采用前沿推进法生成
二维非结构化网格的算法实现

姓名：朱光穆

班级：能动 A02

学号: 2009031200

指导老师：陈斌

目录

1 非结构化网格简介

计算传热学（或数值传热学）是传热学的一门分支学科，采用数值方法利用计算机来求解各类热量传递问题。数值计算中用离散的网格来代替原物理问题中的连续空间，网格中的节点则代表所求解物理量的几何位置。

1.1 网格的分类

按照构造方法分类，网格可以分为结构化（Structured）、块结构化（Block-structured）及非结构化（Unstructured）三种。在结构化网格中，任意一个节点的位置可以通过一定的规则予以命名。采用块结构化网格时，计算区域需分解成两个或两个以上的子区域，在每一个区域中均用一种形式的结构化网格，各子区域之间，可以互不重叠，也可以有一部分重叠。在非结构化网格中，节点的位置无法用一个固定的法则予以有序命名。非结构化网格应用于有限差分法及有限容积法。

1.2 采用非结构化网格的意义

从严格意义上讲，结构化网格是指网格区域内所有的内部点都具有相同的毗邻单元。结构化网格有很多优点：

1. 它可以很容易地实现区域的边界拟合，适于流体和表面应力集中等方面的计算；
2. 网格生成的速度快；
3. 网格生成的质量好；
4. 数据结构简单；
5. 对曲面或空间的拟合大多数采用参数化或样条插值的方法得到，区域光滑，与实际的模型更容易接近；

它最典型的缺点是适用的范围比较窄。尤其随着近几年的计算机和数值方法的快速发展，人们对求解区域的复杂性的要求越来越高，在这种情况下，结构化网格生成技术就显得力不从心了。

同结构化网格的定义相对应，非结构化网格是指网格区域内的内部点不具有相同的毗邻单元。即与网格剖分区域内的不同内点相连的网格数目不同。结构化网格和非结构化网格有相互重叠的部分，即非结构化网格中可能会包含结构化网格的部分。

非结构化网格技术发展于上世纪六十年代，主要弥补了结构化网格不能够解决任意形状和任意连通区域的网格剖分的缺欠。由于非结构化网格的生成技术比

较复杂，随着求解区域复杂性的不断提高，对非结构化网格生成技术的要求越来越高。非结构化网格生成技术中只有平面三角形的自动生成技术比较成熟，平面四边形网格的生成技术正在走向成熟。而空间任意曲面的三角形、四边形网格的生成，三维任意几何形状实体的四面体网格和六面体网格的生成技术需要解决的问题还非常多。主要的困难是从二维到三维以后，待剖分网格的空间区域变得非常复杂。除四面体单元以外，很难生成同一种类型的网格，此时需要在各种网格形式之间进行过渡，如金字塔形，五面体形等等。

1.3 非结构化网格的生成方法

生成非结构化网格的方法可以大致分为三类：

1. 前沿推进法（advancing front method）；
2. Delaunay 三角形化方法（Delaunay triangulation）；
3. 其他方法；

就方法本身而言，前沿推进法更具有启发性（heuristic），而 Delaunay 三角形化方法则严格基于计算几何的基础之上。其他方法包括：集上述两法优点于一体的一些方法，以及在构思上完全与上述方法不同的所谓四叉树法（quadtree）或八叉树法（octree method）等。

2 前沿推进法的基本原理

生成非结构化网格的前沿推进法是在 1985 年由 Lo 提出的，此后该方法得到不断的改进与发展。以下简单介绍 Lo 提出的网格生成方法的原理。

2.1 前沿推进法的实施要点

在前沿推进法中，从边界上的网格点所形成的一系列线段（前沿）出发，逐一与区域内的点形成三角形，所形成的三角形的新边界加入前沿边的行列，而生成该三角形的出发边则从前沿行列中消去，如此不断向前推进，直到前沿行列为空。不同的前沿推进法的实施主要有三点区别：

1. 如何生成边界上的前沿；
2. 如何向区域内加点；
3. 如何推进前沿与内点连成三角形；

Lo 提出的方法主要有以下四步。

2.1.1 边界网格点的设置

1. 外边界逆时针方向输入点，内边界顺时针方向输入点；
2. 将边界点编号；

2.1.2 区域内网格点的生成

1. 确定平均间距和允许的间距浮动范围；
2. 确定区域的范围y_{max}与y_{min}；
3. 在y_{max}和y_{min}之间插入一系列水平线，线间距在许用间距内（图 2-1）；
4. 对水平线在求解域中的部分沿x方向进行插值，要求同一水平线上点间距在许用间距内；
5. 筛选所有内点，将与边界点间距小于许用间距的点删除；

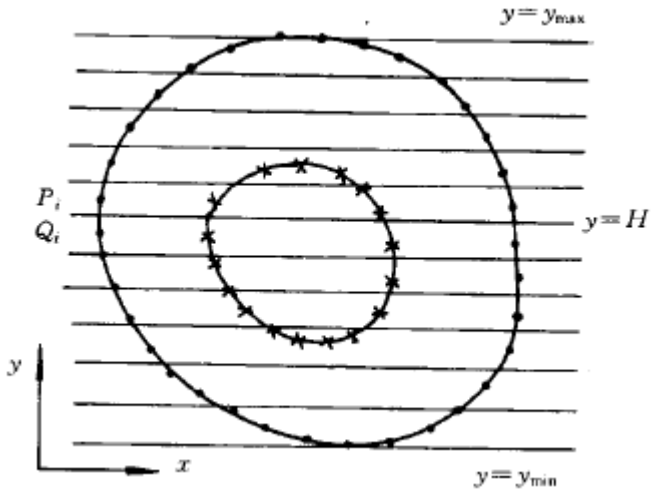


图 2-1

2.1.3 区域内三角形的生成

1. 把边界上所有有向线段（按前面规定的方向）都设置为前沿边，组成集合Γ，所有内点组成集合Λ；
2. 从某前沿出发（例如图 2-2（a）中的$\overrightarrow{15}$），在其左侧内寻找按上述方法生成的内点C_i，找出其中使$\left(\overrightarrow{1C_i}^2+\overrightarrow{C_i5}^2\right)$为最小的点，记为 C，判断$\overrightarrow{1C}$与$\overrightarrow{C5}$是否与其他前沿相交，如不相交，则 C 为所寻找内点，它与$\overrightarrow{15}$组成△15C；
3. 将线段$\overrightarrow{15}$从前沿集中删去，将$\overrightarrow{1C}$和$\overrightarrow{C5}$加入前沿集Γ中（图 2-2（b）），同时将网格点从集合Λ中删去；
4. 重复 1~3 步直到Γ变空，Λ变空；

