平面上的三角划分

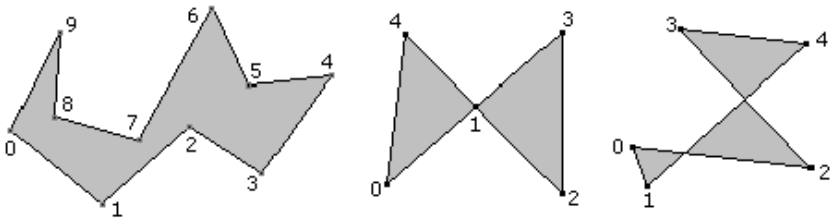
摘要

1. 简单多边形的三角划分。
2. 平面点集上的三角划分。
3. 一些实现细节。
4. 平面点集上的三角划分在寻路中的应用。
5. 参考资料。

简单多边形的三角划分

将简单多边形转换成一组由同样顶点组成的三角形集合是计算机图形学中的一个经典问题。问题中，简单多边形是指由一组有序顶点组成的，点 V0~点 Vn-1。相邻的顶点之间通过边(Vi, Vi-1)连接，并且边（Vn-1,V0）连接起始点。每个顶点被两条边所共享，而边的所有交点都是顶点。如下图：

Figure 1. The left polygon is simple. The middle polygon is not simple since vertex 1 is shared by more than two edges. The right polygon is not simple since the edge connecting vertices 1 and 4 is intersected by other edges at points that are not vertices.



图中，左边的多边形是个简单多边形，中间的多边形点 1 被四条边共享，不符合定义的条件，不算是简单多边形，右侧的多边形中边 14，边 02 的交点不是我们定义的顶点之一，因此该图形也不符合简单多边形的定义。

耳切法（EAR CLIPPING）处理简单多边形的三角划分

简单多边形的耳朵

简单多边形的耳朵是指由连续顶点 V0,V1 和 V2 组成的内部不包含其他任意顶点的三角形。在计算机几何术语中，v0 与 V2 之间的连线 称之为多边形的对角线，点 V1 称之为耳尖。

耳切法算法描述：

1. 将多边形使用双向链表存储，这样可以快速的移除耳朵。列表的构建复杂度是 O(n)。
2. 遍历顶点寻找耳朵。对于每一个顶点 Vi 和围绕该顶点的三角形<Vi-1,Vi，Vi+1>，测试其他顶点否在当前三角形中，如果有一个顶点在三角形里面，则不是耳朵，只有都不在的情况，才算是找到一个耳朵。

   具体实现的时候我们可以考虑以下因素让这个算法更为高效。当发现有一个点在三角形里面的时候便可以开始放弃当前测试。一个凹拐角其两边的夹角大于 180（即该顶点的内角为优角），而一个凸拐角两边夹角小于 180（即该顶点的内角为劣角）。

   一旦凸顶点和耳朵的链表构建成功，每次遍历都会移除一个耳朵。

   假设当前 Vi 是个耳朵并且被移除掉，那么边结构的相邻点 Vi-1,Vi+1 则会发生变化：如果相邻点是凸顶点，那么依旧保持凸点；如果相邻点是个耳朵，那么当 Vi 被移除后则不一定能保持耳朵的状态；如果相邻点是个凹点，那么他则有可能变为一个凸点甚至是耳朵。

   因此当移除顶点 Vi 后，如果相邻点是凸点，则必须遍历相关顶点，通过遍历查看是否包含其他点，来测试它是否是一个耳朵。
3. 当所有的耳朵被移除，算法结束，n 个顶点的简单多边形可以分为 n-2 个三角形。

   我们有 O(n)个耳朵，每一次更新都会触发一个耳朵检测，每次过程中更新 O(n)，所以 EarClipping 的复杂度是 O(n^2)。
4. 图示：

Figure 2. The right polygon shows the ear $\langle 2,3,4\rangle$ removed from the left polygon.

