arXiv:2302.02795v1 [cs.MS] 3 Feb 2023

Trimpack
非结构化三角形网格生成库

Juan M. Tizón [1], Nicolás Becerra[2], Daniel Bercebal Claus-Peter , Grabowsky [3]

Departamento de Mecánica de Fluidos y Propulsión Aeroespacial, Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio (ETSIAE), Universidad Politécnica de Madrid (UPM), Pza. del Cardenal Cisneros 3, 28040 Madrid, Spain

摘要

Trimpack 是一个用 *Fortran* 编写的例程库，可用于在任意区域内生成非结构三角网格，并采用用户定义的尺寸分布。用户必须编写程序，将库中的单元当作数学工具来使用。首先，用户必须通过提供点定义边界来定义区域。该库在内部使用样条对边界进行网格划分，并采用用户提供的节点分布函数。该库提供多种网格划分方法，从根据点云简单地生成 *Delaunay* 网格，到也可生成 *Delaunay* 网格的增量式 *Steiner* 型算法，再到高效的推进前沿型算法。本报告对 *Trimpack* 首次编写期间网格生成领域的最新进展进行了文献综述；这一时期是此类算法发展的重要阶段。接下来，报告详细介绍 *MeshGen*。这是一款用 *C ++* 编写的程序，充分利用 *Trimpack* 库的功能生成非结构三角网格，并配有功能强大的图形界面。最后，报告还详细说明了 *Trimpack* 库的内容。该库依据 *GNU* 通用公共许可证开放，任何希望使用或改进它的人都可以获取。

[1]通讯作者。电子邮箱：jm.tizon@upm.es - ORCID 标识符： https://orcid.org/0000-0002-8687-6657[2]电子邮箱：becerrazun.n@gmail.com - ORCID 标识符： https://orcid.org/0000-0002-4929-2206[3]电子邮箱：clausg@gmx.com

目录

1 引言 **4**

2 网格生成过程 52.1 网格类型.5 2.1.1 结构化网格.6 2.1.2 非结构化.7 2.2 结构化网格生成.8 2.2.1 代数生成系统.8 2.2.2 偏微分方程数值解.9 2.3 非结构化网格生成.13 2.3.1 德劳内基础.13 2.3.2 Delaunay生成算法.17 2.3.3 推进前沿算法.24 2.3.4 施泰纳算法.31 2.3.5 其他生成算法.35 2.3.6 数据结构.38 2.3.7 网格自适应.39

3 MeshGen 2.0 **41**

4 Meshgen 结构 41

5 用户界面 42

5.1 文件菜单.42 5.2 输入文件：.mg 格式.43 5.3 选项按钮.43 5.4 AFM 页面.44 5.4.1 轮廓类型.44 5.4.2 间距配置.44 5.4.3 AFM 版本.45 5.4.4 边交换.45 5.5 Steiner 页面.46 5.5.1 轮廓类型.46 5.5.2 间距配置.47 5.5.3 平滑和最终边交叉按钮.47 5.5.4 Steiner 因子.47 5.5.5 边交换准则.48

6 间距函数 48

7 TRIMPACK 52

7.1 Trimpack 的内容.52 7.1.1 生成算法.52 7.1.1.1 **dlny** .52 7.1.1.2 **afm** .52 7.1.1.3 **steiner** .52 7.1.2 网格细化算法.52 7.1.2.1 **nodins** .52 7.1.2.2 **edswap**.52 7.1.2.3 **smooth** .52 7.1.3 工具.52 7.1.3.1 **cbrdr**.52 7.1.3.2 **connec** .53 7.1.3.3 **edcros** .53 7.1.3.4 **spline**.53

7.1.4 其他子程序.53 7.1.4.1 **stmsh** （本程序中未使用）：.53 7.1.4.2 **centre** .53 7.1.4.3 **fnode**.53 7.1.4.4 **snode**.53 7.2 描述.53 7.2.1 数据结构.53 7.2.2 生成算法.55 7.2.3 网格细化算法.60 7.2.3.1 Nodins：.60 7.2.3.2 Edswap：.62 7.2.3.3 Smooth：.63 7.2.4 工具.64 7.2.4.1 Brdr：.64 7.2.4.2 Connec：.65 7.2.4.3 Edcross：.66 7.2.4.4 Spline：.67 7.2.5 其他子程序.68 7.2.5.1 Stmsh：.68 7.2.5.2 Centre：.68 7.2.5.3 Fnode：.69 7.2.5.4 Snode：.69

8 参考文献 **69**

1 引言

在计算流体力学中，网格是对守恒方程进行离散的载体，并对所得结果有着至关重要的影响。因此，选择正确的生成方法是解决任何问题的关键。为了做出恰当的选择，首先要明确我们的需求，同时考虑到本项工作还应具有教学和培养方面的意义：

- **速度：** 生成算法中固有的搜索和识别过程应尽可能简单。
- **灵活性：** 算法应能很好地适应复杂几何形状。
- **具备适应能力：** 应具备根据解进行调整的能力。
- **简洁性：** 该算法必须易于任何用户掌握。
- **可扩展性：** 所选算法必须具备进一步开发的可能性。
- **集成性：** 还必须考虑将该算法后续集成到另一种数据结构中。