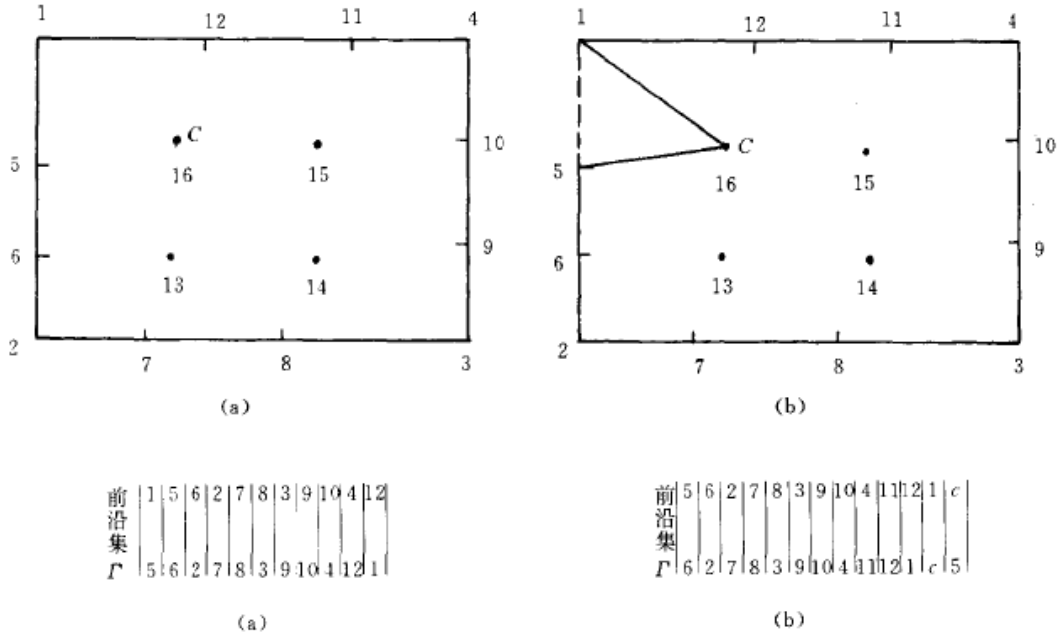


图 2-2

2.1.4 网格的光顺化

用上述方法生成的非结构化网格需要进行光顺化处理，即设法使所生成的三角形单元尺寸变化逐渐接近地进行，并使三角形更加接近于等边三角形。常用的光顺化方法有：

1. 移动内部网格点的位置，使它处于由共享此点的三角形所组成的多边形的中心。

$$(x_i, y_i)^{new} = (x_i, y_i)^{old} + \frac{\omega}{n}\sum_{k=1}^{n}[(x_k, y_k) - (x_i, y_i)] \qquad (2\text{-}1)$$

式中(x_i, y_i)为被移动的网格点坐标，(x_k, y_k)为共享(x_i, y_i)点的所有三角形单元的顶点坐标，n为共享该顶点的三角形顶点个数，ω为松弛因子；

2. 对角线交换法。对个别三角形单元，通过将两相邻三角形所组成的四边形的对角线交换来改善三角形单元的品质；
3. 弹簧比拟法。将三角形边假定为弹簧，具体过程复杂；

图 2-3 是简单的多连通区域非结构化网格的生成示意，其中图 2-3（h）是光顺化处理后的网格。

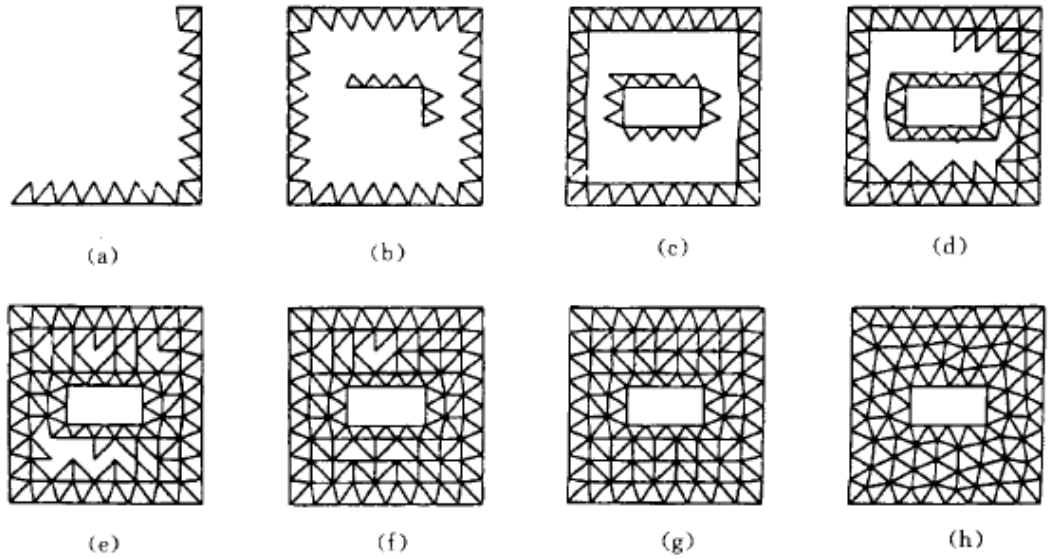


图 2-3

2.2 上述前沿推进法的改进与发展

上述生成非结构化网格的方法中，内点是按边界上网格的平均距离来设置的，这样边界上网格点疏密特性就无法传递到区域内部去；但同时该方法仅限于二维三角形单元。因此不少学者进行了不同方向的改进：