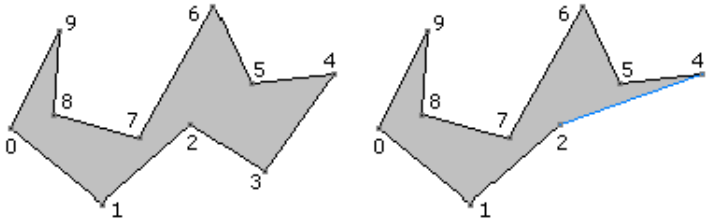
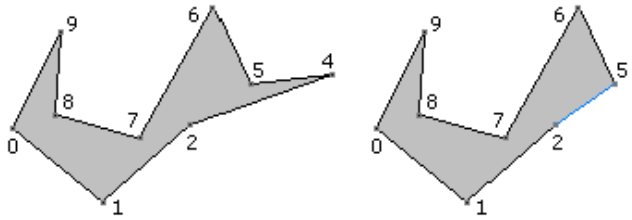
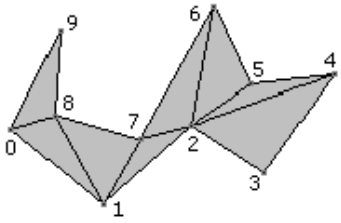


Figure 3. The right polygon shows the ear $\langle 2,3,4\rangle$ removed from the left polygon.



……

最终：



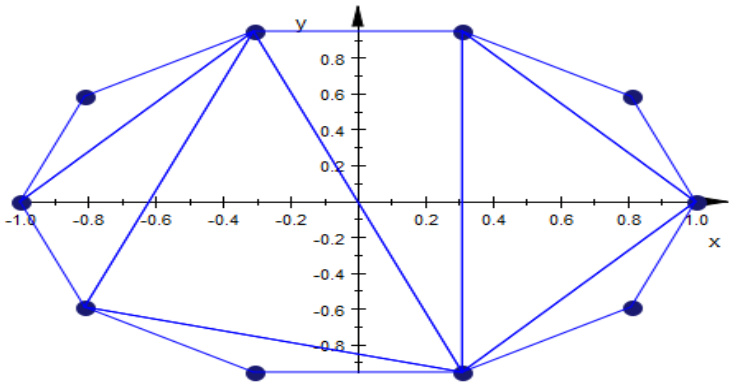
平面点集上的三角划分

假设 V 是二维实数域上的有限点集,边 e 是由点集中的点作为端点构成的封闭线段，E 为 e 的集合。

那么该点集 V 的一个三角划分 T=(V,E)是一个平面图 G，该平面图满足条件：

1. 除了端点，平面图中的边不包含点集中的任何点。
2. 没有相交边。
3. 平面图中所有的面都是三角面。

如图：

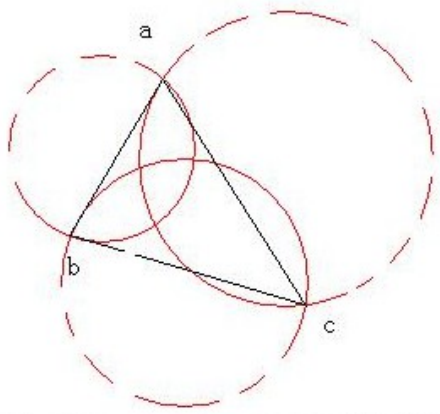


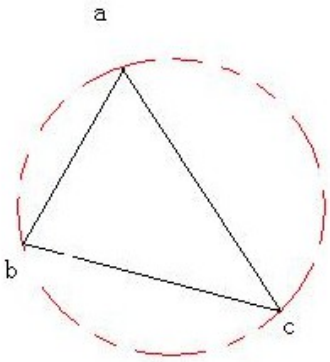
德劳内三角划分（DELAUNAY TRIANGULATION，DT）

假设 E 中的一条边 e（两个端点为 a,b），e 若满足下列条件，则称之为 Delaunay 边：存在一个圆经过 a,b 两点，圆内不含点集 V 中任何的点（不包括在圆上），这一特性又称空圆特性。如果点集 V 的一个三角划分 T 只包含 Delaunay 边，那么该三角划分称为 Delaunay 三角划分。