考虑到所有这些因素，并且事先已知不存在能以最佳方式囊括所有这些特性的算法，我们必须转而选择一种能在上述大多数方面提供合适结果的算法。

基于基本 Delaunay 三角剖分的非结构网格生成算法正是如此。它的主要优点是编程相对简单，这是因为节点的选择依据 Delaunay 判据（参见 Delaunay 三角剖分的性质 2.3.1）；此外，该算法能够为各种流动问题生成灵活的网格。另一个优点是，基本 Delaunay 三角剖分是开发许多其他算法的出发点，例如分治算法、空间推进算法或增量插入德劳内算法。此外，正如前文所述，Steiner算法还需要先对边界进行初始 Delaunay 三角剖分。它可能唯一的缺点是速度不够快，这是由于其依赖的搜索与控制例程开销较大。不过总的来说，该算法能够提供相当合适的结果。

另一种在非结构网格三角剖分方面具有诸多优势的算法是推进前沿法（AFM），最初由 Peraire 开发。因此，我们将在本研究中选择该算法作为第二种网格生成程序。它比 Delaunay算法更快，也更易于编程，因为它可以沿用基础 Delaunay算法的推进前沿程序结构。推进前沿法的优势在于其固有的自适应能力，也就是说，该算法便于根据解进行适配；此外，用户还可以通过背景参数直接控制网格，因此该算法极具竞争力。由于节点插入准则十分敏感，需要调整单个网格参数才能获得最优三角剖分。

2 网格生成过程

2.1 网格类型

基本上，在许多关注领域中，使用两种类型的网格：结构化网格 和非结构化网格。这两种情况下，计算域的外部轮廓都与边界壁面对齐，不过也可以采用非协调边界，即网格轮廓与计算域轮廓不重合。后一种方案使用笛卡尔网格，通过对均匀网格逐次加密，使网格适应计算域，其明显优势是大大简化了复杂计算域中的网格生成过程。然而，这种网格并不常见，因为要精确描述计算域边界层中的流动，会对所得解产生非常显著的影响。

网格生成过程可能对总模拟时间产生重大影响。在航空航天工程中，对飞机外部空气动力学进行数值模拟，可能需要数周时间来生成网格，具体取决于几何形状所需描述的细节。机身和机翼的几何形状可能相对简单，但发动机燃烧室的计算则存在相当复杂的问题。还有其他问题。

计算流体力学中使用的网格可分类如下：

- 结构化网格：这类网格是规则的离散化，网格线通常与计算域边界对齐（并与其垂直）。此类别还包括多块网格，其中每个网格块都是结构化的，但各块之间的连接关系并不规则。对于面向非结构网格编写的程序代码，数据管理会增加额外工作，但可将其概括处理为非结构网格。结构简单的区域或单个网格块内都很容易生成结构化网格。结构化网格可以高效地贴合计算域轮廓以及解的当前曲面，从而提高结果的准确性和可信度。最后，结构化网格占用的存储内存较少，计算时间也较短。其主要缺点在于难以对复杂区域进行离散化，这可能导致过渡区域的网格质量不佳；反之，若投入工作来改善网格质量，又可能使网格块结构变得过于复杂，从而无法发挥其使用优势。

- 非结构网格：非结构网格可以这样定义：如果无法建立一种简单的映射，将网格的每个拓扑组件（节点、边、面等）映射到与其连接的单元，那么该网格就是非结构网格。因此，非结构化网格通过包含这些信息的连接矩阵进行处理。非结构化网格最重要的两个特性是适应性和灵活性。一方面，这类网格可以控制区域中特定位置的网格尺寸（适应性），而不会影响其他无需细化的区域。相比之下，结构化网格中的局部细化会不必要地扩展到整个区域。另一方面，灵活性使网格划分几乎能够自动完成极其复杂区域的网格划分；也就是说，它可以模拟难以采用结构化方式离散的问题，或者将生成网格所需的时间缩短一到两个数量级。其缺点主要是精度较低，因为体的面通常不会与流线对齐，而且这类网格需要占用更多内存和计算时间。混合网格：结合结构化网格和非结构化网格的网格，旨在兼取两者的优点。当结构化部分用于边界层离散化时，这种策略尤为有用，因为网格的长宽比和对齐方式对于获得良好结果至关重要。

- 非传统网格：本节可以介绍具有非共形轮廓的笛卡尔网格，或几何基底为分布式点云、且没有明显相互连接关系的无网格网格。

网格的规则性和光滑度会直接影响流体问题以及一般非线性问题中数值计算结果的质量。因此，在二十世纪最后几十年里，随着数值算法和计算机性能开始足以求解实际问题，基于求解偏微分方程来生成网格的方法变得十分流行。例如，采用等势面可以确保网格线局部垂直于轮廓，从而充分描述流线和垂直梯度。如今，网格生成算法已应用于非常复杂的三维区域。基于微分方程的网格生成方法已不再流行，取而代之的是代数方法；这些方法应用于由源自图形设计和 CAD 系统的算法参数化的区域。如果所采用的空间离散不够规则

且不够光滑，数值算法的预期截断误差就会增大，所得结果也缺乏足够的精度。计算域的复杂性和三维特性意味着，如果不采取谨慎措施，网格的规则性和光滑度就难以达到足够的要求。必须始终检查网格质量，为此需要定义一系列指标，以表征网格质量并比较不同情况。通常，每个网格生成程序都有一套质量指标，这些指标基于各种几何参数之间的关系，但都反映网格在以下类别中的规则性：

