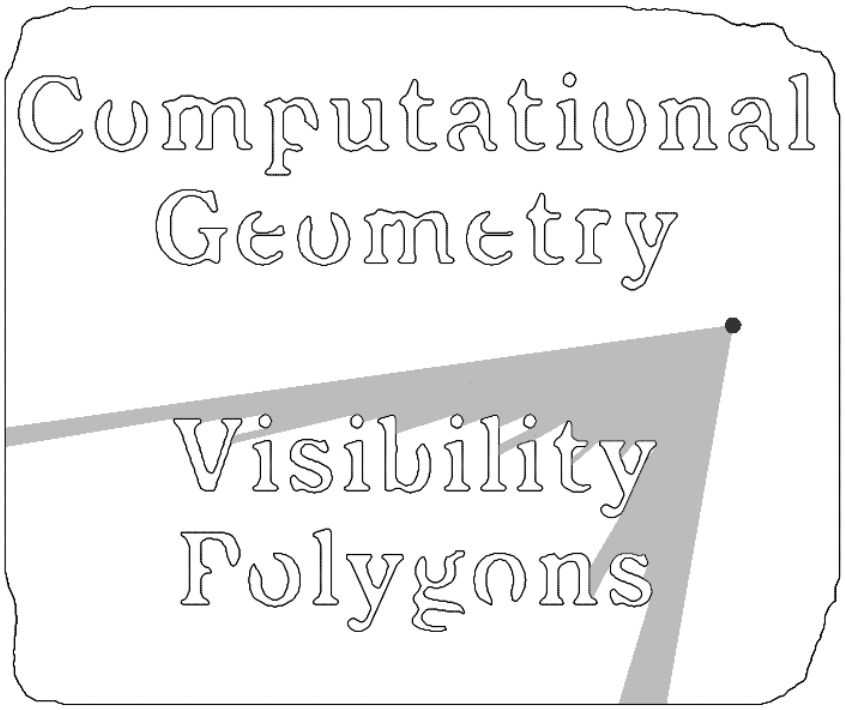
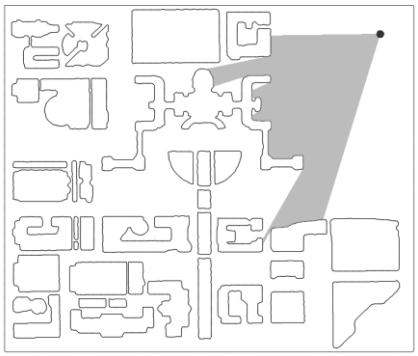
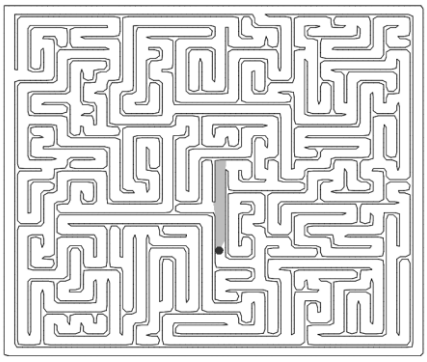
# 基于可见多边形生成算法的 PlanarSight 游戏

**罗必成，温佺，陈翔**

**清华大学软件学院**





## 1. 实验目的

我们组选择的课程实验选题是“基于可见多边形生成算法的 PlanarSight 游戏”。

PlanarSight 是这样一款小游戏，用户可以在窗口当中随意地绘制出一个带障碍物的多边形房间，并且在多边形房间当中布置一系列的随意走动的 Monster。然后游戏一开始，这些 Monster 就开始在房间当中巡视，他们有着相应的朝向和视角大小。然后玩家要用鼠标移动控制游戏的主角 Player，在不被 Monster 发现（也就是不在 Monster 的可视多边形范围内）的情况下逃离这个房间。 这个游戏的实现背后所需要的理论基础就是对平面含洞多边形中的某个点求其可见多边形，再判断某个点是不是在这个可见多边形内。与游戏联系起来，还会有动态更新，计算的需求。

