学号：__2019302110028__

密级：_____

武汉大学本科毕业论文

任意平面多边形切分算法设计与实现

院（系）名 称 ： 计算机学院

专 业 名 称 ： 软件工程

学 生 姓 名 ： 逄博

指 导 教 师 ： 傅佑铭 讲师

二〇二三年五月

郑 重 声 明

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名: _____ 日期: _____

摘　　要

平面多边形的三角剖分是计算几何的基础问题之一。计算机在渲染一个几何形状时，常常需要把原图形分解为多个三角形的并，分别对每个三角形依次渲染。通过减少三角剖分中生成三角形的数目，避免细长三角形的产生，可以优化三角剖分所产生的方案，并改善后续渲染的结果。

本文设计并实现了两种基于平面多边形构建其三角剖分的算法，可在不添加新顶点的情况下快速将任意形状带孔洞多边形划分成数个互不重合的三角形，能正确处理多种复杂情况，且产生的划分中细长三角形较少，能够减小后续操作的误差，便于后续处理。第一种算法从任意平面多边形开始，依次消除了其中的相交边界，内部孔洞，凹点，最终将其转化为数个凸多边形的并集，并分别求出其最优三角剖分。第二种算法则是记录多边形顶点集合，并计算出全部顶点的三角剖分结果，然后在维持三角剖分性质不被破坏的情况下将原多边形的所有边插入剖分结果之中，并删除多边形外部的额外连边，以产生三角剖分结果。与当下常用的几种算法相比，本文提到的算法具有更广阔的适用范围，可以正确处理平面中带复杂形状孔洞的任意多边形，而许多其他算法可能会错误的生成部分覆盖多边形孔洞的划分。本文中算法一具有高度模块化的特点，逐步分割的每一步都可以单独取出进行进一步的调优，可以较为透明的查看分割中途的状态，便于后续改进和演示。本文中第二种算法则是基于多边形点集的最优解基础上进行调整得来的结果，在调整后整体性质优秀，执行效率高。

经过实验，本文的算法可以在数秒内完成对包含十万个顶点的复杂平面多边形的三角划分，且产生的划分结果优秀，三角形形状规整。

关键词：计算几何；平面多边形；三角化

目　　录

1 引言

1.1 概述

计算机在对一个平面多边形进行栅格化时，一般需要将其切分为多个三角形的并，之后对每个小三角形依次渲染，最终合并出原多边形。在切分出的三角形中，具有两个极小锐角的细长三角形被称为“sliver triangles”[1]，具有一些不利于后续计算的特性。细长三角形由于具有极小的锐角，从三角形的末端开始存在较长一段区域宽度极小，渲染时不足一个像素，因此会由于精度原因丢失内容。另外，细长三角形由于长度过长，会导致多边形内较多的出现距离较近的点属于不同三角形，而距离较远的两个点却属于同一三角形的问题。因此，在设计多边形栅格化的算法时，常常需要在做到高效的同时避免细长三角形的大量产生。

本文实现了两种近似算法，可在不添加新顶点的情况下快速将任意形状带孔凹多边形划分成数个互不重合的三角形，能正确处理多种复杂情况，且产生的细长三角形数目较少，便于后续处理。

1.2 研究现状

平面多边形三角化作为计算机图形学的基础课题，当下研究较为成熟，有着较多的相关研究。近些年的论文以对已有的三角划分算法应用为主，在 GIS[2]，医学图像[3]，计算机图形学等方面有了较多的成果。而对三角划分算法本身的研究则多发表于二十一世纪初甚至更早之前。

下面简要介绍几种当前使用较广泛的三角化算法及其概念。

三角剖分（triangulation）：假设在平面有一点集 V，三角形 s 是由点集中的点作为端点的非退化三角形，S 为 s 的集合。则该点集 V 的一个三角剖分 T=（V，S）是一个平面图，满足以下条件：

- 平面图中的任意两个三角形交集面积为 0
- 平面图中所有三角形的并集是点集 V 的凸包

Ear Clipping（耳切法）[4] 是一种较为简易的多边形三角化算法，其思路简单易懂，但缺点是可能会产生较多的细长三角形。原理为不断在多边形内寻找可被

切除①的连续三个顶点构成的三角形。对于任意多边形，至少存在两个可被切除的三角形。轮流切除所有的三角形后算法结束。朴素情况下该算法的复杂度为 $O(n^2)$。

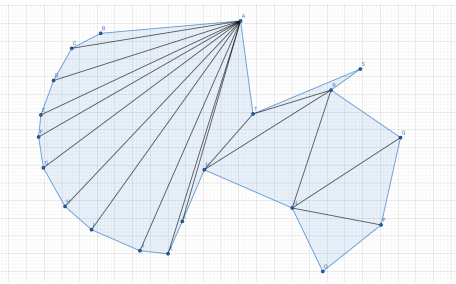

图 1.1　耳切法的典型结果

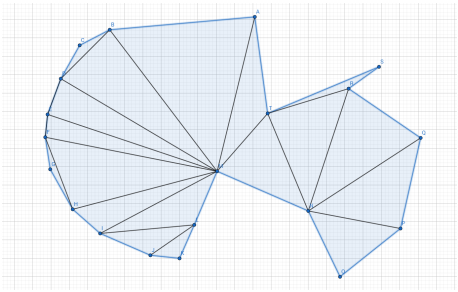

图 1.2　相对更好的一种解

Delaunay triangulation[5] 算法则会生成一组相对更加优秀的解，其满足形成的三角形中最小角尽可能大。但该算法只适用于凸多边形的划分。虽然计算得出的解在处理后也可用于凹多边形划分，但该划分结果可能会产生新的顶点，导致最终的三角形个数增加。Delaunay 三角划分有数种实现方式，其中较为常用的算法如 Bowyer-Watson 算法，其原理是维护一个合法的 Delaunay 三角划分，每次向点集中加入新点，并修改新增点附近的三角划分规则。其复杂度为 $O(n^2)$。或者也可以采用分治法，将点集划分为两部分并合并生成的 Delaunay 三角划分，其时间复杂度为 $O(n * \log_2 n)$[6]。

可以看出，上面的两种三角化算法各有优劣。

耳切法逻辑简单易懂，能够处理任意形状的多边形，对于多边形形态这一方面具有最大的兼容性。但其在产生的多边形形态上缺少控制，贪心的选择能够从当前多边形直接切除的一个三角形，未考虑切除多边形的形态是否较优。

Delaunay 三角划分的实现相对来说更加复杂，精度最高，但是只能对凸多边形产生正确的结果。对凹多边形或带孔多边形进行处理时，无法考虑到多边形内已有的边，会产生与之冲突的剖分结果，导致需要在冲突处新增顶点，将之前的切分线分成两段，才能处理一般情况下的多边形。