- 单元细长率：为衡量各向异性，人们采用多种方法计算边长、对角线长度或厚度之间的关系。只有在边界层网格中，确保流线与.....方向基本平行时，细长单元才被认为是合适的。

- 单元偏斜度：构成面的角度与边的角度之间存在明显差异，以及与 $\pi/2$ （大于或小于）相差很大的角度，都会导致显著的计算误差。
- 相邻单元尺寸不均：由于区域复杂或流场形态复杂，不同区域的适当网格尺寸可能相差很大。尽管如此，过渡仍须平滑。应避免网格尺寸突变，或将其置于流体变量梯度较小的区域。否则，由于近似阶数降低，结果会迅速变差。

  最后需要指出，从操作角度来看，在网格质量较差的情况下，数值解难以收敛；需要更多迭代，计算时间也会延长；最终可能无法达到足够的收敛水平，甚至解根本无法收敛。计算网格的质量会直接影响所得解的精度。通常优先采用结构化网格，尽可能降低畸变，并使其主要方向与流动方向一致。

一致的数值分析应当在采用更精细的离散化时，得到精度不断提高的结果。从这个意义上说，误差与网格尺寸有关，随之而来的问题是：要达到某一置信水平，需要采用何种细化程度，而这一置信水平由误差估计来表征。如果能够采用先验误差估计方法，这项工作就能大大简化；然而，在计算流体力学中，尚未定义足够可靠的指示器。从实用角度来看，目前唯一可行的办法是通过逐次加密计算网格，对解进行收敛性研究。但是，由于所用区域的复杂性、网格细节以及所用数学模型的复杂性和非线性，要就所得解的质量得出确切结论并不容易。所有这些因素共同构成了复杂的局面，而不确定性来源数量众多、性质各异，因此开展严格的收敛性研究至关重要。进行网格尺寸收敛性研究的标准流程，是针对一个或多个案例研究采用一组不同分辨率的网格。可以定义解的点值或积分值，以比较各网格所得的结果。此时，预期使用的是经验证和确认的代码；若网格足够精细，便可借此获得真实的解。当然，我们需要思考何谓真实解或实数解；事实上，“真实数值解”这一术语应在渐近意义上使用。

2.1.1 结构化网格

在最广泛使用的方法中，区域被划分为由四边形单元构成的结构化组合。网格的结构性显而易见：每个内部节点周围恰好都有相同数量的网格单元（或单元）。请注意，在这种情况下，我们可以将曲线坐标系（ξ，η）与网格线关联起来，从而立即确定网格中的两个方向。如果我们沿着 η恒定的直线依次为节点编号，使得编号随 ξ 增大而增大，就能轻松找出网格中任意节点 J 的最近邻。通常，这类网格是通过将研究区域映射到正方形中，再在正方形上构造矩形网格来生成的。如果方程本身也进行了映射，就可以用该网格求解；否则，则应用逆映射，以便在原始区域上得到所需的网格。

可将多种方法视为实现映射的候选方案，例如保角技术、微分方程或代数方法。通常，流动方程的所有主要离散化过程都可以在此类网格上实现。采用结构化网格对计算流体动力学的一大优势在于，可以从大量可用算法中选择合适的解

法。这些算法的优点是通常能够以较高的计算效率实现。其缺点在于，如上所述，将映射方法应用于形状一般的区域时，无法保证得到合格的网格。通过将计算域恰当地划分为多个块，再分别对每个块应用映射方法生成网格，可以缓解这一难题。这是一种极为强大的方法 [57], ，但生成质量较差的单元，以及为形状极其复杂的区域生成网格所需的时间，仍可能带来问题。

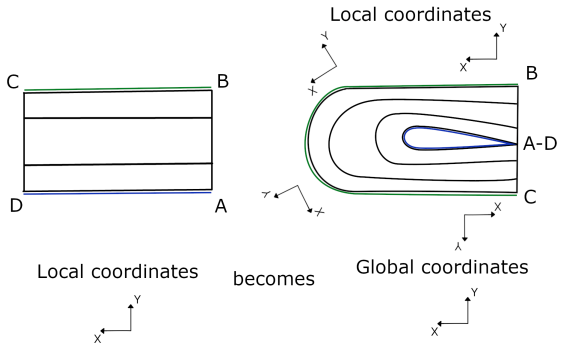


图 1：具有规则结构的网格变换

2.1.2 非结构化

另一种方法是将计算域划分为由计算单元组成的非结构化集合。非结构网格的显著特点是，网格中典型内部节点周围的单元数量不一定恒定。显然，在这种情况下也可以使用四边形单元，但本节将重点讨论三角形网格的使用。现在需要对节点和单元进行编号；为获取邻居所需的信息，我们可以存储属于每个单元的节点编号。显然，这类网格中不存在方向性的概念，因此，基于这一概念的求解技术（例如交替方向隐式方法）无法直接应用。通常用于生成非结构三角形网格的方法，基于 Delaunay [71] 或推进前沿 [64] 方法。基于积分过程的流体流动方程离散化方法，例如有限体积法或有限元法，自然适用于非结构网格。非结构化方法的主要优点是，它为离散化复杂形状的区域提供了非常强大的工具（[59], [66]），尤其是在二维中使用三角形、三维中使用四面体时。此外，非结构网格方法自然具备实现自适应性 [63]的可能。采用非结构网格方法的缺点是，目前可选的求解算法相当有限，而且其计算实现对计算机内存和中央处理器 [67]的要求都很高。此外，这些算法对所用网格的质量相当敏感，因此在生成过程中必须格外谨慎 [93]。