1. 引入背景网格从而控制局部地区的单元特性参数；
2. 边界上网格点的位置由背景网格生成；
3. 在前沿推进过程中来确定网格点的位置；
4. 背景网格的参数由人工设定发展为采用椭圆形偏微分方程来完成；
5. 二维问题的单元由三角形向四边形扩展；
6. 二维问题向三维问题扩展；
7. 由前沿推进法发展为层面推进法；

3 前沿推进法的程序实现

本程序采用 Matlab 进行算法开发，采用 C++编写。

3.1 边界的读入

原始边界点从文件读入，程序约定使用 OFF（Object File Format）文件格式。OFF 文件格式来自普林斯顿大学的普林斯顿形状标准（Princeton Shape Benchmark），文件通过描述物体表面的多边形来表示一个模型的几何结构，遵循以下标准：

1. OFF 文件为 ASCII 文件，以 OFF 关键字开头；
2. 下一行是该模型的顶点数，面数和边数，边数可以忽略，对模型不会有影响（可以为 0）；
3. 顶点以 X、Y、Z 坐标列出，每个顶点占一行；
4. 在顶点列表之后是面列表，每个面占一行，对于每个面，首先指定其包含的顶点数，随后是这个面所包含的各顶点在前面顶点列表中的索引；

示例格式如下：

```
OFF
8 6 0
-0.500000 -0.500000 0.500000
0.500000 -0.500000 0.500000
-0.500000 0.500000 0.500000
0.500000 0.500000 0.500000
-0.500000 0.500000 -0.500000
0.500000 0.500000 -0.500000
-0.500000 -0.500000 -0.500000
0.500000 -0.500000 -0.500000
4 0 1 3 2
4 2 3 5 4
4 4 5 7 6
4 6 7 1 0
4 1 7 5 3
4 6 0 2 4
```

本程序从 OFF 文件中读入原始边界，包括内边界，要求第一条边界作为外边界。格式如下：

OFF

定点数 边界数 （略）

……

顶点坐标

……

边界顶点数 …… 边界顶点编号 ……

……

函数原型为： bool load_off(const std::string&, std::vector<std::vector<Point>>&)；

每条边界以向量形式存入容器，函数会检查 OFF 文件是否被损坏或者格式不正确，并返回布尔型。

3.2 边界的检查

即使 load_off 函数返回 true，边界也可能存在下面 3 个问题：

1. 内边界顶点位于外边界之外；
2. 边界线自交（这会导致负面积），或者相交；
3. 内外边界方向不符合要求；

这要求我们检查并且修正读入的边界。函数 check_off 负责调用函数 _is_inner_outof_outer 检查内点是否越界，调用函数_has_intersecting_lines 检查是否有边界自交或相交，调用函数_check_direction 修正错误的边界方向。函数原型如下：

bool check_off(std::vector<std::vector<Point>>&);

bool _is_inner_outof_outer(const std::vector<std::vector<Point>>&);

bool _has_intersecting_lines(const std::vector<std::vector<Point>>&);

void _check_direction(std::vector<std::vector<Point>>&);

如果内点位于外边界之外或者边界有交线，函数 check_off 返回 false 并退出程序，否则返回 true。

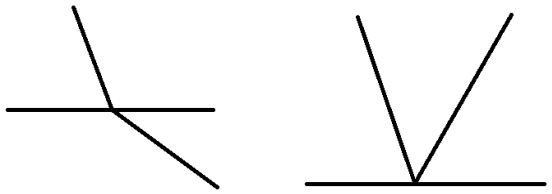
3.3 边界点插值

边界检查成功后，程序会提示用户输入点距 VStep，程序默认使用±10%作为许用点距范围，之后程序对每条边界的每条边用许用点距进行插值。依照原始边界容器的顺序，每条边界插值后的点坐标依序以矩阵形式存入容器中。

函数原型为：std::vector<matrix> boundary_interpolation(const std::vector<std::vector<Point>>&, double);

3.4 内点插值

边界点插值后，程序会尝试以许用间距在y_{max}与y_{min}之间插入一系列水平线，如果无法满足要求，将报错返回，并提示输入更小的间距。每条水平线求其与所有边界的交点。特别地，如果交点是原始边界顶点，需要检查其属于以下哪种情况（图 3-1）：



（a）　　（b）

图 3-1

对于图 3-1（b）交点，不计入水平线与边界交点。对于同一水平线上一系列交点，按x增大方向排序，奇偶数点之间按许用间距插值，插值后内点坐标以矩阵形式保存。

函数原型为：matrix inner_interpolation(const std::vector<std::vector<Point>>&, const matrix&, double);

对于现有的内点，需要剔除与边界点间距不符合要求的点。

函数原型为：matrix filter_inner_interpolation(const std::vector<matrix>&, const matrix&, double);