通过这个项目的研发， 我们希望可以更好地对可见性问题产生深入的理解。 我们也希望通过这个小游戏，从寓教于乐的角度出发， 让大家对计算几何问题的研究产生兴趣。

## 2. 实验原理

如上文所述， 我们需要解决的最基本的问题是对于一个平面含洞多边形 P 和其中的一点 q， 求 q 关于 P 的可见多边形。 我们计划采用的算法是 Asano[1]， 算法需要 $O(n^2)$的时间和空间进行预处理（ n 指的是平面含洞多边形的边数），但仅需线性时间计算可见多边形。

下面将介绍算法的主要三个步骤，计算直线排布，计算三角剖分以及线性集合算法。

### 计算直线排布

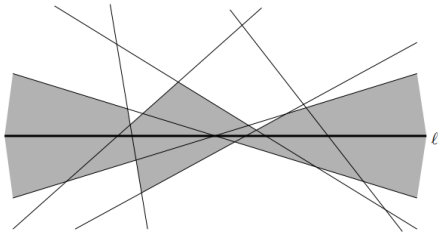
由于在后面的算法中需要计算多边形 P 所有顶点关于 q 的极角序， 并且一般的排序算法在 O(nlogn)时间内完成无法满足线性时间的条件， 因此在这里其参考了

CHazelle[2]中的算法。 其中将平面内的点和直线按一定关系转化为相应的直线和点，这样就将原先的极角（ q 与 P 顶点所在直线的斜率） 转化为变换后直线间交点的横坐标。 接着需要计算一组直线排布，如下图所示， 其中的直线由原先多边形 P 的顶点转化而来，我们可以在 $O(n^2)$的时间内计算它们的交点以及将其表示为 DCEL 的形式。 当加入一条新的直线 L（由 q 转化）时， 我们可以根据 DECL 表示中边之间的关系在线性时间内得到 L 与所有直线交点横坐标的有序关系， 从而得到了多边形 P 所有顶点关于 q 的极角序。

**DCEL 与对偶图**

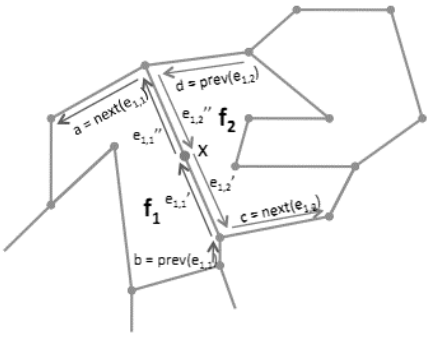
Doubly Connect Edge List (DCEL)是平面图的一种存储结构，对偶图 Arrangement 描述了平面内点和线的对应关系。实验中将建立平面内的点、线与对偶图中的线、点之间的联系，并通过 DCEL 的数据存储形式来实现快速查询其他点对于一个点的极角序的算法，时间复杂度为 O(n)（n 为点的数量）。

首先使用对偶图的办法将点转换成直线。在平面内，可以将一个点 $P_1(x_1, y_1)$转化为对偶图中的直线 $x_1*X+Y-y_1 = 0$，将另一个点 $P_2(x_2, y_2)$转化为对偶图中的直线 $x_2*X+Y-y_2 = 0$，联立两个方程，解得交点坐标 $P(x_0, y_0)$ 中 $x_0 = (y_2-y_1)/(x_2-x_1)$，即为平面内两个点的斜率（如果斜率不是无穷的话）。于是，平面内点与点之间的斜率关系就转化为对偶图中线与线交点横坐标的关系。当我们根据当前点转化成的对偶图中的直线，从“左”向“右”依次遍历整个对偶图时，根据与当前直线的交点的横坐标，即可得到一组点关于一个点的有序的极角序。而整个从“左”向“右”遍历的过程，则基于 DCEL 这一平面图存储结构，根据 Zone Theorem，一条直线在对偶图中穿过的直线数量为 O(m)，（m 为直线的数量），保证了构造和查询的效率。