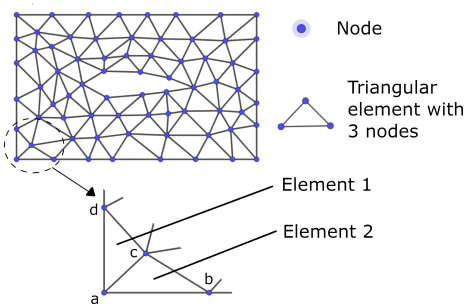


图 2：具有不规则结构的网格

2.2 结构化网格的生成

在本综述及其引用的参考文献中，数值生成的结构化网格被定义为由边界拟合曲线坐标系的坐标线交点构成的有序点集。这种坐标系的核心特征是，某条坐标线（三维情况下为曲面）与物理区域边界的每一段重合。这样，边界条件就可以完全沿坐标线表示，无需插值。利用坐标线的交点定义网格点，可形成一种组织结构；当相关偏微分方程经过变换，以曲线坐标取代笛卡尔坐标作为自变量后，所有计算都可以在固定的正方形网格上完成 [53]。

关于此类网格生成的基本思想、必要的变换关系，以及将其应用于偏微分方程数值解的过程，参考文献 [26] 以入门形式进行了汇总。参考文献 [19] 详细讨论了各种类型的边界贴合坐标系及其数值生成方法，并列举了许多将其应用于场问题的实例。不同类型的网格更适用于不同的物理问题、构形以及使用方式。要在各种坐标中进行选择，首先必须考虑特定问题所需的约束。一般区域的基本约束是其边界几何形状。当坐标与边界相匹配时，便无需进行边界插值，而且网格也会在边界附近与所需的解对齐。对于二维情形，若无其他要求，保角坐标系通常是最佳选择。然而，除边界几何形状外，沿边界的逐点分布通常也需要作为进一步的约束。该分布由一个或多个边界坐标系描述，与几何形状共同构成完整的边界表示。当边界表示由任意方式指定时，由于解析延拓的限制，保角变换并不适用。下一个较简单的情形是采用正交坐标。在二维中，正交坐标通常适用于平面和曲面。在三维区域中，正交坐标系受到诸多限制，通常并不适用。一般情况下，能做的最好处理是用正交坐标系围定此类区域，从而可在边界处指定完全正交性。还可以通过指定导数施加进一步的边界约束，以规定进入或离开区域的变化率。在任意维度上，都可以实现为拓扑复杂构形平滑拼接组合网格的能力。除了各种边界约束外，还可以在区域内部获得显著优势。其目的通常是在网格点数量固定的情况下，更充分地解析给定问题的数值解。还可以施加约束，使某种理想的网格结构平滑地嵌入区域内部，从而获得其他优势[13]。

结构化网格生成系统是用于生成定义网格的曲线坐标系的一套程序。这些系统主要分为两类：代数系统，其中坐标由插值确定；以及偏微分方程系统，其中坐标是这些方程的解。

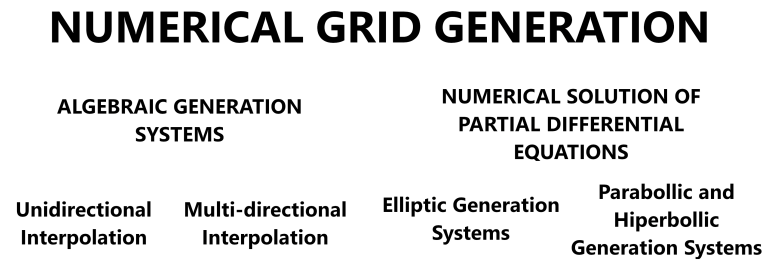


图 3：结构化网格生成的类别

2.2.1 代数生成系统

曲线坐标系的生成问题，可以表述为：根据边界上给定的值，生成变换区域内部的笛卡尔坐标值。当然，这可以直接通过对边界进行插值来实现，这类坐标生成方法称为代数生成系统 [27]。在许多情况下，代数生成是速度最快的方法，

其应用综述见参考文献（[19],[23],[24]）。代数方法还可以明确控制网格点分布。一些代数网格生成系统会将边界斜率不连续性传播至场内，而且本身不具备平滑机制。参考文献 [19] 中遇到了这方面的一些问题。不过，在多曲面方法中使用局部插值，可以避免不连续性传播到场内 [16]。代数方法尤其适用于交互式图形，因为它可以快速生成网格 [53]。

单向插值 代数网格生成本质上是在场中的边界和/或中间曲面之间进行插值。简单的一维拉伸只涉及使用变换函数，通常应用于通过其他方式生成的坐标系 [19]。参考文献（[12],[34],[51],[35],[41],[45]）中给出了沿直线进行简单拉伸的示例。另一个简单应用是归一化两个边界之间的间距，这种方法已常用于近期的应用中（[29],[52],[38]）。