1.3　概念定义

定义 N_i 为二维平面上的点，N_{ix}, N_{iy} 分别为该点的横纵坐标；

①当一个三角形内部不包含其他顶点且该三角形属于原多边形一部分时认为其可切除

定义有序集合 $S:N_0,N_1,\cdots,N_{n-1}$ 表示有 n 个节点的无孔洞简单多边形，对任意 $i\in[0,n-1]$ 满足 $N_i,N_{(i+1)\bmod n}$ 之间存在一条连边。

定义有序集合 $T:S_0,S_1,\cdots,S_{n-1}$ 表示有 n 段边界的有孔洞简单多边形，其中 S_0 表示多边形的外边界，其余则表示多边形中的孔洞。

1.4 代码结构

本代码采用 C++ 实现，在注重运行效率的同时兼顾了一定的可读性。

对于第一种算法，我们使用双向链表存储节点及其连边，每个多边形内记录一个指向链表节点的指针，整个图对象内使用数组记录指向每一个多边形的指针；

为了便于数据处理和计算，每个链表节点同时存储了当前节点的标号，部分代码处理时利用节点的标号辅助确定节点的相对位置；

图对象内同时存储了指向全部初始结点的指针，可根据需求进行排序以便于部分处理。

对于第二种算法，我们采用数组记录多边形的所有顶点，并采用链表按极角序记录每个顶点的所有连边信息。

IO 部分实现了以 svg 图片格式输出中间过程和最终结果，可以直观的展示图片分割结果。

代码编译产物为静态链接库，可在其他程序中通过函数调用进行计算。

1.5 预期效果

代码支持从标准输入或文件读取多边形，并将分割结果以图片或三角形集合的形式输出。多边形分割完成后由数个三角形构成，且各三角形之间互不相交，也不会产生覆盖区域在原多边形外部的三角形。在三角剖分的结果中细长三角形的数目较少，大部分三角形较为规则。

2　采用逐步分割的方式将多边形三角化

我们尝试逐步将待解决的问题简化，将多边形逐渐处理成更加简单的形式以最终解决问题。

2.1　将非简单多边形标准化

通常情况下，我们处理的多边形为边界互不相交的简单多边形。对于边界产生了交叉的复杂多边形，需要先进行分割处理，使其变为等价的数个简单多边形。

在判断复杂多边形所覆盖范围时，有着多种可行的定义方法。此处我们采用的定义为：对于一个点，从该点向多边形外无限远处连接任意一条射线，当且仅当该射线与多边形边界相交奇数次时点在多边形内部。

如图 2.1所示，该复杂多边形由六个顶点组成，所包含的区域用浅蓝色表示。

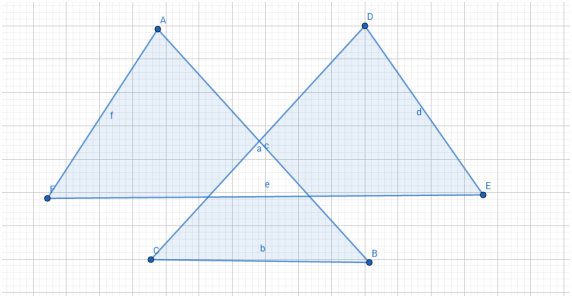

图 2.1　复杂多边形的示例

由于多边形的边界为一条闭合折线，易知对于每个多边形边界上的顶点，与其连接的边为偶数条。因此，射线的位置对相交次数的奇偶性不产生影响。

明确了划分规则后，我们可以采用逐层分离的方法将原多边形分割为简单多边形的组合。具体方法如下：

1. 首先，对于多边形的边界，找出所有未被记录的交叉点，并将对应相交的边分别从交叉点处分割为两部分。
2. 选择当前点集中 x 坐标最小的点，作为新多边形的起始点。
3. 遍历当前点的所有连边，按极角序选择当前点到上一个点连线方向逆时针连接到的第一条边①，将另一个端点加入当前多边形。

①对于第一个点直接选择极角在 $[-\pi,\pi]$ 区间内最大的边

4. 转到 3 直至返回初始点。
5. 记录当前多边形，并删除当前多边形对应的边和连边数小于 2 的顶点。
6. 转到 2 直至点集为空。
7. 对于切割出的每个多边形，计算其内部到多边形外跨越的边数，判断其是否为孔洞。若多边形为孔洞，将其加入上一个生成的多边形的孔洞列表内。

该算法的本质是记录所有未被顶点标记的交叉点，并重新计算原多边形的边界，判断多边形中每一条边具体所属的多边形边界曲线。

2.2 消除多边形内部的孔洞

通过在顶点上添加桥边和辅助节点，我们可以通过一次切割联通两个多边形孔洞，或是将一个多边形孔洞与外界相连。如图 2.2所示，在添加额外的两个顶点和两条边后，两个多边形孔洞的边界将会合并为一个连续的环，而其代表的多边形区域不发生改变。由此，通过数次合并后，原有的带孔多边形将变为不含孔洞的凹多边形。

2.2.1 预处理

为确保多边形孔洞合并后点的顺序正确，边界不产生交叉，需要提前对多边形孔洞的顶点方向进行预处理。

我们令多边形外圈的点以逆时针顺序排列，内圈孔洞则以顺时针顺序排列。如此处理之后，两个多边形孔洞在合并时将保持正确的旋转方向，避免边界出现交叉。

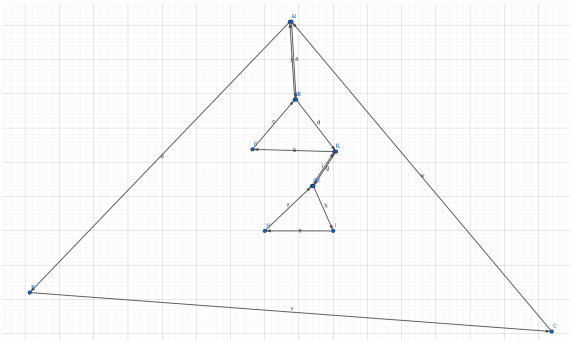

图 2.2　正确处理多边形旋转顺序后的合并

2.2.2 朴素的合并方式

将多边形上的所有点按照 x 坐标从小到大为第一关键字，y 坐标从大到小为第二关键字排序。按顺序遍历所有顶点，并记录当前已经遍历的所有节点所属的边界。当新遍历到的顶点所属的边界与之前的边界未进行过合并时，选择之前遍历过的顶点之中与当前点距离最近的顶点，向其延伸一条射线，并于第一段相连的多边形边界处相连。以此线段作为切割线，合并相关联的两个边界。