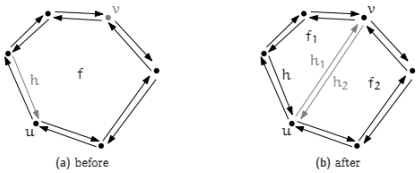



**Theorem:** The complexity of the zone of a line in an arrangement of $m$ lines is $O(m)$

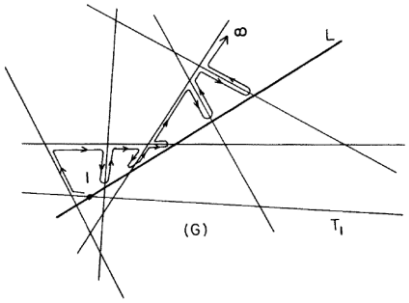
DCEL 是存储平面图信息的一种数据结构， 实验中对 DCEL 的操作中，有两个重要的基本操作，一个是在边上增加一个顶点，即边的分割：



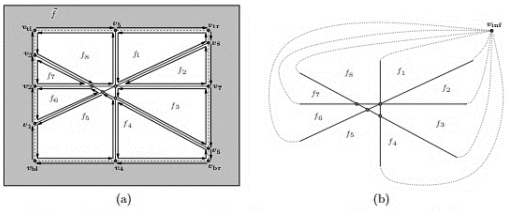

另一个是链接两个顶点，即面的分割：




两个重要的对外接口操作是在平面图中插入一条直线，以及按照从“左”向“右”查询一条直线穿过的所有的直线。查询操作与（1）中相关联，如下图所示：

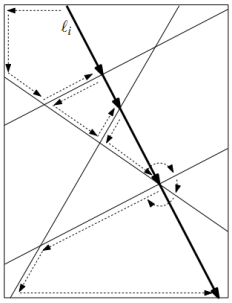



由于实验中 DCEL 存储的是对偶图中的直线，而直线是无边界的，故而需要特殊处理一下 DCEL 的存储结构，如图：



逻辑上的 DCEL 的形式应当为(b)图中的形式，而在程序运算过程中，为了更简单快捷地计算，会在所有直线的外围假想一个非常巨大的包围盒，包含所有直线的交点。

在预处理阶段，根据输入的地图信息，得到所有点，将所有点转化为对偶图中的直线，通过 DCEL 建立起平面图结构。插入和查询操作类似，过程




实时运行阶段，根据当前需要查询点转化为对偶图中的直线，到 DCEL 结构中按照横坐标序列从“左”到“右”查询所有直线，得到所有点关于当前点的斜率的关系。再根据获得的斜率关系以及点与当前要查询点之间的坐标关系来划分点的极角序为[0, π)还是[π, 2π)。

从而，将一个斜率有序的数组转化为两个极角有序的数组，分裂后，再按照极角序，重新排列起来。

有些情况需要特殊处理，如在对偶图中两条直线平行的情况，对应于在原二维平面内，两个点所在直线平行于 Y 轴的情况。此种情况下，对于这些点的斜率的判断不会在 DCEL 求解过程中，而需要在外部单独计算，好在垂直情况下，极角序只有 0 和 π 两种，根据计算的结果直接剥离开即可。还有就是 X 坐标相同时，按照 Y 坐标从小到大的方向去查询。

我们可以就直线排布的算法进行相关的算法分析。首先分析一下预处理阶段需要的时间和空间复杂度，如下图：

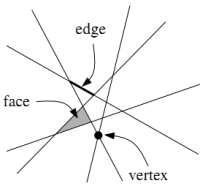
**Combinatorial Complexity:**

- $\leq n(n-1)/2$ vertices
- $\leq n^2$ edges
- $\leq n^2/2+n/2+1$ faces: add lines incrementally

$$1+\sum_{i=1}^{n} i = n(n+1)/2+1$$

- equality holds in *simple* arrangements

Overall $O(n^2)$ complexity




DCEL 结构中，点、线、面都是 $O(n^2)$的空间复杂度，时间复杂度也是 $O(n^2)$。1000 个点的测试输入，在 DCEL 存储结构中会有近 50 万个点，200 万条边，还有 50 万个面。当点数达到 5000 时，DCEL 存储结构中会有近 1000 万个点，4000 万条边，还有 1000 万个面。而在 DCEL 中，每一个点，边，面都是一个存储结构，还会存有其他的信息，对内存空间要求很高。