区域内可能需要设置中间曲面，以处理若干畸变区域；若仅在边界之间进行插值，生成的网格将不尽理想。参考文献 [18] 展示了一个示例：在修正与边界重叠的坐标系时，只使用了一个中间点。“双边界”技术（[20],[22]）和“外曲面”方法（[23]）是仅利用相对边界曲面进行一维埃尔米特插值的示例。

多曲面方法（[19],[23]）是一种相关的单向插值过程。该过程通过对指定向量场进行插值，然后在每个插值点进行向量归一化来构造，以实现所需的渐缩塌缩，从而使边界相匹配。在参考文献 [15] 中，多曲面方法采用分段线性插值函数生成嵌入式网格，并确保坐标线斜率的连续性。参考文献 [16] 将这一方法扩展为使用高阶局部插值函数，从而也实现曲率连续性。文献 [14] 描述了一组能够使用多曲面过程自动完成网格构造中必要部分的子程序。多曲面方法已在参考文献（[30], [39], [48]）中应用于跨声速流动解。

多向插值 各种插值方法的主要区别在于所使用的曲线或曲面有多少、是什么类型，以及是否在这些曲面上指定导数；其次还取决于所使用的混合函数类型。参考文献（[19],[23],[18]）讨论了几种方法。插值的精度阶可以通过增加更多曲线或曲面，或增加更多信息（例如在曲线或曲面上指定高阶导数）来提高。

超限插值 [18] 是曲面曲线插值方法之一。超限插值最初是为雕塑曲面和固体的计算机辅助设计而开发的，它是在沿曲线和曲面定义的函数之间进行插值，而不是在点值之间插值，因此能够在不可数多个点上匹配函数。（超限插值之所以具有不可数性，是因为一般边界可能由无穷多个点定义；相比之下，张量积结构，即投影算子的乘积，只使用由离散值集定义的角点信息，例如分段线性函数。在更高维度中，这种插值可表述为一系列单变量插值，即投影，并将它们组合成布尔和投影。（布尔和是定义超限插值的主要机制。多方向结果是单向插值的布尔和。）函数规定曲线或曲面上的变量值（也可能包括某些导数）。这些曲线或曲面之间内部的值则通过插值确定，所用的插值函数通常称为混合函数。混合函数通常是多项式，但也可以使用其他函数。参考文献 [50]中使用超限插值生成与拐角附近解析生成网格相连的网格。参考文献 [42]中介绍了其在二维翼型上的另一种应用。参考文献（[17], [47], [37]）中使用超限插值进行三维网格生成。

2.2.2 偏微分方程的数值解

边界贴合坐标系的生成，是通过根据区域边界上给定的值（和/或与边界相交的坐标线的斜率），确定物理区域内部的曲线坐标值来完成的（见图4）。

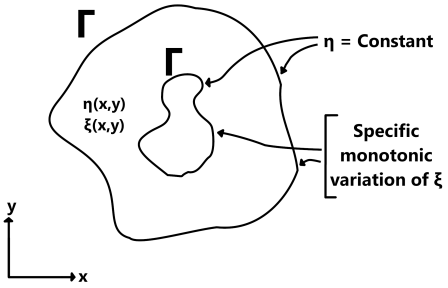


图4：变换方案示例

物理边界曲线（在三维中为曲面）的每个线段上，一个坐标保持为常数，而另一个坐标沿该线段单调变化 =， [27]。

变换区域中的等效问题，是根据该区域边界上给定的值和/或斜率，确定变换区域内部物理坐标（笛卡尔坐标或其他坐标）的值（见图5）。

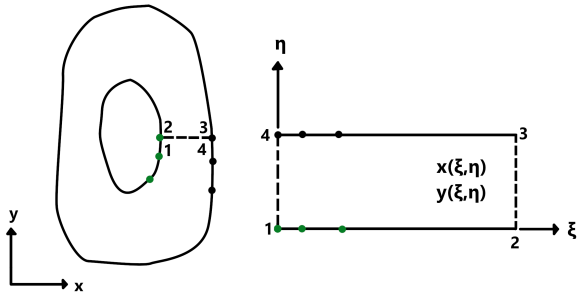


图5：变换方案示例

这个问题更便于计算，因为变换区域的边界由水平线段和垂直线段组成，因此该区域由彼此相邻的矩形块构成，至少在通过回入边界（分支切割）连接的意义上是如此 [27]。

根据边界值生成函数的场值可以通过多种方式实现，例如在边界之间进行插值等。然而，这类边值问题的解是偏微分方程中的经典问题，因此将坐标视为偏微分方程组的解是合乎逻辑的。如果在物理区域的整个闭合边界上指定坐标点（和/或斜率），则方程必须是椭圆型；而如果只在边界的一部分上给出规范，方程则应为抛物型或双曲型。例如，指定物理区域的内边界，而周围的外边界任意确定时，就会出现后一种情况。