遍历多边形上的所有节点后，多边形的每个孔洞都直接或间接的与多边形外边界相连，因此我们便可以消除多边形内部的孔洞。

2.2.3 对上述算法的改良

借助最小生成树的思路，我们可以最小化切割边的总长度。

枚举多边形中的每对分属两个不同多边形孔洞的点对，并判断其连线是否与其他边冲突，记录所有合法的切割边。

将切割边按长度从小到大排序，按顺序处理边。对于每条切割边，若切割边所连接的两部分多边形未被合并，且该边与已有的切割边之间不产生冲突，则沿此切割边将两侧的多边形边界合并为一体。当选择的切割边数目恰好与多边形孔洞数目相等时，所有的多边形边界均合并完成，原多边形的孔洞消除完成。

2.2.4 采用分治法合并孔洞

我们可以采用分治计算 Delaunay triangulation 算法中的思想，使用相似的方法完成对孔洞的合并。

首先将多边形孔洞的所有点按照 x 坐标从小到大为第一关键字，y 坐标从大到小为第二关键字排序。接下来每次将待处理的点集分为左右两半，递归合并点集中的不同多边形片段。

当点集中的点不多于两个时，递归终止。若点集中的点分属不同多边形，则添加一条新的切割边连接两个顶点，将其加入候选集合。

接下来我们处理两个点集答案的合并。

从原多边形孔洞的边集中搜索，判断是否存在原有的连接两半点集的边。

若不存在这样的边，则使用分治计算 Delaunay triangulation 算法中的计算方式，找出所有合法的 LR-edge，在候选集合中修正与之冲突的切割边，并额外添加

一条最短的 LR-edge 至集合中。此处的合法 LR-edge 计算方式与分治计算 Delaunay triangulation 算法中的相似，但是寻找可行端点时需要排除连边时会与原多边形中已有的边产生冲突的顶点。

若存在连接两半点集的边，则将其直接加入集合，在候选集合中删除与之冲突的切割边，并寻找一条额外的切割边，连接原切割边所连接的集合。

由此思路，即可完成对所有孔洞的合并。接下来取孔洞中与边界距离最短的点对相连即可将多边形的孔洞消除。

2.3 分割不含孔洞的凹多边形

2.3.1 确定多边形方向

为了便于后续计算，我们规定多边形 S 的点依次逆时针排列。在计算时我们需要将初始的多边形端点标准化为按逆时针排序。利用向量叉积的原理，我们可以计算出多边形的方向。对于多边形的每条边，计算坐标原点到起点的向量，与边起点到终点的向量叉乘并除以 2。将结果求和后该向量的模即为多边形面积，而向量 z 轴方向则代表了多边形的旋转方向。

$$\sum_{i=0}^{n} \frac{N_i \times (N_{(i+1) \bmod n} - N_i)}{2} \tag{2.1}$$

计算出初始多边形旋转方向后，即可根据需求调整端点的顺序以标准化多边形的方向。

2.3.2 确认多边形凹凸性

首先我们遍历多边形的所有内角，检测其是否为凸多边形。

若多边形为凸多边形，转至 2.4，直接生成其 Delaunay 三角划分；若多边形为凹多边形，则采用下面的算法将其分割为数个凸多边形。

2.3.3 确定分割方案

在这一步我们采用启发式搜索的方式进行分割。

我们定义合法的切割边 (V_i, V_j) 满足以下条件：

- V_i, V_j 属于同一个多边形
- 以 V_i, V_j 为端点的线段除端点外与多边形无共同点

- V_i, V_j 两个顶点对应的多边形内角至少有一个大于 180°
- $0 \le i < j \le n-1$

对于原多边形 S 中的一条合法的切割边，我们采用如下方式来评估其价值：

- 设按此方式切割后产生的两个多边形为

  $S_1 : \{V_0, V_1, \cdots, V_{i-1}, V_i, V_j, V_{j+1}, \cdots, V_{n-1}\}$

  $S_2 : \{V_i, V_{i+1}, \cdots, V_{j-1}, V_j\}$

  定义 $C(S_i)$ 为多边形 S_i 的周长，　$L(V_i, V_j)$ 为点 V_i, V_j 之间的距离
- 定义切割边的基础评分为

$$\frac{\min(C(S_1), C(S_2))}{L(V_i, V_j)} \tag{2.2}$$

- 定义切割边的评分修正为

$$\frac{\min(\angle V_j V_i V_{i-1}, \angle V_j V_i V_{i+1}, \angle V_i V_j V_{j-1}, \angle V_i V_j V_{j+1})}{90^\circ} \tag{2.3}$$

  若切割边两个端点对应的内角均 >180° 则评分修正额外 *10
- 每条边的价值最终评估为基础评分 * 评分修正。

遍历多边形，对于每一个内角大于 180° 的顶点，采用此算法[7][8] 快速求出以该点为端点的所有合法切割边，计算其价值并加入记录。

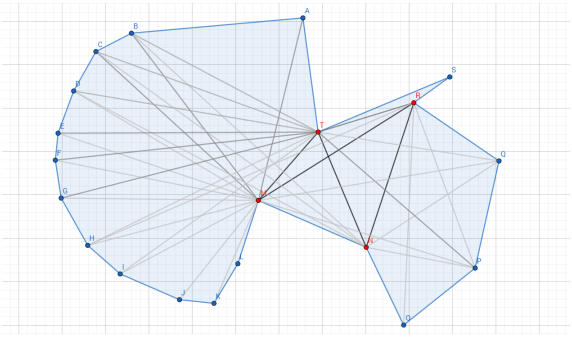

图 2.3　对候选边进行估价

接下来对于所有合法的切割边，将其以价值从大到小排序，依次尝试加入分割列表。

当我们尝试将一条合法切割边加入分割方案时，在已经选定的分割方案中检查是否存在与当前切割边冲突的方案。若存在冲突则直接跳过该边。具体来说，若两条切割边在非顶点的位置相交，则说明两种分割方案互相冲突，无法同时选用。此时我们选择价值相对较高者。

当我们计算得出了分割方案后，即可开始进行下一步的实际分割。

2.3.4 对凹多边形进行分割

确定了全部的可行分割方案后，我们可以开始划分凹多边形。

具体实现流程如下：

1. 对分割方案 V_i, V_j 排序，以 i 为第一关键字递增，以 j 为第二关键字递减。
2. 若栈顶元素 V_{stk} 满足 $stk < j$ 则弹出栈顶元素直至 $stk \geq j$。
3. 若栈顶元素 V_{stk} 满足 $stk = j$ 则将当前分割方案中的 V_j 指向的节点变更为 V_stk，并弹出栈顶元素。
4. 选择接下来的分割方案，将 V_i, V_j 节点复制一份为 V_i', V_j'。
5. 在链表中连接 V_{i-1}, V_i', V_j 与 V_{j-1}, V_j', V_i。
6. 将 V_j 加入栈中。