3 个不共线点构成的 DT

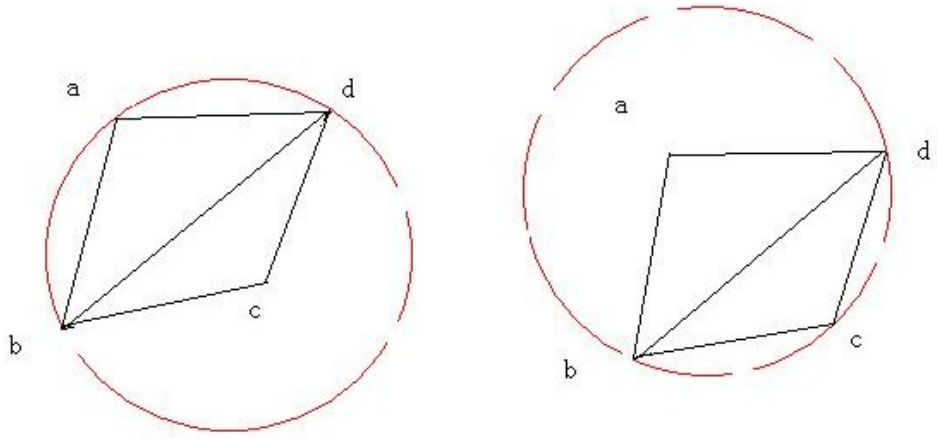
由三个不共线点构成的 DT 很简单，即这三个点构成的三角形。对每条边，都能找到一个经过边的端点，但不包含其它点的圆。一个特殊的情况，如该三角形的外接圆。



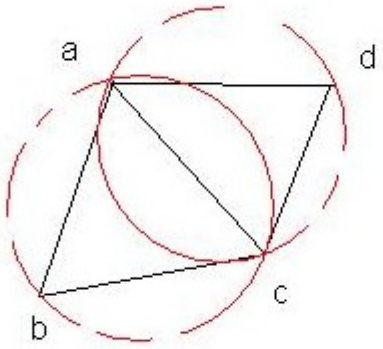


4 个点构成的 DT

由四个点构成的 DT，下图的划分不是 DT，因为过边 bd 的圆，要么比包含 a 或者 c。



这种情况可以进行一次翻转：



就得到了一个 DT。实际上，如果由边 bd 划分形成的三角划分不是 DT，那么经过一次翻转，由 ac 划分形成的三角划分必定是 DT。

这点基于 DT 的另一个重要性质：如上图，如果∠dab+∠bcd 大于 180 度，那么，边 bd 构成的划分不满足 DT 的性质。经过翻转，由于四边形的内角和为 360 度，∠abc+∠adc 必小于 180 度，因此，由边 ac 构成的划分满足 DT 的性质。

DT 算法描述

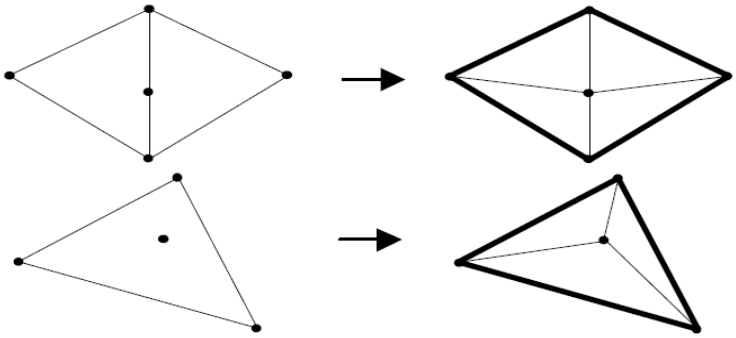
构造的 DT 的算法包含，扫描线法（Sweepline），随机增量法（Incremental），分治法（Divide and Conquer）。这里介绍一种最容易实现的算法，随机增量法：

1. 构造一个 super triangle，包含所有的点。

2. 按照随机的顺序依次插入点集中的点，在整个过程中都要维护并更新一个与当前点集对应的DT。

考虑插入 vi 点的情况，由于前面插入所有的点 v1,v2,...,vi-1 构成的 DT(v1,v2,...,vi-1)已经是 Delaunay 三角剖分，只需要考虑插入 vi 点后引起的变化，并作调整使得 DT(v1,v2,...,vi-1) ∪ vi 成为新的 Delaunay 三角剖分 DT(v1,v2,...,vi)。