预处理阶段，相对比较耗时间，考虑到平台是 C++，大量的 new 操作会极大地降低预处理时间，我们采用了先批量 new 后再根据标号取用申请空间的方式来适当提高运行时速度。

运行时，考虑到此种算法空间复杂度需要 $O(n^2)$，有大量的信息常驻内存，时间复杂度为 O(n)，但是常数操作比较多。而如果采用简单暴力的排序算法，在计算极角序的时候直接调用 C++的 std::sort 的话，兴许会有不一样的效果，因为 sort 的时间复杂度，虽然近似为 O(nlogn)，但是排序的结构非常简单，常数也十分地小，而空间复杂度，则相比于 DCEL 的存储，也大大的降低，并且，很多信息并不需要常驻内存。于是，在程序中，我们同时实现了使用 std::sort 进行的极角排序和采用对偶、DCEL 来实现的极角排序。

实验通过手绘小数据，图像处理提取图片边缘转化为输入点的方式，对运行时程序进行 sort 和 DCEL 方法的运行时效率比较。结果发现，在很小规模的数据下（30 点以内 ）DCEL 会稍微比 sort 快一点，而到了几百几千的数据，DCEL 与 sort 的速度不相上下，运行时间很小。

## 三角剖分

在计算平面上可见多边形的整个算法流程当中，三角剖分起到了一个非常重要的地位。它为最终线性集合算法生成可见多边形提供了重要的边的拓扑排序信

息。为了实现线上时间复杂度为 O(n)的平面上可见多边形的生成算法，三角剖分部分的算法被分成了预处理阶段（Pre-process Phase）和执行阶段（Process Phase）。其中预处理阶段使用时间复杂度为 O(nlogn)的受约束 Delaunay 三角剖分生成算法，而处理阶段使用时间复杂度为 O(n)的三角网格更新算法。

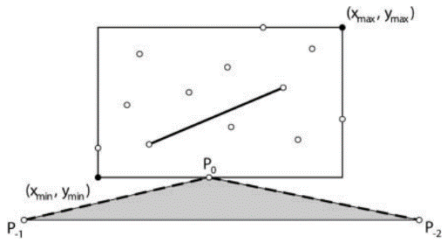
预处理阶段：

预处理阶段的受约束 Delaunay 算法需要完成的事情就是在 O(nlogn)的时间复杂度之内，对平面上的多边形进行三角剖分，要求最终生成的三角剖分包含所有多边形上的每一条边，并且最终生成的三角剖分形成一个原有多边形的凸包（之所以需要生成凸包是为了方便后面执行阶段的时候向上方射出一条射线更新三角剖分时的操作）。

在程序的实现当中，这一块使用了开源的多边形受约束 Delaunay 三角剖分库 poly2tri 来完成。poly2tri 的算法思路来源于 V. Domiter 和 B. Zalik 在 2008 年 3 月 19 日在 IJGIS 上发表的论文[4]。这是一种基于扫描线的三角剖分算法，具体的算法如下所示：

初始化：

将所有点按 Y 坐标升序排列，如果 Y 坐标相同，则按 X 坐标升序排列。插入两个人工点 $P_{-1}$，$P_{-2}$，使得他们位于所有点的下方，并且对于 X 坐标来说，$P_{-1}$ 位于所有点的左侧，$P_{-2}$ 位于所有点的右侧。这样我们就得到了构造的第一个三角形，$P_0P_{-1}P_{-2}$。该三角形包含两条前沿边（Advancing Front），$P_{-1}P_0$ 和 $P_0P_{-2}$。在最后一步最终确定的时候，所有与人工点相连的三角形将会被删去。这一步结束后，示意图如下所示：