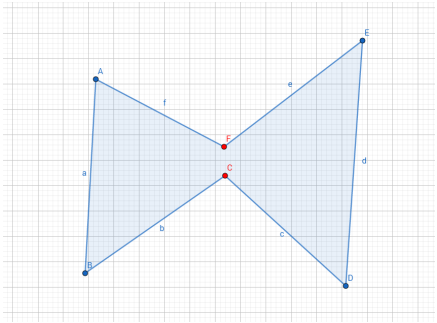

图 2.4 分割前的多边形

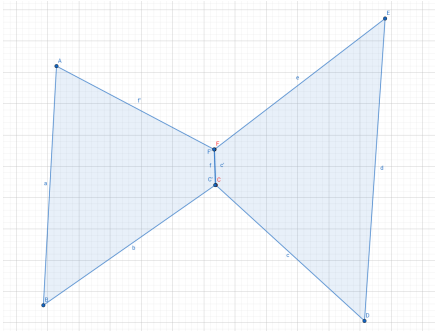

图 2.5 在待分割点的逆时针方向新增等位点

以这种方案实现对多边形的分割时，可以巧妙的准确找到每个分割边所对应的顶点的真实位置。即使顶点被复制多次，初始指向顶点的指针指向了错误的多边形，也可以利用栈找到每个指针的正确位置。

完成上述流程后，原本的一个凹多边形将被分为数个部分，我们只需递归的对每个子多边形使用相同方式进行处理即可。

2.3.5 一些其他的改进思路

本算法可以较为有效的将凹多边形分割为数个凸多边形，但生成分割边时将会尽可能多的对原多边形进行分割，可能会在单个凹点处进行过多的切割，导致出现较小的锐角。对应图 2.3的情况，尽可能多的选择分割边时将会对一些评分较

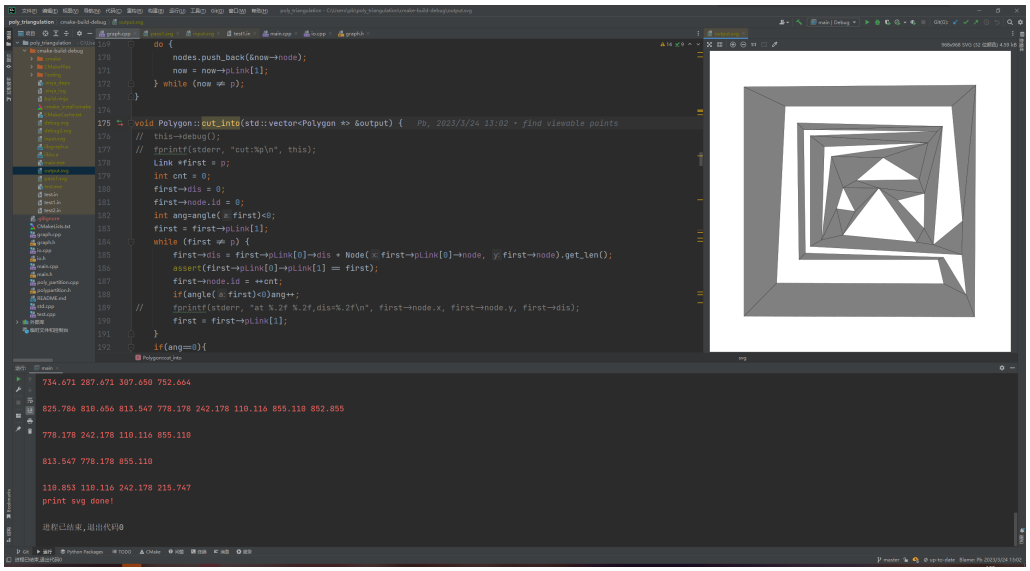

图 2.6　代码运行到此步的结果

低不需要进行分割，但并未和已选边产生冲突的垃圾边也进行切割，导致产生大量的细长三角形。以下是一些改进本算法的思路。

2.3.5.1　限制分割线数量

在确定可行分割方案的过程中，默认采用了贪心算法，只要存在可行的分割方法就会加入结果中。因此可能会出现从一个凹顶点连线到凸多边形每个顶点进行分割的情况；而理想情况下本章算法的目标为用相对较少的线段将输入的凹多边形分割为数个凸多边形。因此，我们可以对分割线的数量进行限制。

在较为理想的情况下，每一条分割线可以消除 1-2 个凹点。也就是说，将一个凹多边形分割为数个凸多边形时，使用的分割线数目很可能少于凹顶点的数目。因此，我们可以限制每次选择分割方案的数目上限。

我们可以在算法的开始阶段记录当前多边形的凹顶点数目，并将其乘上一个修正量后作为分割线的最大数目。在依次检验分割方案是否与已选中部分冲突时，若已选中的分割线数目已经等于最大数目，则丢弃剩余的所有方案，只使用有限数量的分割线进行分割。由于被分割出的小多边形会用递归的方式再次经过本算法检验凹凸性并进行分割，算法的正确性不会受到影响。

正确的选择修正量可以在精确度与速度之中做出平衡。较大的修正量会产生更多的分割线，生成的小多边形将更不可能是需要重新执行本算法的凹多边形；而较小的修正量则能够获得较为优秀的划分结果，但会增加算法的运行时间。笔者

认为，在实际计算中取 0.5 至 1 的修正量较为合适。

2.3.5.2 禁止单个顶点进行多次切分

在评估时，单个凹顶点可能有相似的数个较优秀候选切割边。这些切割边之间互不冲突，且评估价值相近。在筛选分割方案时，容易出现单个顶点选择数条同质切割边的情况。不仅会产生一些细长三角形，同时还影响算法效率。可以在切分时限制单个顶点最多被切分一次，跳过后续的相似切割边。

2.3.5.3 动态调整启发函数

在启发式函数中，采用的评估方案通过计算边长与切下的小多边形周长比值作为基础评分，以顶点凹凸性和最小角度数作为修正值。但在完成了一次切割后，后续的待选切割边两端的多边形周长将会发生变化，实际评估值将会改变，需要随时进行更新。考虑到实际评估值在分割过程中仅会下降不会上升，我们可以利用大根堆存储所有的切割方案，并在取新方案时对该方案进行更新并重新放入堆中，直到取出的方案已被更新过。

2.4 对凸多边形进行三角化

最终的凸多边形②是平面多边形中最简单最易处理的一类。

我们在实现了对凸多边形的三角化后即完成了整个算法。

2.4.1 朴素的划分方案

由于凸多边形的性质，多边形任意两个顶点之间连边均在多边形内部。因此最简单的划分方案即为对于 $i \in [2, n-2]$，沿边 S_0, S_i 进行一次分割，即可产生总共 n-2 个三角形。该方案最为简便，且产生的三角形均有一个共同顶点，但产生的三角形形状均为细长型，相对形态较劣。