椭圆型生成系统某些椭圆系统所体现的极值原理，即解的极值不能出现在场内，可用于确保物理区域与变换区域之间的一一映射（[4], [3]）。这是因为，沿物理边界段的曲线坐标变化必须是单调的，并且在相对的边界段上具有相同的取值范围 [27], 。显然，由此可知，物理区域内部不允许出现曲线坐标的极值，否则坐标系就会发生重叠。请注意，相关的是物理空间中偏微分系统的极值原理，也就是说，将曲线坐标作为因变量的系统；因为在边界上必须保持恒定且单调的是曲线坐标，而非笛卡尔坐标。因此，重要的是物理空间中偏微分方程的形式，也就是包含对笛卡尔坐标求导的方程形式。

关于坐标系生成的另一个重要特性，是椭圆系统的解所具有的固有光滑性。此外，边界斜率不连续性不会传播到

场中。最后，椭圆算子的平滑趋势和极值原理，使得我们能够针对任何构型生成网格，且网格线不会重叠。

因此，采用椭圆偏微分方程组作为坐标系生成手段具有诸多优点。当然，其缺点是必须求解偏微分方程组，才能生成坐标系。

参考文献 [19] 追溯了用于网格生成的椭圆系统形式的发展历程。上述参考文献以及参考文献 [27] 涵盖了许多由椭圆偏微分方程生成的坐标系的生成与应用实例。最佳的一般选择似乎是基于某种微分几何考量的方法 [28]。可以通过将这些方程中的某些度量系数替换为指定函数来进行线性化处理 [19], ，但这种方法很可能会导致形状和配置更为一般的畸变网格。目前使用最广泛的椭圆型生成系统是基于泊松型方程组的系统。正如参考文献 [19] 中所述，该系统已在许多工作中得到应用，参考文献 [25] 也对此进行了讨论。其他应用见参考文献（[40], [44], [31], [33], [49], [7]）。

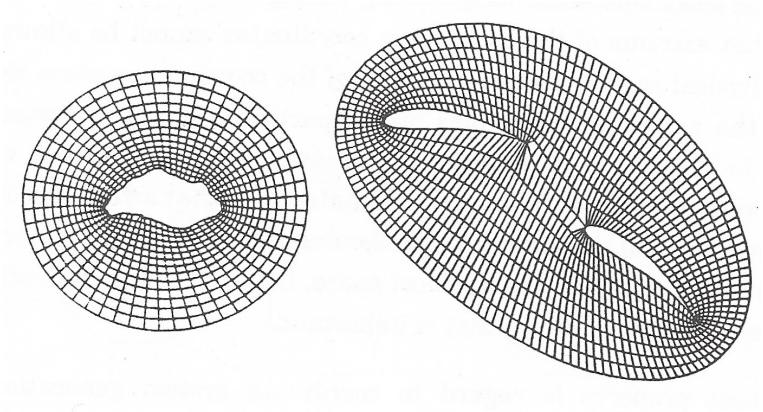


图 6：椭圆网格

抛物型和双曲型生成系统 网格生成系统也可以基于双曲型或抛物型偏微分方程，而非椭圆方程。在这两种情况下，网格都是通过数值求解偏微分方程生成的，并沿三维空间中两个边界曲面之间的某一曲线坐标方向推进。这两种方法都无法指定一般区域的全部边界——只有椭圆方程才能做到这一点 [27]。

对于双连通区域，如果已指定两个边界，则可应用抛物型系统生成这两个边界之间的网格。然而，双曲型情形只允许指定一个边界，因此仅适用于在物理无界区域中进行计算，此时计算外边界的精确位置并不重要。抛物型和双曲型网格生成系统的优点是通常比椭圆型生成系统更快，但正如前面所述，它们仅适用于某些特定构型。双曲型生成系统可用于生成正交网格。

超文本 Markup Language (HTML) **bolic 网格生成** 下面是使用此方法生成网格的示例（图 7 ）:

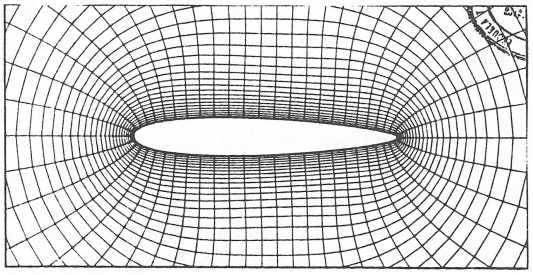


图 7：双曲型网格

对单元体积的规定可防止坐标系发生重叠，即使是在凹边界上方也是如此。在这种情况下，为防止单元体积趋于零，网格线间距会从边界向外迅速增大，如下图 8 所示。

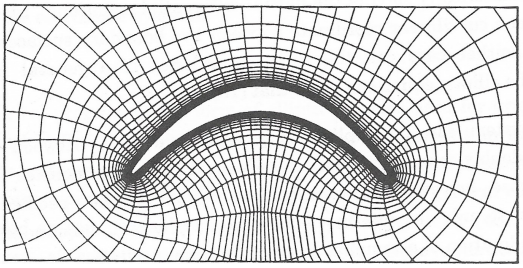


图 8：双曲型网格

虽然这可以防止重叠，但在某些情况下，随之产生的快速扩张可能导致截断误差问题。这种方法可以扩展到三维，此时从边界延伸出的坐标线与另外两组坐标线正交，但后两组坐标线彼此并不正交。显然，无论是双曲型还是椭圆型，都没有一种系统能在三维中实现完全正交。