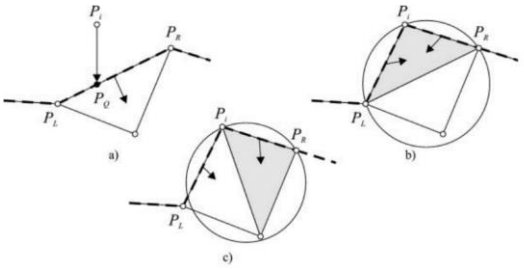

扫描：

对各个点按 Y 轴正方向进行扫描，扫描时分为这两种情况：

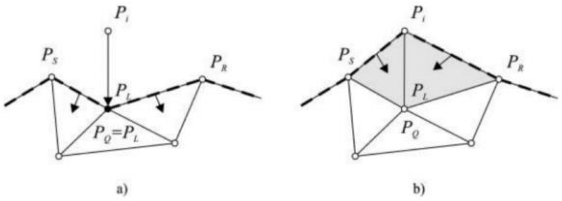
1. 某个顶点是孤立的点，或者是一个下边缘点时，顶点事件（Point Event）被触发。

   设该点为 $P_i$，该点到前沿边线段上的竖直方向投影记为 $P_Q$，前沿边线段的左右断点记录为 $P_L$,$P_R$，$P_L$ 在前沿边上的左边的临近线段断点 $P_S$。此时包含两种情况：

   1.1 点 $P_Q$ 在 PLPR 线段上，并不与端点重合。此时新的三角形 $P_RP_iP_L$ 将会被构造出来，新的前沿边将会被更新为 $P_LP_i$ 和 $P_iP_R$。如下图所示：

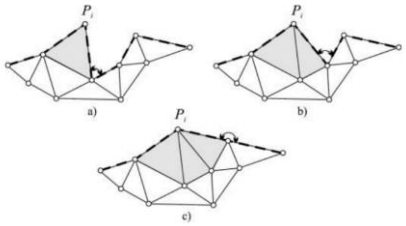
   
   

   1.2 点 PQ 与 PL 端点发生重合。此时将构成两个新三角形，$P_LP_iP_S$ 和 $P_LP_RP_i$。如图所示：

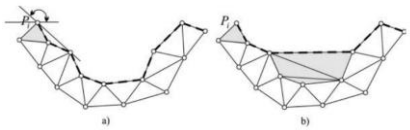
   
   

   将前沿边上其他的对于 $P_i$ 来说可见的点与 $P_i$ 相连以构成新的三角形。在构成新的三角形的时候，需要进行必要的角度检查：

   i) 新的三角形的边与前沿边线段之间的夹角必须小于 90 度，否则创建三角形行为将停止。如图所示：

ii) 前沿边上线段之间的夹角如果大于 135 度，说明存在盆状结构（Basins）。该盆状结构需要被更多新的三角形填充。如下图所示：



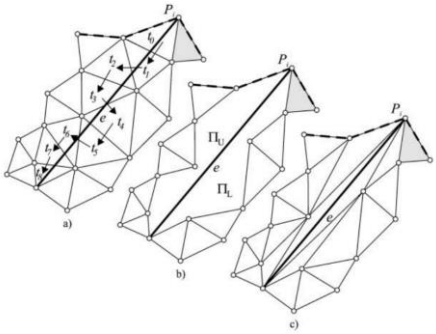
2. 某个顶点是一个上边缘点时，边事件（Edge Event）被触发。

当扫描遇到一个 Y 坐标较大的上边缘点时，一条受约束边 e（Constrained Edge）必须被插入到三角剖分的结果当中。在插入这条边的时候，将会遇到该条边与目前结果当中的三角形相交的问题。对于这种情况，该算法使用这样的三个步骤进行处理：

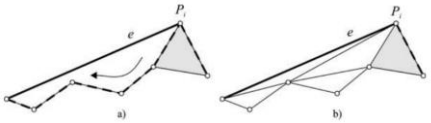
2.1 确定第一个与 e 相交的三角形，并按照特定的遍历方案，将该条边 e 所串联的所有三角形标记为需要删除

2.2 删除所有被 e 相交的串联三角形

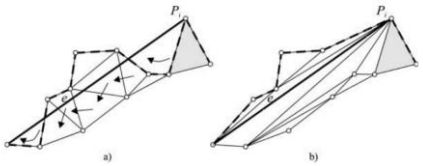
2.3 将被删除的空白区域重新进行三角剖分



这里将会面临三种情况。第一种情况就是 e 在前沿边线段的下方，这种情况照上述处理办法进行处理即可。第二种情况是 e 在前沿线段的上方。这是最简单的处理情况。由于没有任何相交的三角形，故不需要进行第一步遍历所有串联的三角形并标记删除的任务，只需要将 e 与上沿边构成的空白区域进行三角剖分即可。如下图所示：