函数返回过滤后的内点坐标矩阵。

3.5 生成三角形网格

三角形网格的生成，核心在于充分完备地使用最小三角形覆盖计算域。为了方便计算，需要对已有数据进行组织。图 3-2 是数据组织的大致过程。

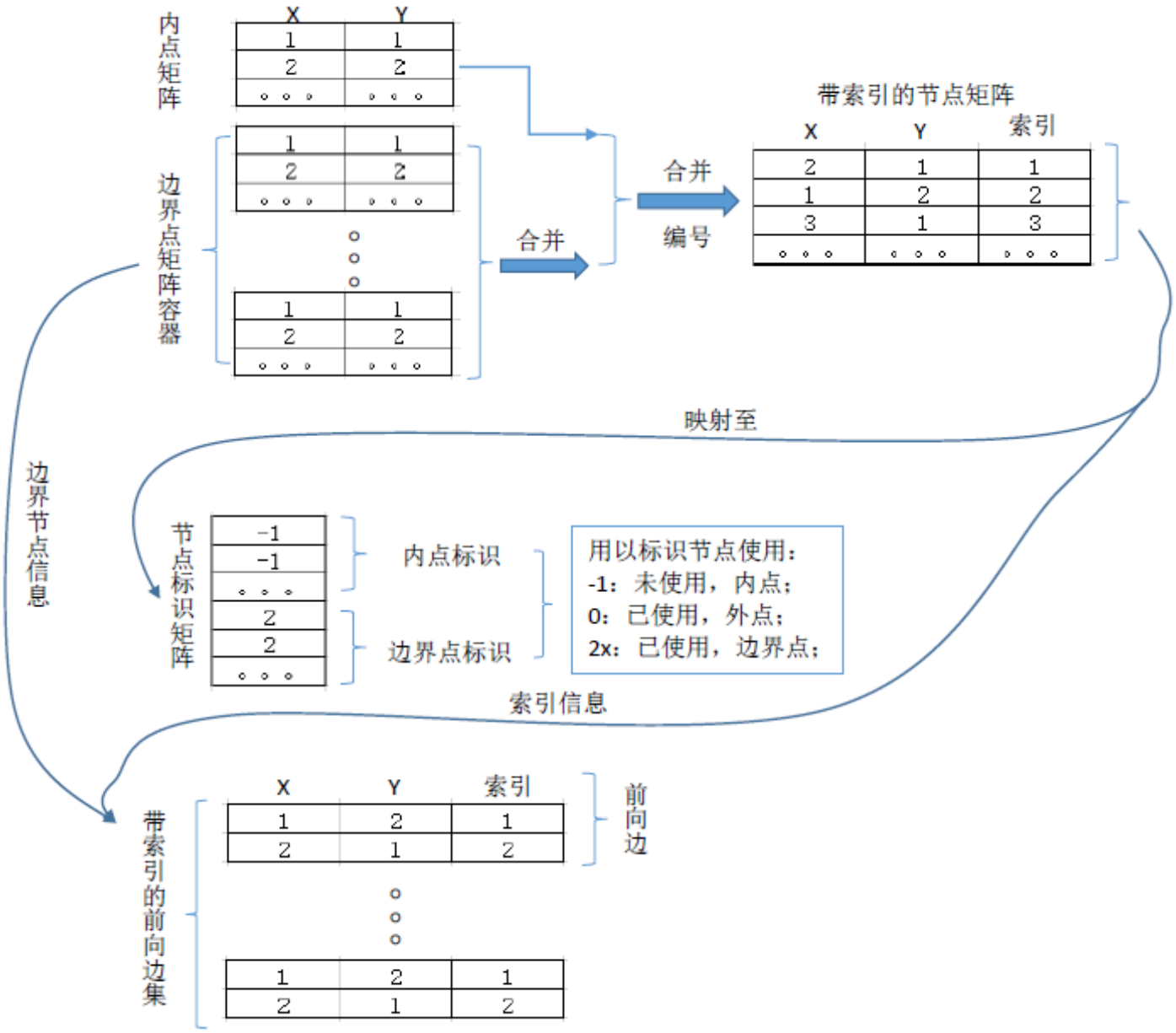


图 3-2

前向边是有方向的，由第一点指向第二点。建立如上数据结构后的步骤如下：

1. 从前向边集取出一条边，将此边上两顶点标识各减一；
2. 在这条边附近取一个足够大的范围，使之在这范围内一定存在节点；
3. 在已排序好的带索引的节点矩阵中搜索出此范围内的所有节点；
4. 对搜索出来的节点进行计算，求出满足以下条件的点：
   a) 此点必须是边界节点或者内节点，即节点标识不能为 0；
   b) 此节点必须在此前向边左侧；
   c) 前向边的两节点到此点的连线不能与现有边界交叉；
   d) 满足上述条件的点中$\left(\overrightarrow{1C_i}^2+\overrightarrow{C_i5}^2\right)$最小的点；
5. 如果找不到合适的点，提示计算出错，退出程序；
6. 将此三点加入三角形矩阵；
7. 检测新生成的两条前向边是否与现有边重合，重合则在前向边集删除现有边，并将其两顶点标识各减一；否则，在前向边集尾部添入此边，并将其两顶点各加一（如果此点原标识为-1，先置零再加一）；
8. 重复 1-7 步直到前向边集为空；

函数原型为：std::pair<matrix, matrix> create_triangles(const std::vector<matrix>&, const matrix&, double);

3.6 网格的光顺化

程序采用移动内部网格点的位置，使其处于由共享此点的三角形所组成的多边形的中心的方法对网格进行光顺化。公式 2-1 中的ω为松弛因子，该值在 0 到 1 之间为欠松弛因子，大于 1 为超松弛因子。理论上，欠松弛因子能改善收敛条件，超松弛因子能加快收敛速度，推荐值一般在 0.6 到 0.8。程序会提示用户输入松弛因子，默认是 1。

程序采用收敛系数和误差系数来控制收敛条件。误差系数指网格中节点移动的最大距离与节点间距的比值，收敛系数指两次相邻的收敛迭代的误差系数的差值。当两个系数都满足要求时结束迭代。