这种双曲型网格生成系统的通用性比椭圆型生成系统低一到两个数量级，而其所需的计算时间大致相当于求解椭圆型系统时的一次迭代。指定单元体积分布可以避免在采用从边界向外投影的方法时，凹边界可能导致的网格线重叠。不过，涉及凹边界时，网格可能会有些扭曲。涉及凹边界时，网格可能会有些扭曲。涉及凹边界时，网格可能会有些扭曲。单元体积规范还可以控制网格线间距，但凹边界仍可能导致预期的间距出现在错误的坐标方向上，如图下部所示，因为受到控制的只是体积，而不是两个独立坐标方向上的间距。如前所述，构造出的网格是正交的。

然而，双曲型生成系统不如椭圆系统通用，因为无法指定区域的整个边界。如上所述，边界斜率不连续性会传播到场中，因此，沿着从边界斜率不连续点延伸出的坐标线，度量元素会出现不连续。最后，由于双曲型偏微分方程在某些情况下可能出现激波型解，因此，在某些边界点和单元体积分布规范下，可能会生成非常不理想的网格。在某些边界点和单元体积分布规范下会出现这一结果。在某些边界点和单元体积分布规范下会出现这一结果。与此相反，椭圆型生成系统往往会强调光滑性，这是由椭圆偏微分方程的性质决定的。在参考文献（[8], [32]）中，采用了一种双曲型生成系统，其中可以指定雅可比行列式，即单元面积。生成的坐标系在边界处正交，在其他位置也近似正交。参考文献 [19] 也讨论了这种生成系统。

抛物型网格生成抛物型网格生成系统可以通过修改椭圆型生成系统构造，使一个坐标方向上的二阶导数不再出现。这样，解就可以像上文所述的双曲型系统那样，从某个边界向外逐步推进。不过，在此过程中，方程仍保留了朝向推进方向的另一边界所产生的一些影响。

在参考文献 [21] 中，这种抛物型生成系统的构造基本上是：首先用二阶中心差分表示椭圆型生成系统中的所有导数，然后将一个坐标方向上前向线的所有值替换为按某种方式根据前几条线上的值以及外边界上给定的值所指定的值。这样，差分方程就化为一组 2×2块三对角方程，沿着各坐标线依次求解，并从指定边界向外推进。借助与椭圆型系统有某种类比关系的控制函数，可以控制坐标线间距。通过对前向线上的值进行函数形式的指定，可以使网格接近正交。

抛物型生成系统的速度也比椭圆型生成系统快，其程度与双曲型系统相同，因为同样只需依次求解三对角方程。对前向线上的值进行函数形式的指定，并使其受到外边界的影响，会引入来自第二个边界的平滑效应，而双曲型系统中不存在这种效应。不过，与双曲型系统相比，抛物型生成系统无法如此直接地实现正交性；此外，前向线上的值的指定形式以及控制函数的形式尚未得到充分发展。

Ref [11] 采用抛物型系统，即在通常的椭圆型系统中加入时间导数，以获得网格计算方程，并使其与流动解方程耦合，用于处理随时间变化的自由液面。

最后需要指出的是，过去二十年来，人们已经完善了这些不同的结构化网格生成系统；正如我们所见，这些系统已发展得相当成熟。因此，在实际流动问题中，通常采用结构化网格生成方法。

2.3 非结构化网格生成

首先，概述非结构化网格生成相较于结构化网格生成的主要优缺点，以便了解这种生成系统的适用性。

非结构化网格的优点包括：

- **自适应性**。非结构化网格生成具有固有优势，即能够根据流动问题的解调整网格。通过网格自适应，可以提高这些解的精度：将点移至关注区域，并从不重要的面积中移除。这意味着，在点数相同的情况下，使用非结构化网格生成的网格比结构化网格能获得更精确的解。

- **灵活性**。非结构化网格生成的一大优势，是能够在复杂且困难的几何形状中良好地适配网格。从这一点来看，非结构网格完全胜过结构化网格。这一优点是其诞生的根本原因之一。

其主要缺点如下：

- **生成过程复杂：**。复杂且耗时的搜索与识别算法、特殊的数据结构，以及对计算机时间的大量需求，使非结构化网格生成并非一项简单的任务。

- **求解过程复杂**。如前所述，求解算法仅限于积分过程（有限元法或有限体积法）。缺少相互关联的矩阵结构，使这些过程变得复杂得多。此外，对计算时间和内存的需求也极高。

现在将介绍本应用程序中使用的不同非结构化网格生成算法。生成非结构化网格有三种基本方法：**Delaunay**、**高级前沿法（AFM）**和**Steiner**。

2.3.1 德劳内基础

Delaunay 三角剖分，也称为“德洛内”三角剖分，是由相连的凸三角形构成的网格，并满足 Delaunay 条件。该条件规定，每个三角形的外接圆内都不得包含网格中的任何点（9）。

