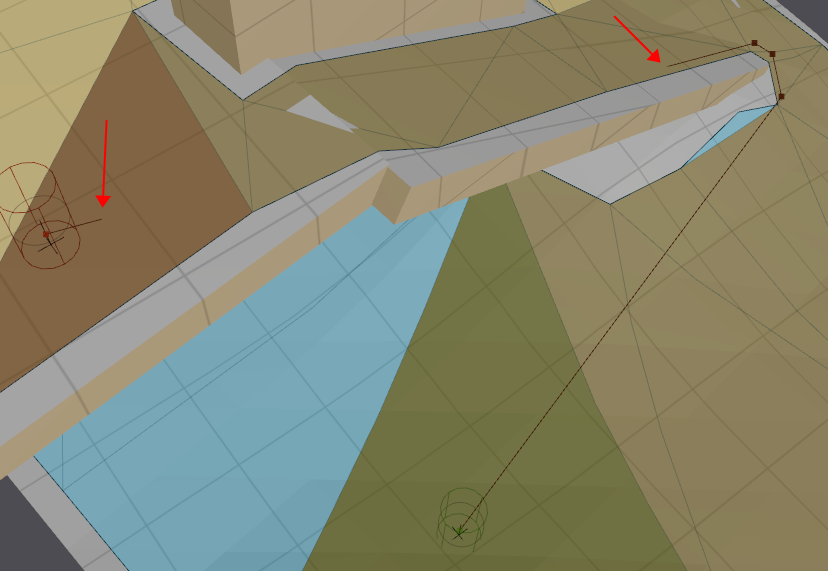
### 路径拼接迭代

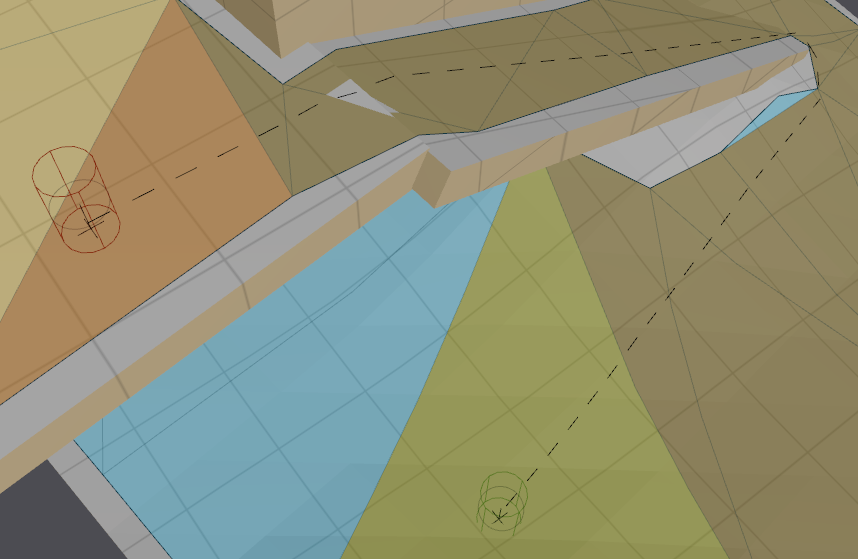
“沿着表面走”计算之后，会进行局部寻路，获得路径，如果存在一条路径，连接了当前点到目标点，目标点就是计算结果点集中的一点，同时会以目标点为算法下一次的迭代点去重复上述的过程。

既然目标点，作为算法最新的迭代点了，那么就说明起点到当前迭代点之间已经计算结束了，就需要把起点到终点的多边形路径集合中，关于从起点到当前迭代点的那一部分剪切掉，做个整合。这一部分还进行了一步路径搜索的优化。

简单的说，就是以刚才计算的结果点作为新的起点，重复上面的过程。由于涉及到路径的裁剪拼接优化，这一步可能就会形成新的路径。一旦形成新的路径，之前的“引导点”和“目标点”就需要重新计算，就连“漏斗优化”也需要重做。

### 小结





其实Follow算法的核心，就是解决了单纯的二维漏斗算法在三维空间中，增加高度信息后，会出现“空间穿过”的情况。算法的核心是去设置一个常量单位距离作为“迈一步”的距离，在路径多边形中去求解每一步的点，从而缩小误差。让路径看起来尽可能的是在“沿着导航网格的表面”走，从而解决“三维空间穿过”的问题。

## SLICED

### 方法概述

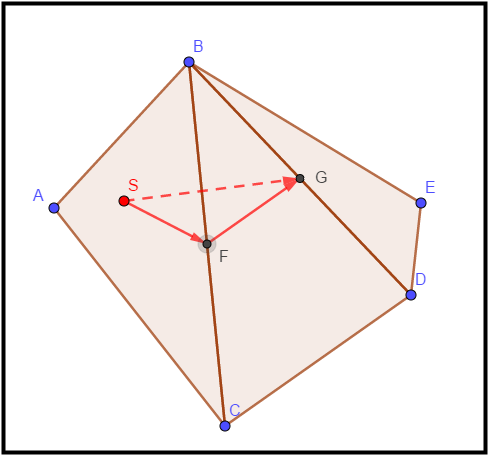
该方法被称为“切片”寻路方式，所谓“切片”。就是将一次完整的寻路切分，分成每次进行局部部分寻路，分多次进行最后获取完整的结果路径。

Detour中的demo举例：上层寻路方式为A星基础寻路，而A星作为搜索路径方式，是不断的迭代搜索下一个多边形进行的。在A星中有一个开放列表来维护这个迭代关系，一般是一个优先级队列（Open-List）。正常寻路都是直接进行一次完整的A星获取最终路径。而切片是每次从开放列表中取出指定数量的多边形搜索，每次只进行该数量的计算，但是只要寻路没有结束，开放列表是一直在公用且维护的。

所以说：切片寻路就是对A星算法的整体一次计算完成，拆分成了每次指定数量的部分寻路，可以理解为一种局部搜索方式。其主要作用主要有：减轻单帧完整计算的压力，动态阻挡时完整路径的不确定性等。

### 算法优化

由于切片采用的是局部搜索方式，所以算法中采用局部射线的方式优化了路径。



* 图中，正常A星寻路采用的是公共边中点作为多边形距离的计算。
* 当路径为多边形，ACB，CDB，DEB时，称ACB为CDB的前节点或者父节点，而DEB为CDB的next节点或者邻居节点。
* A星是从当前节点去搜索邻居节点的，定义当前节点与邻居节点公共边中点，作为邻居节点的计算点。例如：多边形CDB和DEB，其公共边BD的中点G作为DEB的计算点。同理点F作为CDB的计算点。
* 切片做的优化就是，确认当前多边形往邻居路径搜索距离的时候，以父节点的计算点向邻居节点的计算点打射线，如果中间无阻碍，射线成立，则直接以射线距离作为A星的运算距离，从而使得路径看起来更加平滑。
* 如图，正常A星是从S，F，G去做的计算。则对S到G做射线成功的时候，G点的A星运算距离，就是射线距离。考虑到A星运算是以三维距离做的，所以说，射线也必须能计算出实际三维距离，这就是上述“射线算法”中计算交点的一个重要作用。