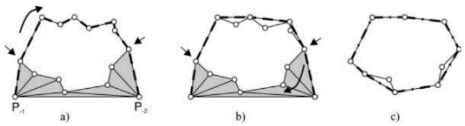


最复杂的是第三种情况，e 与前沿边有一个或多个交点时。此时需要将遍历方法进行一些转换。当前沿边线段上的点在 e 的上方时，遍历方案将依据被串联的相交三角形进行遍历；当前沿边线段上的点在 e 的下方时，遍历方案将依据前沿边线段上的顶点进行遍历。



当所有顶点和约束边都被扫描后，进入最终确定环节：

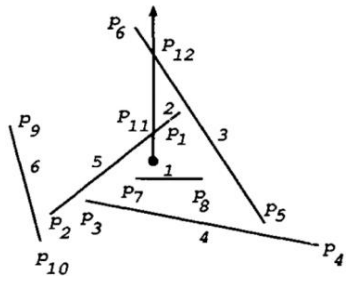
删除所有的与人工点相连的三角形，添加构成模型凸包的边框三角形到最终的结果当中。这样便完成了约束 Delaunay 三角化的扫描线算法。如下图所示：



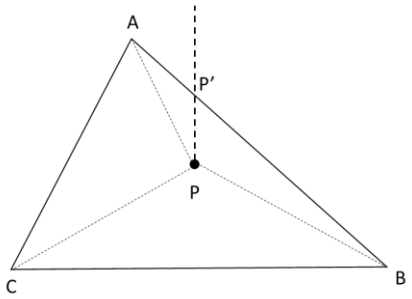
执行阶段：

预处理阶段结束以后，我们得到了一个多边形所在凸包的受约束 Delaunay 三角剖分。前面提到过，之所以使用多边形所在的凸包进行三角剖分的预处理，是为了执行阶段的时候射出一条射线对三角网格进行更新更加方便。在介绍执行阶段的三角网格更新之前，先对发射一条射线切分三角网格的目的加以解释。

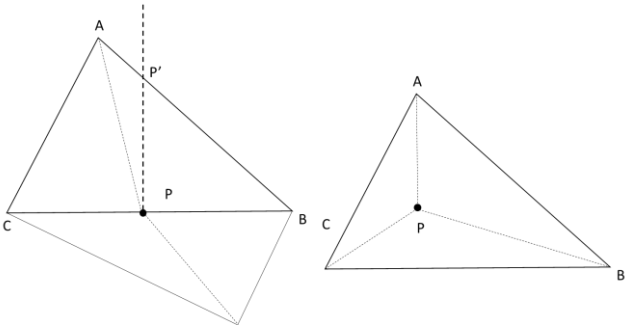
执行期间的三角剖分更新是为了给最后的线性集合算法提供多边形线段的拓扑顺序。而线性集合算法需要一个角度范围作为横坐标的输入，所以我们要把连续的 360 度空间给划分成一段从 0 度到 360 度的空间，为此，我们就需要射出一条直线作为 0 度和 360 度的划分线，并且把这条直线相交的所有线段分成两段。如下图所示：



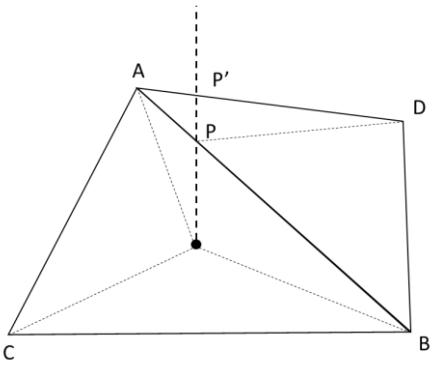

三角剖分更新的过程比较简单。首先查找到当前点 P 所在的三角形，设为 ABC，而 P 向上垂直发射一条射线，交三角形 ABC 的 AB 边上，交点设为 P’。这个时候，把三角形分为四个三角形，分别为 APC，PBC，AP’P 和 P’BP。如下图所示：