2.4.2 改进的朴素划分

由于对凸多边形分割后产生的两个多边形仍为凸多边形，我们可以采用分治法来一定程度的优化上述方案。一种可行的方案如下：

②每个内角均小于 180 的多边形

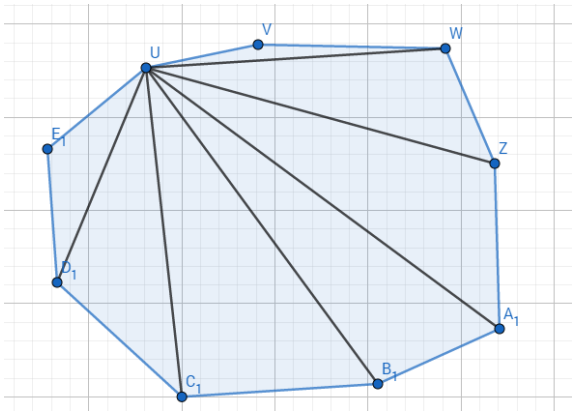

图 2.7　最简单的一种三角划分策略

- 若当前多边形为三角形则结束算法；
- 随机选择两个不相邻的顶点连边，将当前多边形分割为两部分；
- 对分割出的两个子多边形调用本算法进行分割，并将分割的结果加入答案。

沿着这个方向，该方案仍然有一定的优化空间。我们若倾向于每次尽可能的将多边形分割为面积相似，形状接近圆形的两个部分，分割的最终结果将会更加优秀。但是该方案代价相对较大，且最终产生的结果相比于随机法并没有显著的提升。

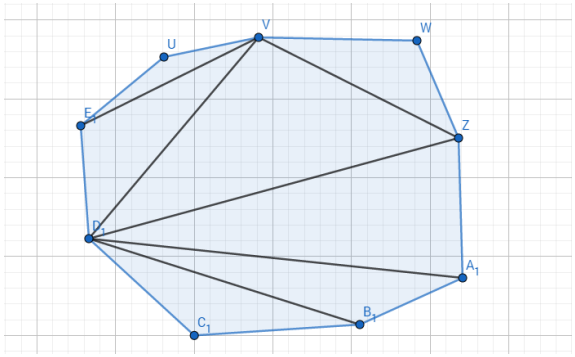

图 2.8　经过随机化优化后的分割结果

2.4.3　完美三角剖分

Delaunay triangulation（简称 DT）是一种三角剖分方式，其结果满足以下性质：

- 对于三角剖分中产生的每一个三角形，其外接圆中不包含点集 V 中的其他点。
- 若对点集的三角剖分结果中每一个三角形的最小角升序排列，则 DT 所得到的数值比其他三角剖分的数值大。
- 若点集中任意四点均不共圆，则产生的三角剖分唯一。

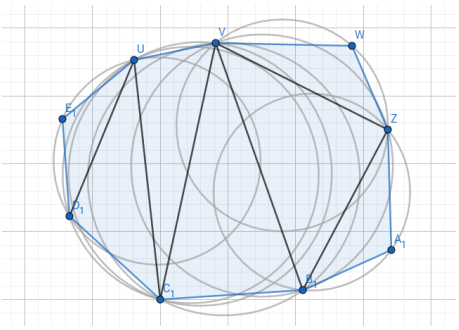

图 2.9　Delaunay triangulation 的分割结果

若进行三角剖分的点集中存在四点共圆，虽然计算其 Delaunay triangulation 时得到的结果不唯一，但仍不破坏其余性质。由于最小角最大的特性，DT 可以产生理论最优的划分。而根据三角剖分的性质，对凸多边形顶点点集进行三角剖分所得到的三角形集合即为待求的三角化结果。

一种利用分治思想在 $O(n\log n)$ 复杂度内计算点集的 Delaunay triangulation 方法如下：

首先将给定点集按照 x 坐标升序排列，如图 2.10所示。

图 2.10　排好序的大小为 10 的点集

一旦点集有序，我们就可以不断地将其平分成两个部分，直到子点集大小不超过 3。然后这些子点集可以立刻剖分为一个三角形或线段。

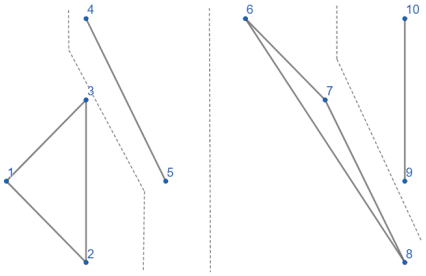

图 2.11　分治为包含 2 或 3 个点的点集

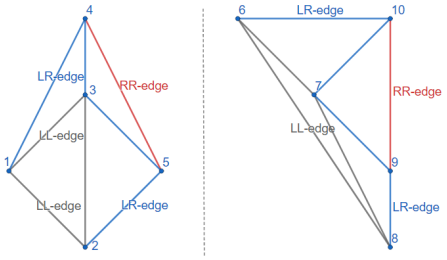

图 2.12　三种不同的边

在分治回溯的过程中，已经剖分好的左右子点集可以依次合并。合并后的剖分包含 LL-edge（左侧子点集的边）。RR-edge（右侧子点集的边），LR-edge（连接左右剖分产生的新的边），如图 2.12 LL-edge（灰色），RR-edge（红色），LR-edge（蓝色）。对于合并后的剖分，为了维持 DT 性质，我们可能需要删除部分 LL-edge 和 RR-edge，但我们在合并时不会增加 LL-edge 和 RR-edge。

合并左右两个剖分的第一步是插入 base LR-edge，base LR-edge 是最底部的不与任何 LL-edge 及 RR-edge 相交的 LR-edge。

然后，我们需要确定下一条紧接在 base LR-edge 之上的 LR-edge。比如对于右侧点集，下一条 LR-edge 的可能端点（右端点）为与 base LR-edge 右端点相连的 RR-edge 的另一端点（6, 7, 9 号点），左端点即为 2 号点。

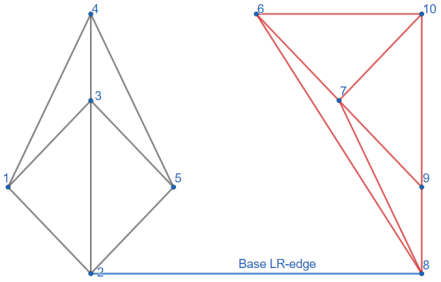

图 2.13　合并左右剖分

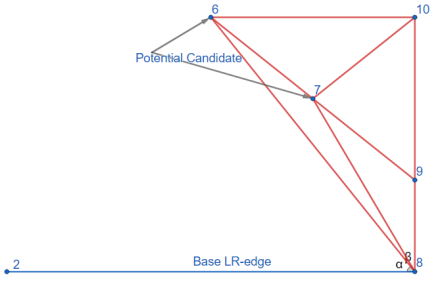

图 2.14　下一条 LR-edge 的可能位置

对于可能的端点，我们需要按以下两个标准检验：

1. 其对应 RR-edge 与 base LR-edge 的夹角小于 180 度。即，新选择的端点在当前 base LR-edge 的“上方”。

2. base LR-edge 两端点和这个可能点三点构成的圆内不包含任何其它可能点。

如图 2.15，6 号可能点所对应的绿色圆包含了 9 号可能点，而 7 号可能点对应的紫色圆则不包含任何其它可能点，故 7 号点为下一条 LR-edge 的右端点。