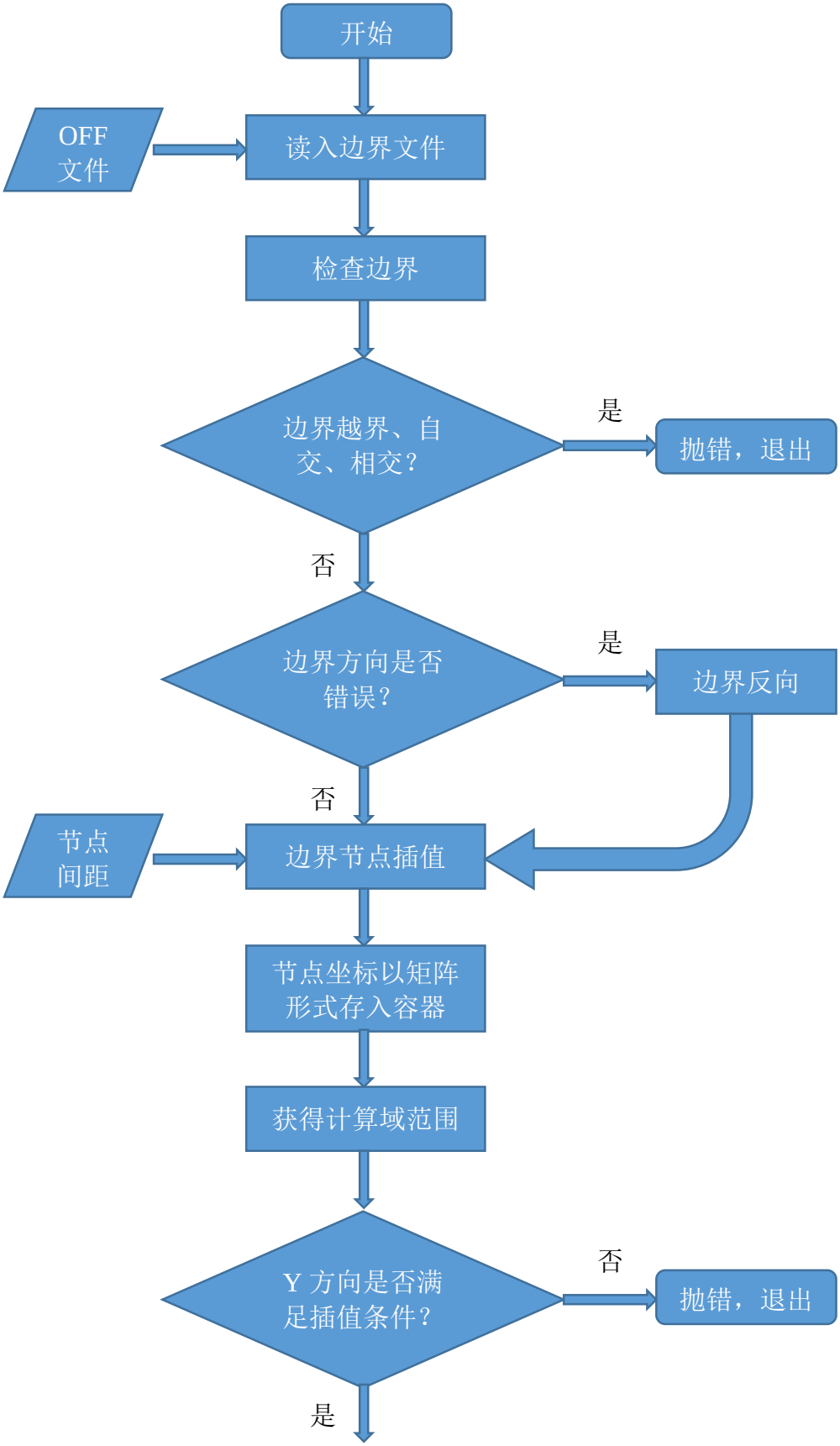
函数原型为：std::pair<matrix, double> smooth_triangles(matrix, unsigned, const matrix&, double, double, double, double);