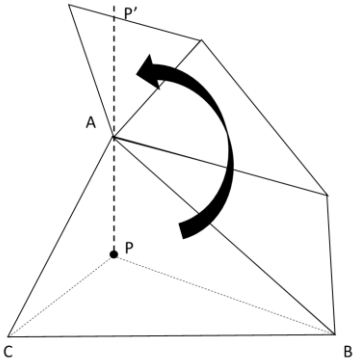

如果出现了 P’与 A 重合的情况，则三角形 AP’P 退化为一条边；如果出现了 P 落在边 BC 上的情况，则三角形 PBC 退化成两条边，并且切分另一个与三角形 ABC 共边 BC 的三角形。如下图所示：




接下来只要循环处理，将 P 替换成 P’，将三角形 ABC 替换成三角形 ABC 共边 AB 的三角形，然后继续按照上面的对三角形进行细分的方式对下一个三角形继续进行细分。如下图所示：

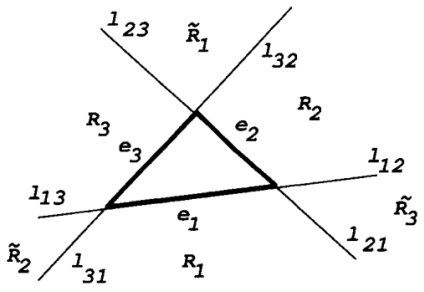



需要注意的是，当这条射线射到某个顶点上时，需要跳过很多个三角形知道射线射到最上的三角形的边界上，这种退化的情况如下图所示：




该循环将一直进行到射线射到一条不与任何三角形公共的边上后终止。所以对于多边形可能存在凹陷（Reflex）的情况，所以初始三角剖分要对该多边形的凸包进行，从而保证射线射到多边形的边界的时候，循环就可以终止。

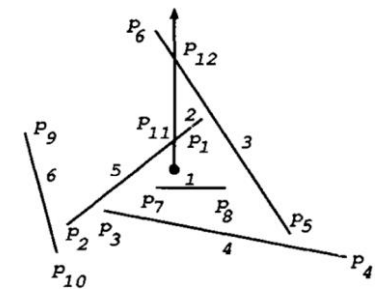
执行阶段更新完了所有的三角形以后，将每一个三角形的每条边作为节点，建立一个图的偏序关系。建立偏序关系的方法如下图所示，首先每个三角网格当中的三角形都可以将空间划分成几个部分，如图中的 $R_1$，$R_2$，$R_3$，$\tilde{R}_1$，$\tilde{R}_2$，$\tilde{R}_3$，$l_{13}$，$l_{31}$，$l_{12}$，$l_{21}$，$l_{23}$，$l_{32}$ 等

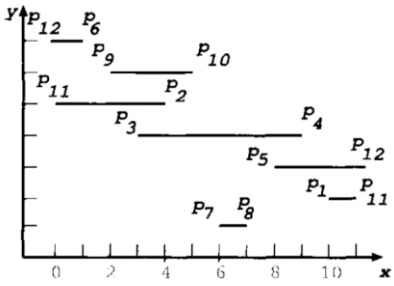


如果此时待求可见多边形的点位于 $R_1$ 的内部，那么就认为 $e_2>e_1$，$e_3>e_1$，$R_2$，$R_3$ 的情况类似；如果此时待求可见多边形的点位于$\tilde{R}_1$的内部，那么就认为 $e_2<e_1$，$e_3<e_1$，$\tilde{R}_2$，$\tilde{R}_3$的情况类似；如果此时点位于 $l_{12}$ 上，那么就认为 $e_2>e_3$。根据这种偏序关系建立一个有向无环图，最后对该图进行拓扑排序，得到最后的结果就是多边形所有边的拓扑顺序，为下一步的线性集合算法提供 Y 坐标上的排列信息。