对于左侧点集，我们做镜像处理即可。

当左右点集都不再含有符合标准的可能点时，合并即完成。当一个可能点符合标准，一条 LR-edge 就需要被添加，对于与需要添加的 LR-edge 相交的 LL-edge 和 RR-edge，将其删除。

当左右点集均存在可能点时，判断左边点所对应圆是否包含右边点，若包含则不符合；对于右边点也是同样的判断。在未出现四点共圆的情况下每次只会有一个可能点符合标准，否则任选其一即可。

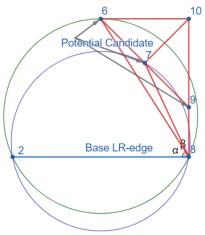

图 2.15　检验可能点

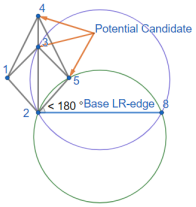

图 2.16　镜像处理另一半点集

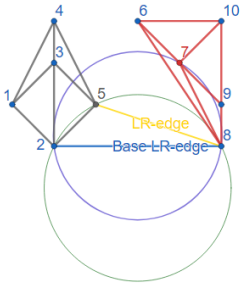

图 2.17　下一条 LR-edge

重复以上步骤即可完成 Delaunay triangulation 的构建。

由此我们完成了对任意多边形的三角化。

3　通过构建受限制 Delaunay 三角网将多边形三角化

在论文编写期间，受分治 Delaunay triangulation 算法以及一些已有研究[9] 的启发，我想到了另一种解决本问题的思路。

我们已知，Delaunay triangulation 广泛被认为是三角剖分中的最优结果，其具有多种优良性质，如最小角最大，局部修改只影响小范围区域等。而在三角剖分中，我们能够产生选择的最小单位如图 3.1所示，是一个包含四个点的凸多边形。根据内部两条对角线的选择方式，有着两种不同的分割结果。标准的 Delaunay Triangulation 根据三角形外接圆相关的信息用于评估两条边的选择，而两条对角线的选择可以在不影响四边形外部的情况下进行切换。

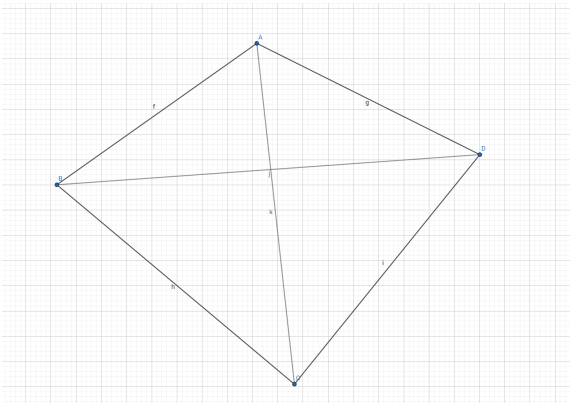

图 3.1　三角剖分中的最小单位

从另一个角度来看，对于一个完成剖分后的三角网，除去外边界以外的边，所有的边都恰好包含于两个三角形，而这两个三角形可能会组成上述的凸四边形。在这种情况下，我们可以对这条边，以及这条边所对应的四边形的另一条对角线进行一次“交换”，在仅有小范围改动的情况下对三角剖分进行修正。

由此我们可以考虑一种思路，即在点集的 Delaunay 三角网中，强行加入原多边形已有的边作为限制，并在改动尽量小的情况下维持答案为合法的三角剖分。

此算法要求存在包含全部原多边形边的合法三角网，即原多边形的边不能相交。因此无法用于复杂多边形，但复杂多边形可以通过一定的预处理后变为可以执行本算法的状态。

3.1 生成 Delaunay 三角网

使用分治算法生成 Delaunay 三角网的步骤不再赘述。不过，为了便于后续计算，我们采用与之前不同的数据结构用于存储信息。

我们采用数组记录整个多边形的点集，并分别对每个顶点按极角序记录其在三角网中的连边。

计算完 Delaunay 三角网后，我们便获得了一份初始的三角剖分结果。这份剖分结果中不包含原多边形连边的信息，是接下来计算答案的基础。

3.2 向三角网中加入强制连边

接下来，我们将原多边形的每一条边强制加入 Delaunay 三角网。在一条边的加入期间，我们最终将会删除所有与之冲突的边，并补充一定数量的次优边以保持三角网的性质。

下面，我们选择一条边，开始将其加入 Delaunay 三角网中。

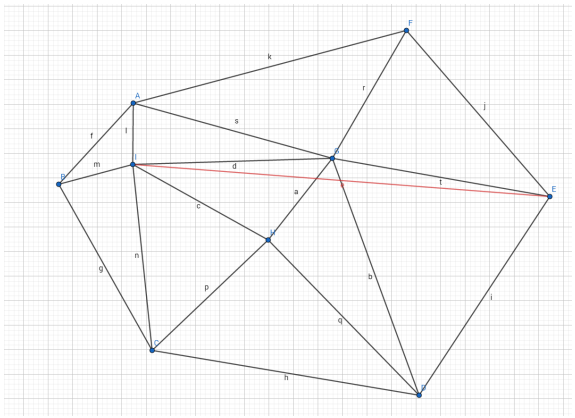

图 3.2 向三角网中强制加入限制边

一种最简单的情况是，当前的边已经被原三角网所包含。在这种情况下，我们不需要对原三角网进行修改，只需要标注当前边属于不可移除的限制边即可。

另一种情况下，当前边的加入仅涉及了两个三角形，也就是上文中图 3.1的情况，存在两条冲突的”对角线“，代表我们在当前四边形的剖分结果中需要强制选择相对较差的结果。也就是说明，此时在四边形对角线的二选一中需要进行一次“交换”操作，即可完成对三角网的修正。

最后一种情况则是，新增的连边破坏了多个三角网。如图 3.2所示，新加入的红色边与多条已有的边相冲突。在这种情况下，我们只需依次处理每条与新增边

冲突的旧边即可。我们按冲突点顺序遍历所有的冲突边。对于一条冲突边，我们可以寻找这条边对应的两个三角形，并对其尝试进行一次“交换”操作。在进行两次”交换“后，如图 3.3所示，我们成功将新的限制边加入了三角网中。