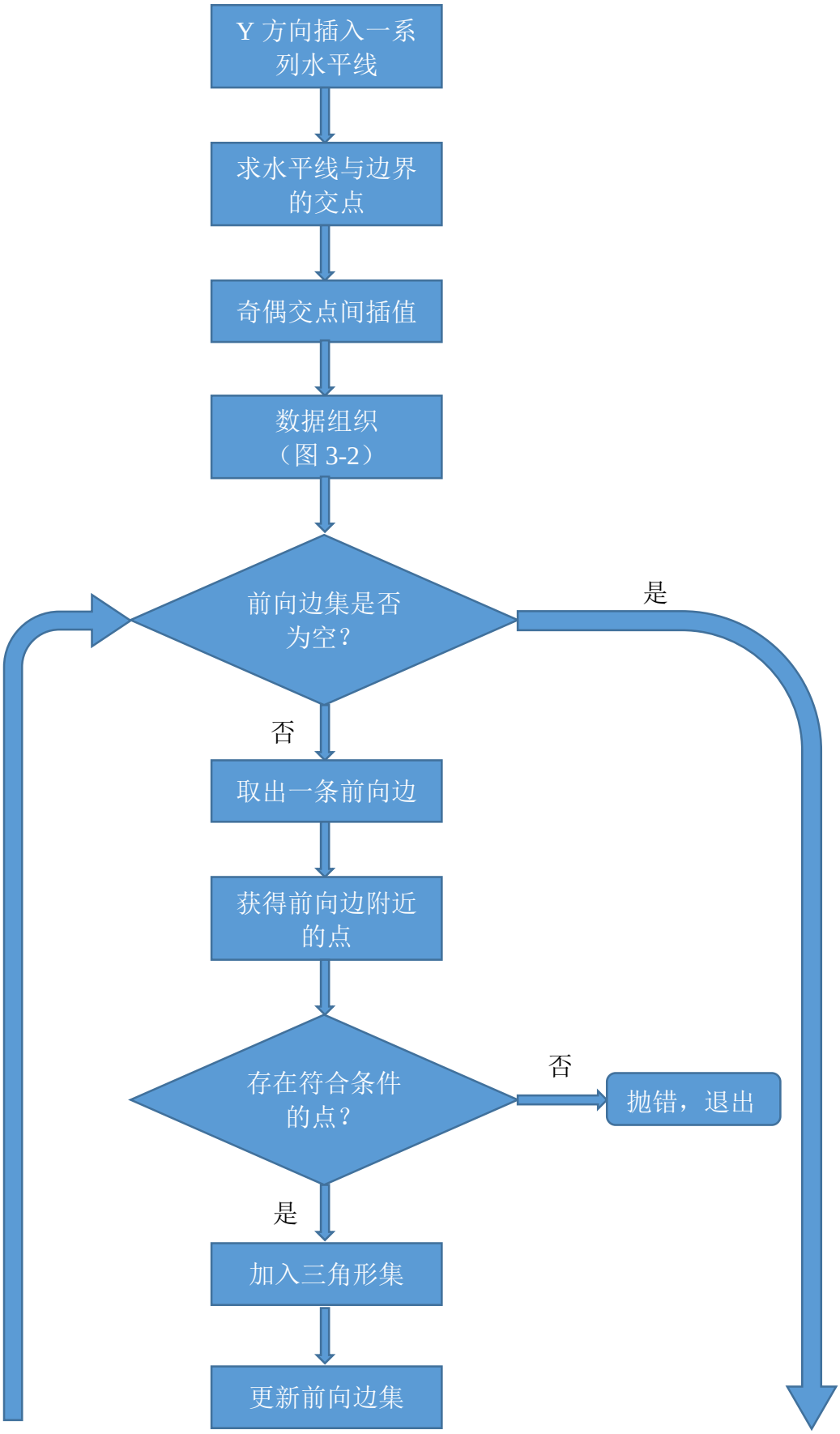
3.7 保存至文件

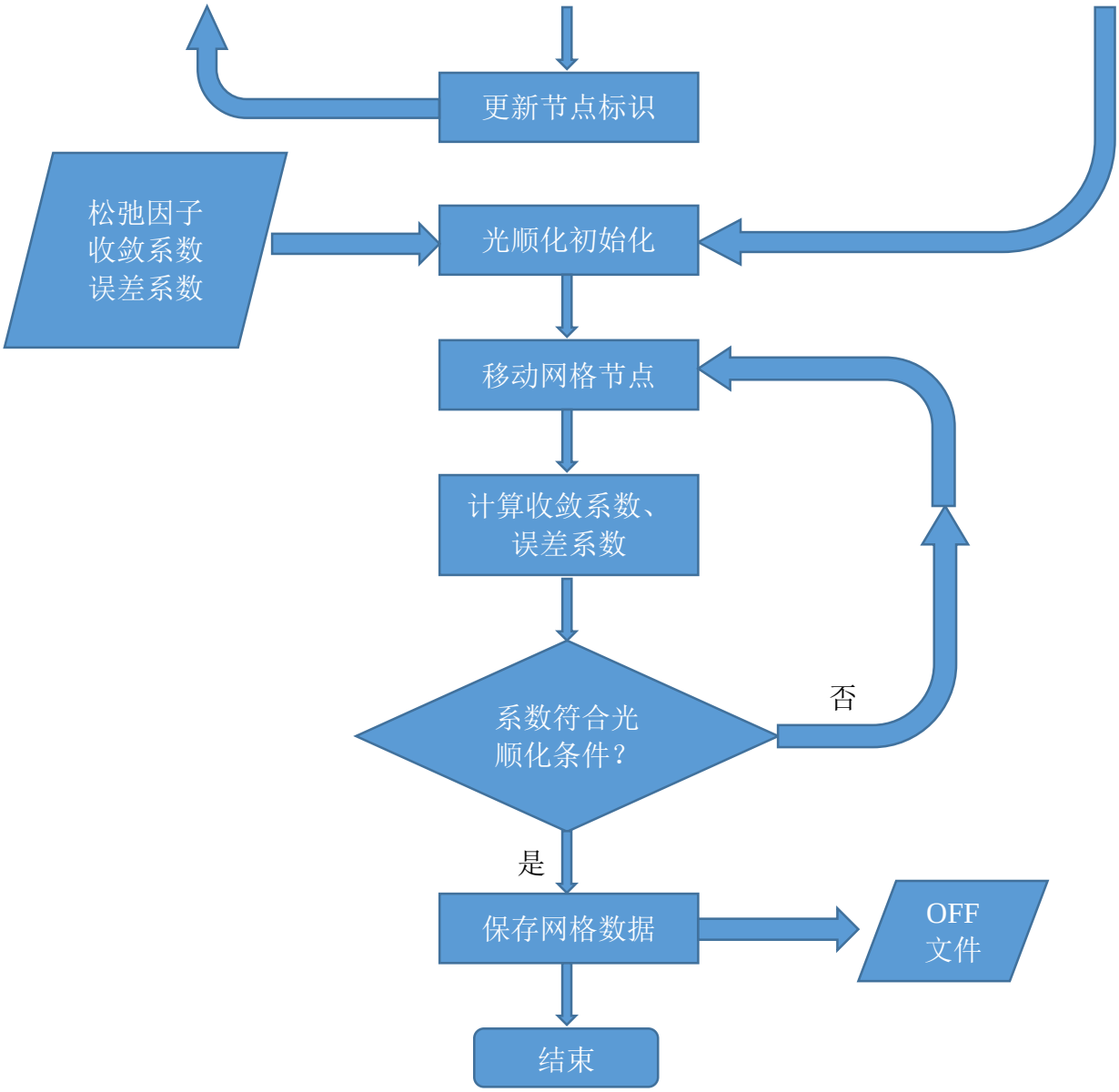
生成的三角形网格仍使用 OFF 文件格式输出，每个三角形作为一个面。

函数原型为：void save_OFF(const matrix&, const matrix&, const std::string&);