插入调整过程如图：



首先确定 vi 落在哪个三角形中（或边上），然后将 vi 与三角形三个顶点连接起来构成三个三角形（或与共边的两个三角形的对顶点连接起来构成四个三角形）。

由于新生成的边以及原来的边可能不是或不再是 Delaunay 边，故进行边翻转来调整使之都成为 Delaunay 边，从而得出 DT(v1,v2,...,vi)。

3. 最后，移除 super triangle。

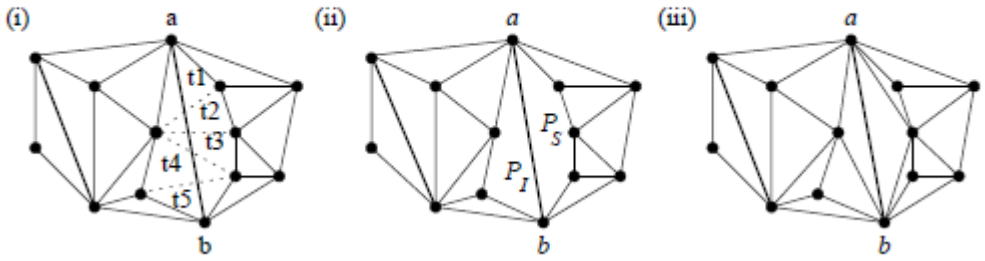
约束型德劳内三角划分（CONSTRAINTED DELAUNAY TRIANGULATION，CDT）

CDT 是 DT 的扩展，之中强制规定了一些必须存在的边，且 CDT 中的边，不全是 Delaunay 边。

插入约束边

1. 将约束边的端点看做常规的点，加入到 DT 中。
2. 移除所有被约束边切割的三角形。添加该约束边。

3. 对约束边两侧的区域进行三角划分。



注意，这里不能直接使用 EarClipping 来划分，因为两侧的区域不一定是多边形。

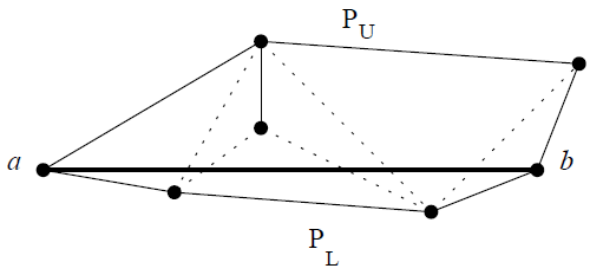


Figure 7: Example in which the upper pseudo-polygon contains repeated vertices

而是进行以下步骤（设约束边为 ab）：

1. 任选 ab 的一侧作为例子，令 L 为这一侧的所有点的集合。
2. 在 L 中找到这样一点 c，该点满足三角形 abc 的外接圆不包含 L 中的其他点。
3. 构造三角形 abc，ac 边和 bc 边将 L 划分为新的两个区域。
4. 递归执行该方法，直到区域点集为空。

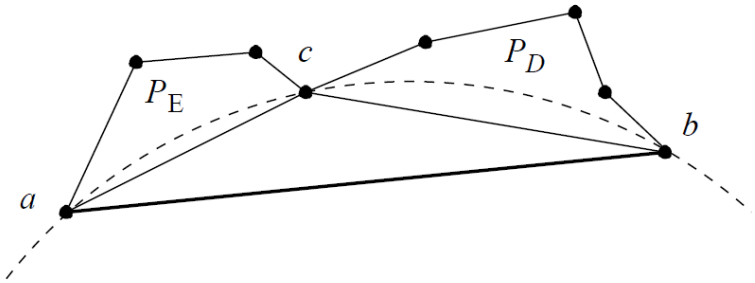


Figure 8: Representation of the recursive algorithm to triangulate a pseudo-polygon.