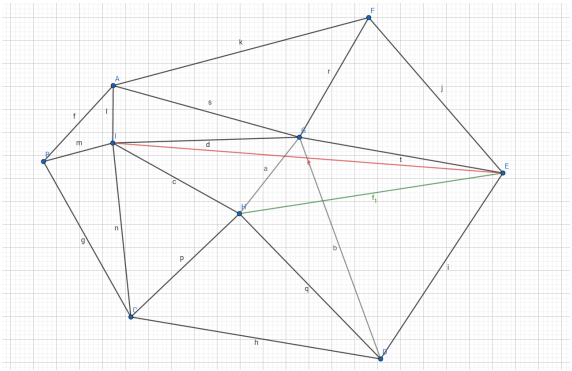

图 3.3　通过交换操作修正三角网使其满足要求

若两个三角形组成的四边形为凸四边形，且四边形有三个点在新增边同侧或为新增边端点，则“交换”操作结束后即可消除一条冲突边。若组成的四边形为凹四边形，无法执行“交换”操作，我们暂时跳过该点，在后续操作中将其完成消除。

如图所示，我们以新增边的顶点之一为起点，向另一个顶点在新增边的两侧各连接一条最贴近新增边的折线。该折线满足性质，从折线中间某条边向新增边方向搜索，下一条边必为冲突边。两条折线产生了一个包含新增边的多边形，且多边形内部仅存在冲突边，所有冲突边均在多边形内。易证这样的折线在新增边的两侧有且仅有一条。

将可以直接执行“交换”操作的边处理完后，有较小的概率仍存在未被消除的冲突边。如图 3.4所示，此种情况下多边形不存在可以进行交换的边。此时我们选择冲突边中最长的一条，对其进行一次“交换”操作。操作结束后，根据李立新的证明[10]，必然存在了新的可消除冲突边。多次处理后我们最终可以对全部冲突边进行消除，并成功将新增边强制加入三角剖分结果作为约束。

由此，我们可以构建一个包含原多边形中全部连边的点集三角剖分结果。

3.3　删除多边形外部的多余边

从之前对点集的三角划分到最终多边形三角剖分的结果仍存在一定的距离。完成点集的三角剖分结果计算后，剖分结果所形成的多边形是包含原多边形所有节点的凸包。我们需要从剖分结果中去除不属于原多边形的部分，保留的部分即

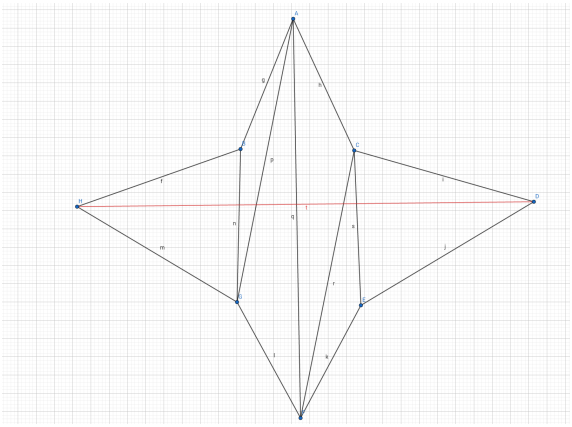

图 3.4　不存在可以直接通过交换消除的冲突边的特殊情况

为三角剖分的答案。

我们采用如下方法删除不属于原多边形的三角形。

首先通过预处理判断多边形中每条边界的性质，确认边界包围的区间是否属于多边形；同时标准化多边形的旋转方向，使多边形的外边界为顺时针旋转，而孔洞的边界则为逆时针旋转。

枚举剖分结果中的三角形。若三角形中三条边均属于原多边形，则直接根据对应多边形边界的属性判断内部是否为孔洞；否则任选一条不属于原多边形的边，任选该边的一个端点，判断该边是否落在了端点对应的内角上。若边在内角，说明当前三角形属于端点所对应的子多边形边界内部，与子多边形类型相同；否则则说明与子多边形类型相反。删除属于孔洞类型的三角，保留属于多边形内部的三角形。由此，我们便可以得到对原多边形的三角剖分。

4　代码结果及实验验证

我们分别采用手工构造和随机生成的数据对算法进行测试。

考虑到对凸多边形的三角划分有确定性的最优解，为了便于对比，我们的代码实现在所有子多边形均为凸多边形后即产生输出。

图 4.1是手工构造的一组测试用例，具有大量互相嵌套，位置关系复杂的孔洞。我们将主要通过本组数据对算法进行测试。

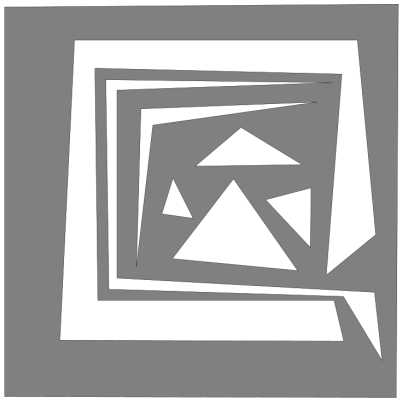

图 4.1　具有多个复杂孔洞的多边形

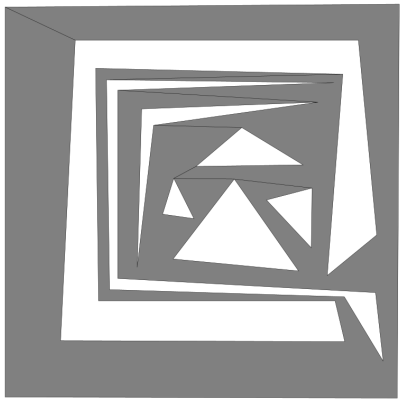

图 4.2　对多边形的孔洞进行消除

首先，如图 4.2, 算法对多边形的所有孔洞进行合并。通过数次切割，所有的内部孔洞通过一条宽度为 0 的桥与多边形外界相连，多边形的所有边界合并为一体，以便于接下来的处理操作。

在这一步，经测试发现，合并方式的不同对最终结果并没有非常大的影响。究其原因是，略经优化后的朴素做法所选择的边并不会特别差；而少数相对较差的边即使在合并孔洞这一步未被选中，在接下来的初步分割或是最后的 Delaunay 三角网生成中仍然大概率作为一条切割边。而朴素算法相比于准确的计算出总长最短的切割边，可以较大程度的节省运算所耗时间。

在完成对多边形孔洞的合并之后，我们便可以开始对凹多边形进行分割了。在此步，我们的目标是将原有的凹多边形分割成相对数目较少的凸多边形，并通过控制 2.3.5.1中提到的修正值在效率和精准度之间做出平衡。