3.8 程序流程图


开始
OFF
文件
读入边界文件
检查边界
边界越界、自
交、相交？
是
抛错，退出
否
边界方向是否
错误？
是
边界反向
否
节点
间距
边界节点插值
节点坐标以矩阵
形式存入容器
获得计算域范围
Y 方向是否满
足插值条件？
否
抛错，退出
是


Y 方向插入一系列水平线
求水平线与边界的交点
奇偶交点间插值
数据组织（图 3-2）
前向边集是否为空？
是
否
取出一条前向边
获得前向边附近的点
存在符合条件的点？
否
抛错，退出
是
加入三角形集
更新前向边集

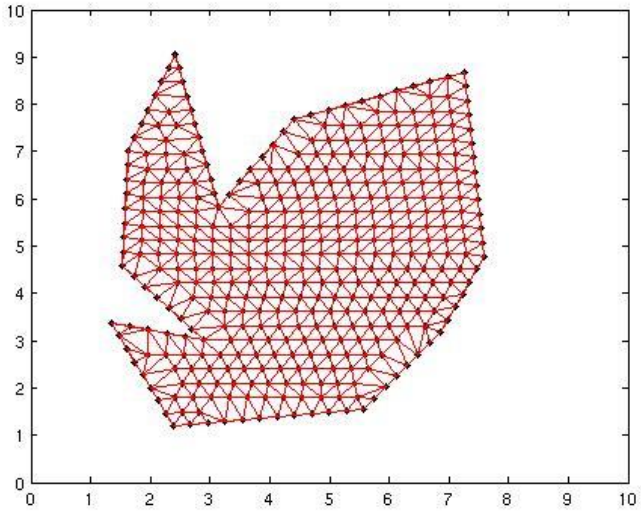

更新节点标识
松弛因子
收敛系数
误差系数
光顺化初始化
移动网格节点
计算收敛系数、
误差系数
系数符合光
顺化条件？
否
是
保存网格数据
OFF
文件
结束

4 算例分析

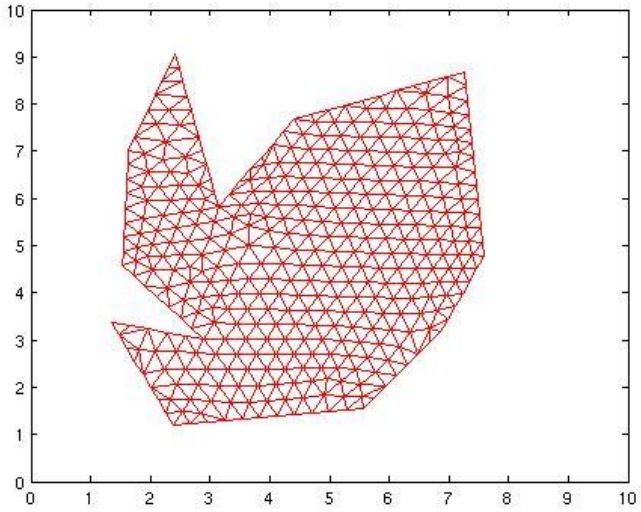
未加特别说明时，以下算例中均采用 0.3 节点间距，默认松弛因子，默认收敛系数和误差系数。

4.1 单连通域的网格

图 4-1 是单连通域的网格，（a）（b）分别是光顺化之前和之后的网格。



（a）



（b）

图 4-1

由图 4-1 可以得到如下结论：

1. 单连通区域能生成质量较好的网格，绝大部分网格大小形状符合要求；
2. 单连通区域在尖角处网格变形较为严重，个别网格过小、过多偏离正三角形；
3. 单连通区域在直线边界上有着较好的网格；
4. 单连通区域边界处的不合格网格并未影响内部网格，越远离边界，网格质量越好；
5. 光顺化对于网格质量有着明显的改善，体现为：
   a) 网格形状更加接近正三角形；
   b) 部分大小不合适的网格被重新调整大小；
   c) 相邻网格的大小和形状更加相近，突变更小；

4.2 单内边界的多连通域光顺化后的网格

图 4-2 是单内边界的多连通域光顺化后的网格。

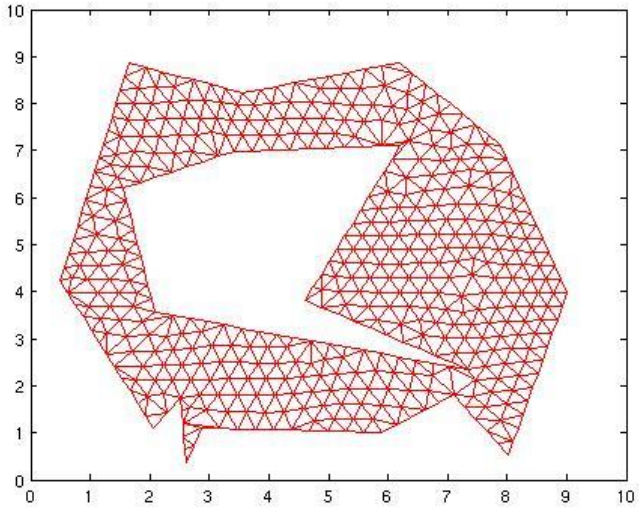


图 4-2

由图 4-2 可以得到如下结论：

1. 单内边界的多连通域的网格与上述单连通域的网格拥有相同的特性；
2. 与单连通域不同的是，单内边界的多连通域网格在内边界与外边界距离较近时，网格大小会偏大，形状会变形；