移除约束边

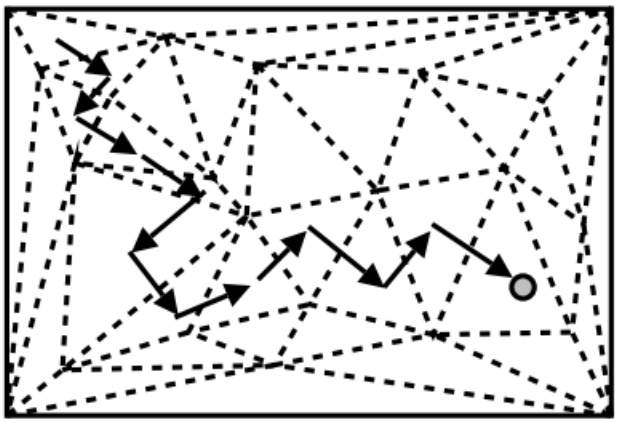
1. 收集所有使用该边的端点的边及三角形，得到一个多边形的区域。
2. 将该多边形顶点按极角排序（使逆时针遍历这些点，能形成一个简单多边形）。
3. 对该多边形执行 EarClipping，得到新的三角划分。

一些实现细节

查找点 V 落在哪个三角形内

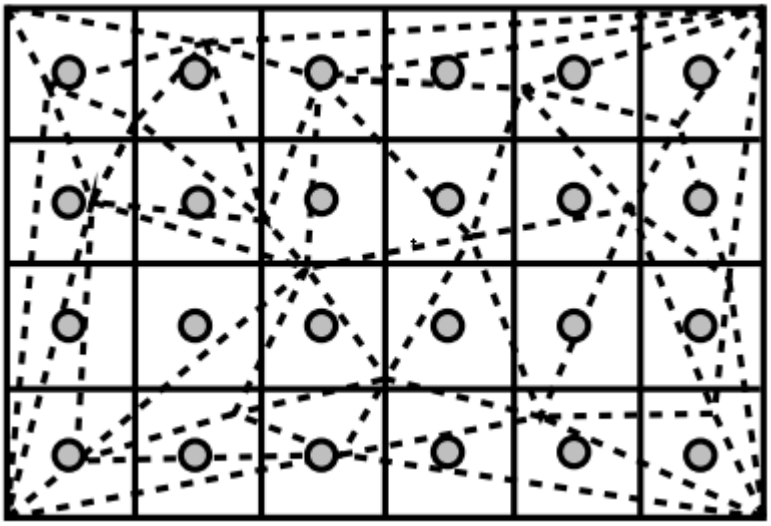
常规的方法，从任意一个三角形开始，判断点 v 与边 AB，BC，AC 的方向。如果 v 在边的逆时针方向，则继续检查下一条边。如果 v 在边的顺时针方向，那么继续检查该边另一侧的三角形。

直到找到包含 v 的三角形。最坏情况下，需要检查所有的三角形。



作为优化，可将场景分成若干个格子。当创建一个三角形时，检查该三角形的外接矩形所包含的格子。如果某个格子的中心在三角形内，那么标记该格子的一个指针成员变量指向该三角形。

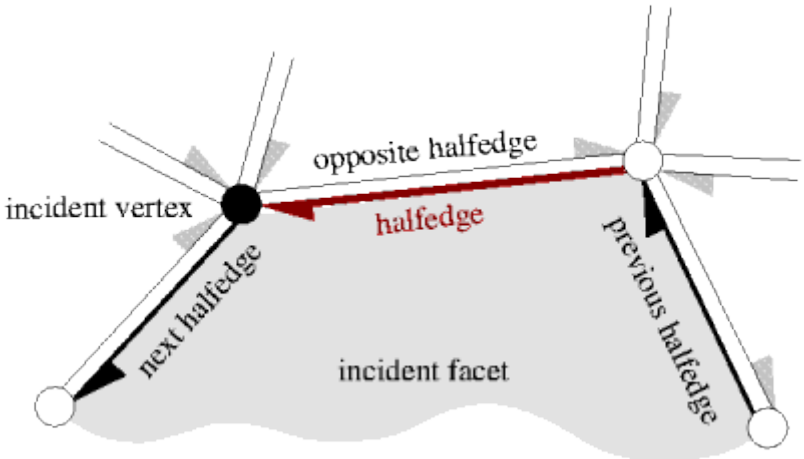
在查找点 v 所在的三角形时，就从该点所在的格子指针变量所指的三角形开始。如果该三角形为空，那么再从任意三角形开始。