**线性集合并计算可视多边形**

计算可见多边形的算法利用了之前计算出的顶点极角序和边的偏序关系，其将多边形顶点的极角序转化为 x 坐标（如下左图中 $P_{11}$ 和 $P_{12}$ 两点均在竖直的射线上，其 x 坐标均为 0；之后顶点的极角顺序为 $P_6$、$P_9$、$P_3$、$P_2$ 依次向后，因此其 x 坐标分别为 1、2、3、4 等等），并将多边形中的边关于输入点的偏序关系转化为排序的序号值并进一步转化为 y 坐标（$P_7P_8$=1、$P_1P_{11}$=2、$P_5P_{12}$=3 以此类推），从而将原先如下左图中的求可见多边形问题转化为了右图中对于每个 x 坐标求其对应的 y 值最小的线段。这种转化保证了原先图形中线段的遮挡关系，即每个区间对应的最低的线段一定是原图中输入点可见的线段。




进行转化后可以使用 Gabow[3]中的线性集合并算法计算，在线性时间内得出点 q 的可见多边形。算法的伪代码如下所示：

```
for k = 1 to |S_q|
    x = find(l_k)
    while (x < r_k)
        vis[x] = k
        link(x)
        x = find(x)
```

其本质为自底向上遍历每一条线段，为每一个 x 值标记其所对应的最低的线段的 y 值，并记录在 vis 数组中，最后将这些线段依次连在一起便得到了可见多边形。算法中涉及两个函数，find 函数返回包含 x 的集合中的最大元素，link 函数将包含 x 与包含 x+1 的集合合并起来。对于上面的例子，算法如下执行操作：首先遍历到 y 值为 1 和 2 的线段，即 $P_7P_8$ 和 $P_1P_{11}$，根据后边的 link 操作相应的 x 值 6、7 和 10、11 会分别组成集合，同时 vis[6]=1，vis[10]=2；下一个循环中遍历到 y 值为 3 的线段 $P_5P_{12}$，此时 8 和 9 会先组成集合，同时 vis[8]和 vis[9]会记为 3，之后在 while 循环中 x=9 时 link(9)操作会将集合{8, 9}与集合{10, 11}合并起来，再执行 find 时返回 11 会达到线段的右边界，循环也就终止了，这样 vis[10]便保留了之前的值，即已组成集合（vis 已被标记）的 x 值不会再次被访问，也就保证最后求出正确的线段 y 值。

上边的算法可以求出一个点关于多边形的整个可见区域，而我们的程序中需要求出某一视角内的可见多边形。因此我们根据视角的左右边界计算出其所在的 x 区间，通过输入点和两个边界求出其与所在 x 区间对应的最小 y 值线段的两个交点，之后按 vis 数组中的值依次连接相应线段便可得到一定视角范围内的可见多边形，这个过程同样可在线性时间内完成。

## 3. 总结

平面上可见多边形的生成算法当中包含了受约束 Delaunay 三角剖分，二维凸包，对偶图，DCEL 数据结构，线性集合算法，拓扑排序等等经典算法。在完成 PlanarSight 这个游戏的过程当中，着实地考验了一把我们的编程实际开发能力。通过这个课程实验，我们不仅对计算几何的相关算法认识地更加深入，同时结合理论的实践开发能力也得到了有效的提高。在完成这个课题实验之后，我们将会继续就平面上可见多边形这一课题进行进一步的拓展研究，争取获得更大的成果。

# 参考文献

[1] Asano T, Asano T, Guibas L, et al. Visibility of disjoint polygons[J]. Algorithmica, 1986, 1(1-4): 49-63.

[2] Chazelle B, Guibas L J, Lee D T. The power of geometric duality[J]. BIT Numerical Mathematics, 1985, 25(1): 76-90.

[3] Gabow H N, Tarjan R E. A linear-time algorithm for a special case of disjoint set union[C]//Proceedings of the fifteenth annual ACM symposium on Theory of computing. ACM, 1983: 246-251.

[4] Domiter V, Žalik B. Sweep-line algorithm for constrained Delaunay triangulation[J]. International Journal of Geographical Information Science, 2008, 22(4): 449-462.