4.3 多内边界的多连通域光顺化后的网格（间距 0.3）

图 4-3 是多内边界的多连通域光顺化后的网格。

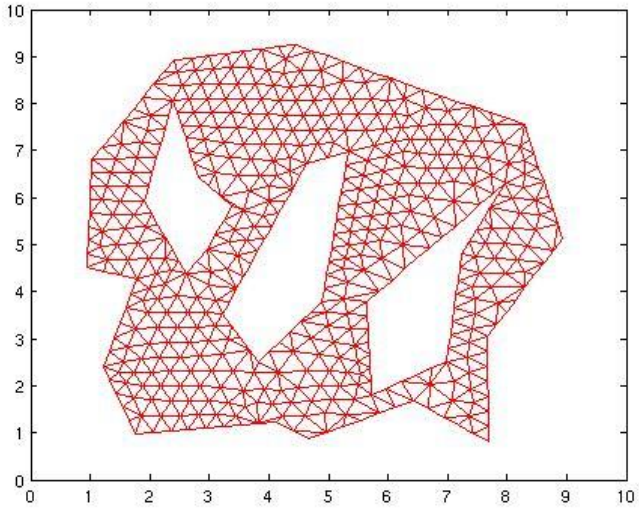


图 4-3

由图 4-3 可以得到如下结论：

1. 多内边界的多连通域的网格与上述单内边界的多连通域的网格拥有相同的特性；
2. 与单内边界的多连通域不同的是，多内边界的多连通域网格还在内边界与内边界距离较近时，网格大小会偏大，形状会变形；

4.4 多内边界的多连通域光顺化后的网格（间距 0.2）

图 4-3 是多内边界的多连通域光顺化后的网格。此节点间间距为 0.2：

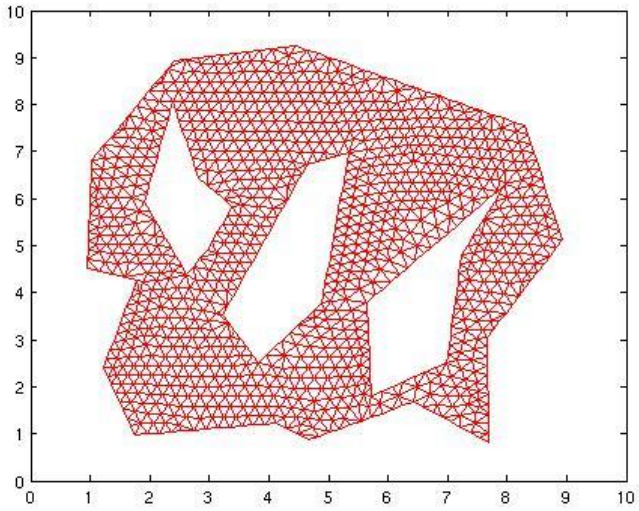


图 4-4

图 4-4 与图 4-3 对比可以得到如下结论：

1. 整体网格质量变好，大小形状符合要求的网格所占的比例显著提升；

2. 尖角处网格质量提升明显，部分网格大小和形状更加符合要求，甚至在某些尖角处，网格完全符合要求（坐标：（3.8，2.5））；
3. 边界处不符合要求的网格对内部网格影响显著减小；
4. 边界间距过小对内部网格质量的影响依然存在，但受影响的范围在缩小；

4.5 算例分析总结

由上面的分析可以发现，几乎在所有情况下，由本程序生成的网格均有良好表现，拥有均匀的网格分布，高比例的高质量网格，但同时依然存在少数不符合要求的网格。而良好的边界节点分布，不太小的边界间距，更小的节点间距，合理的光顺化，都会显著改善生成的网格。其中改善最为显著的方法是减小节点间距和光顺化，但过小的节点间距和过于复杂的光顺化，会大幅增加计算时间，此处需要进行取舍，以取得经济性和网格质量间的平衡。

5 不足与展望

由于时间仓促，以及本人水平有限，尽管经过了数周不懈的努力，本程序仍然存在一些不足。可改进之处主要体现在网格算法和程序设计两方面。

5.1 关于网格算法

5.1.1 存在的问题

1. 节点插值虽然规定了许用间距，但对于较小间距的节点间的插值，只能保证其大于最小许用间距，而不能保证小于最大许用间距，这是导致边界附近网格过大的主要原因之一；
2. 内节点的插值以一系列水平线为参考，在边界与 Y 轴方向有较大交角时，很容易导致靠近边界的内节点在筛选小于许用间距的节点时被删除，从而导致边界节点到内节点距离增加。这是导致边界附近网格过大的另一个主要原因；
3. 没有分区域分块处理，导致在生成三角形的过程中，每个新三角形要与每条前向边验证是否相交。性能分析显示，这种验证已成为性能瓶颈；而从图形上分析，这种验证绝大多数都是不必要的；
4. 移动到共享网格节点中心的光顺化方法，快速而实用，但不足够好。它能修正大部分不符合要求的网格，但对有些网格无能为力，例如严重变形的网格，和虽然节点间距过大，但节点却在中心的网格；

5.1.2 可能的解决方案

1. 灵活规定许用间距，在适当的区域缩小间距，加密节点；
2. 使用更好的内节点插值方法，在筛选内节点后需要考虑移动节点以缩小和边界间距；
3. 将计算域分块，在块内生成三角形再合并块，并调整块边界三角形；
4. 网格的光顺化采用对角交换法或弹簧比拟法等多种方法；

5.2 关于程序设计

5.2.1 存在的问题

1. 数据采用 double 型，参考同使用 double 型的 Matlab，计算误差使用了 2.67e-9，这个设定是否合理有待商榷；
2. 为了提高运算速度，本程序并未使用 stl 作为矩阵的实现，而是写了一个矩阵类。由于时间仓促，并未写内存池用以应对频繁的内存申请释放，在应对大的计算域时，可能出现多种内存问题；
3. 为生成三角形而准备的数据结构有很大的改进空间，可以进一步缩小内存占用，减少内存访问，提高运算速度；
4. 由于时间仓促，程序只在较为核心的部分进行了异常处理，其他异常均抛出至主函数 catch；

5.2.2 可能的解决方案

本文中的程序由作者在较短时间内独立完成，受到时间及个人能力限制，许多问题都没有得到妥善的解决。下一步工作中，在时间充裕或有师兄指导的条件下，作者会对本程序进行进一步完善。