图 4.3,4.4是修正值分别取 0.5 和 1 时的分割结果。可以看到，修正值较小时，这一步的切分较为保守，可以保留下相对更规整的凸多边形，同时避免了一部分判断失误而产生的错误划分。算法在运行本测试用例时消耗时间仅需 40 毫秒，其

中修正值取 1 时运行时间相比修正值取 0.5 时减少约 3 毫秒。

综合两图可以看到，经过这一步，多边形中的全部凹顶点均被分割为数个凸顶点，所有的多边形均被转化为凸多边形。接下来分别对每个位置调用 Delaunay 三角剖分算法即可得出最后对多边形三角化的结果。

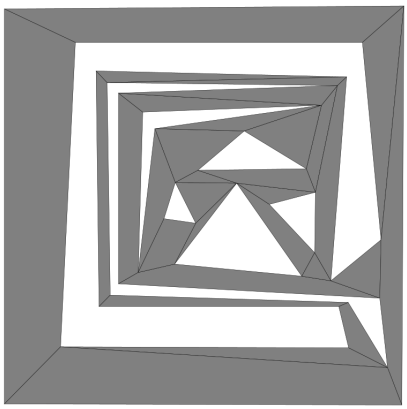

图 4.3　取较小修正值的结果

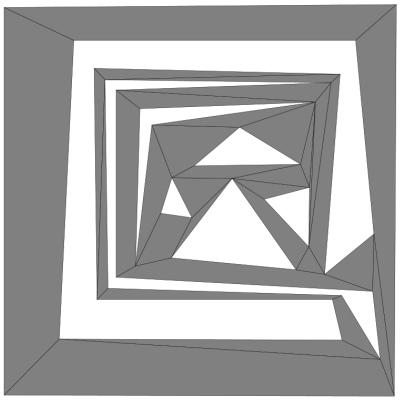

图 4.4　取较大修正值的结果

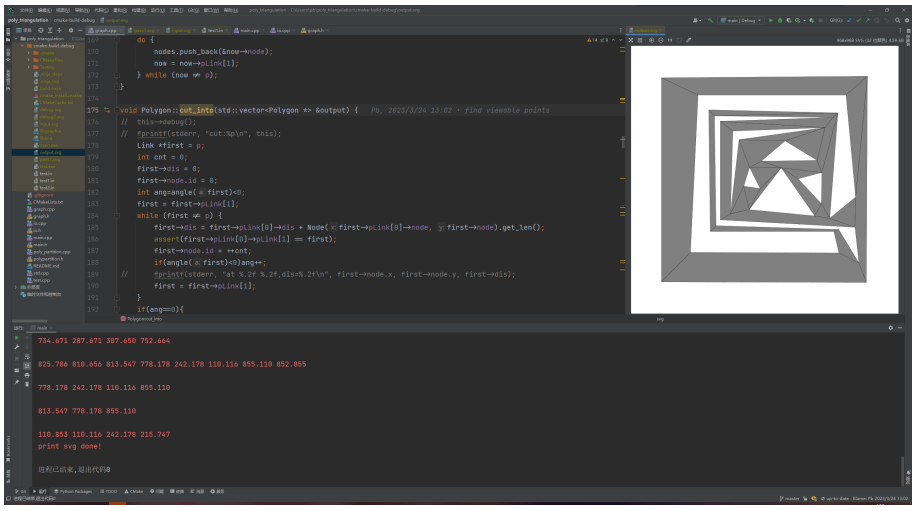

图 4.5　实验截图

与常用的耳切法等算法相比，本文的算法实现具有高效，准确，可自定义等多种优势，可以较好的解决多边形三角化的问题。

参考文献

[1] EDELSBRUNNER H, LI X-Y, MILLER G, et al. Smoothing and cleaning up slivers[A]. Proceedings of the thirty-second annual ACM symposium on Theory of computing[C], 2000 : 273 – 277.

[2] 刘学军, 符锌砂. 三角网数字地面模型的理论, 方法现状及发展 [J]. 长沙交通学院学报, 2001, 17(2) : 24 – 31.

[3] 秦绪佳, 欧宗瑛, 侯建华. 医学图像三维重建模型的剖切与立体视窗剪裁 [J]. 计算机辅助设计与图形学学报, 2002, 14(3) : 275 – 279.

[4] EBERLY D. Triangulation by ear clipping[J]. Geometric Tools, 2008 : 2002 – 2005.

[5] 闵卫东, 唐泽圣. 二维任意域内点集的 Delaunay 三角划分生成算法 [J]. 计算机学报, 1995, 18(5) : 365 – 371.

[6] LEE D-T, SCHACHTER B J. Two algorithms for constructing a Delaunay triangulation[J]. International Journal of Computer & Information Sciences, 1980, 9(3) : 219 – 242.

[7] HERSHBERGER J. Finding the Visibility Graph of a Simple Polygon in Time Proportional to Its Size[A/OL]. SCG '87 : Proceedings of the Third Annual Symposium on Computational Geometry[C], New York, NY, USA : Association for Computing Machinery, 1987 : 11–20.
https://doi.org/10.1145/41958.41960.

[8] 金文华, 何涛, 唐卫清, et al. 简单多边形可见点问题的快速求解算法 [J]. 计算机学报, 1999, 22(3) : 275 – 282.

[9] CHEW L P. Constrained delaunay triangulations[A]. Proceedings of the third annual symposium on Computational geometry[C], 1987 : 215 – 222.

[10] 李立新, 谭建荣. 约束 Delaunay 三角剖分中强行嵌入约束边的多对角线交换算法 [J]. 计算机学报, 1999, 22(10) : 1114 – 1118.

致谢

在实验代码和论文的编写过程中，我的诸多朋友，导师，同学等人为我提供了许多帮助。

感谢我的导师傅佑铭老师，与我一起确定题目，指导我完成开题报告，任务书和论文的编写，为我最终的论文提供了大量详细明确的修改建议。感谢我的舍友们，以及校内一起参加 ACM 竞赛的各位同学，协助我敲定代码中具体算法的实现细节，帮我排查算法和代码中的漏洞。感谢我的朋友们，在我寻找参考资料时为我指明了探索的方向，让我找到了指导我代码的关键论文。感谢我的辅导员赵玥老师，为我们的毕业设计提供了许多帮助和下午茶。感谢我的女朋友，在我实现代码，编写论文时一直陪伴着我，帮助我缓解工作的劳累。

附录 A　代码

本文相关的代码中，已经实现的部分全部上传至了 GitHub。

https://github.com/SkyCrystal/poly_triangulation

代码中目前已经实现了输入输出以及 3，4 章中代码的核心部分。

代码分为三个部分：图处理，svg 模块，调用逻辑。其中图处理和 svg 部分分别编写，并以静态链接库的形式供 main.cpp 调用。

代码目前可能仍然存在一些 bug，欢迎对其进行指正和 pr。