半边数据结构（HALFEDGE）

在构造 DT 或者 CDT 时，经常需要访问一条边两侧的三角面，来判断是否满足 DT 的性质，也经常访问一个面的所有邻接三角面等。

为了高效的解决这种问题，可以使用 QuadEdge 等数据结构。这里介绍一种更加简单使用的数据结构：“半边”数据结构。



定义

下面是边的定义：

```
struct HE_edge {
    HE_vert* vert;    // 边的末端顶点。
    HE_edge* pair;    // 另外“半”条边，方向与此相反。
    HE_face* face;    // 该边所属于的面。
    HE_edge* next;    // 该边的下一条边。
};
```

该边只存储实际边的一半。而且该边是有方向的，通过终点即可以知道。

下面是点的定义：

```
struct HE_vert {
    float x, y, z;
    HE_edge* edge;  // 任意一条以此点为起点的边。
};
```

下面是面的定义：

```
struct HE_face {
    HE_edge* edge;  // 包围该面的任意一条边。
};
```

用法

1. 获取边的两点：

```
HE_vert* vert1 = edge->vert;
HE_vert* vert2 = edge->pair->vert;
```

2. 获取边两侧的面：

```
HE_face* face1 = edge->face;
HE_face* face2 = edge->pair->face;
```

3. 遍历一个面的所有边：

```
HE_edge* edge = face->edge;
do {
    // do something with edge
    edge = edge->next;
} while (edge != face->edge);
```

三角划分在寻路中的应用

可以将 CDT 应用与 2D 场景内的寻路上，用约束边勾画出障碍物的轮廓，然后生成 CDT。较用格子划分而言，表达障碍物的形状更精确。并且也可以通过修改 CDT，在运行时改变障碍物的位置（效率上可能不高）。

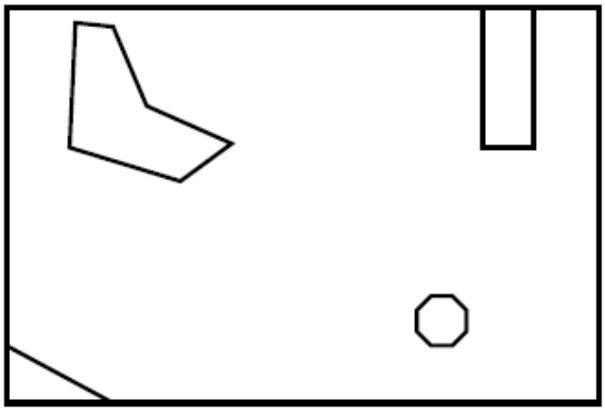


Figure 3.7: Same environment with curves approximated by line segments

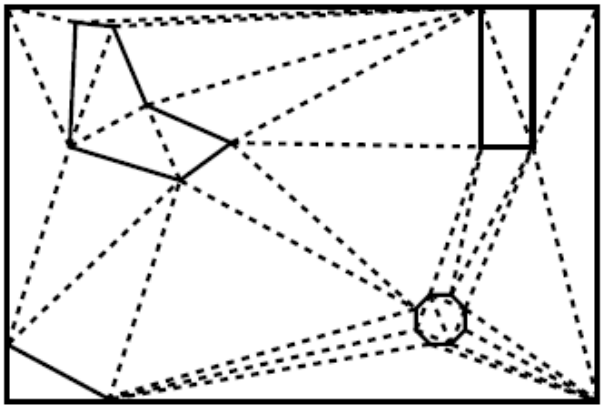


Figure 3.8: Environment represented by a constrained triangulation

参考资料

1. https://en.wikipedia.org/wiki/Delaunay_triangulation
2. https://skatgame.net/mburo/ps/thesis_demyen_2006.pdf
3. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.9234&rep=rep1&type=pdf
4. http://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf
5. http://www.flipcode.com/archives/The_Half-Edge_Data_Structure.shtml
6. http://www.docin.com/p-1034849935.html
7. http://www.cnblogs.com/soroman/archive/2007/05/17/750430.html