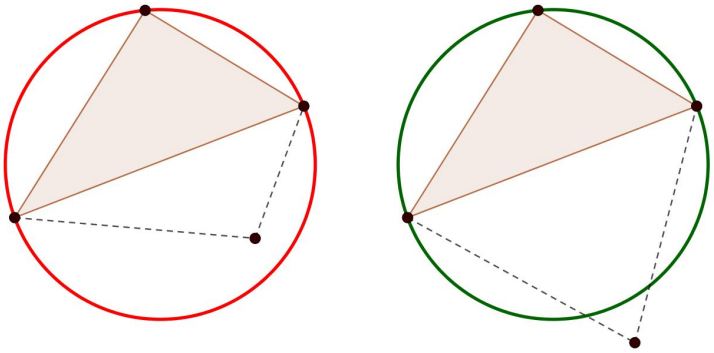


图 9：Delaunay 条件。绿色外接圆符合该条件，红色外接圆则不符合。

谈到 Delaunay，就不能不提 Voronoi 图（或 Dirichlet 镶嵌）。这是因为，对于给定的一组点，Voronoi 图是 Delaunay 三角剖分的对偶图。Delaunay 三角形的外心对应于 Voronoi 图的顶点（10、11）。

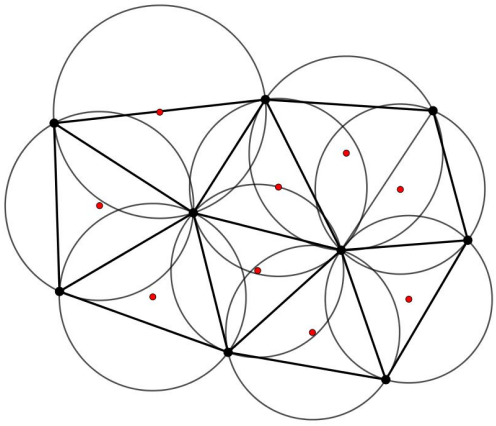


图 10：Delaunay 图外接圆

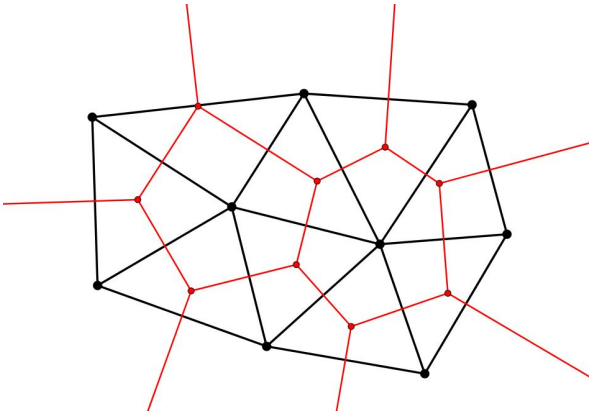


图 11：Voronoi 图

Delaunay 三角剖分的性质

1. **唯一性**。Delaunay 三角剖分是唯一的。此结论假设不存在四个共圆的节点。其唯一性源自 Voronoi 图的唯一性。

2. **满足 Delaunay 判据**，如图 12 所示。代数形式可写为：若点 D 位于外接圆 ΔABC 内部，则等于：

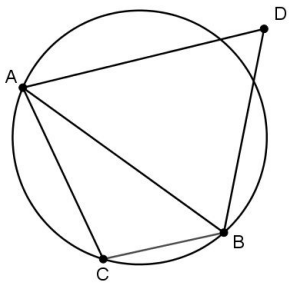


图 12： ΔABC 和 D 的内切圆测试（Delaunay 条件成立）

$$\langle ABC + \langle CDA = \begin{cases} < 180^\circ & \text{Delaunay condition true} \\ = 180^\circ & \text{A,B,C,D cocircular} \\ > 180^\circ & \text{Delaunay condition false} \end{cases} \tag{1}$$

3. **边圆性质**。 节点的三角剖分是 Delaunay 三角剖分，当且仅当存在某个圆经过每一条无点边的两个端点。这个表述非常有用，因为它还提供了一种定义约束 Delaunay 三角剖分的方法，其中某些边是预先指定的。节点的三角剖分是约束 Delaunay 三角剖分，当且仅当网格中的每一条边都存在某个圆经过其两个端点，且该圆内不含三角剖分中任何其他可见于该边的节点。在图 13 中，由于约束边 a-b，节点 d 对线段 a-c 不可见。

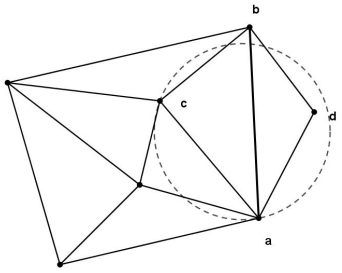


图 13：受　　约束的 Delaunay 三角剖分。由于线段 a-b 的约束，节点 d 对于 a-c 不可见

4. **等角性质**。Delaunay 三角剖分使三角剖分中的最小角最大化。因此，Delaunay 三角剖分被称为最大最小三角剖分。对于构成凸四边形的所有相邻三角形对，这一性质在局部也成立。这是 Lawson 局部边交换算法的基础。

5. **最近邻性质**。连接一个顶点与其最近邻所形成的边，是 Delaunay 三角剖分的一条边。这一性质使 Delaunay 三角剖分成为解决最近邻问题的有力工具。请注意，最近邻关系并不能描述 Delaunay 三角剖分中的所有边。

6. **最小粗糙度**。对于任意散乱数据集，Delaunay 三角剖分是一种具有最小粗糙度的三角剖分， [6], [74]。给定任意数据 f_i （网格的所有顶点）以及这些点的一个三角剖分，可以构造出唯一的分片线性插值曲面。Delaunay 三角剖分具有这样的性质：在所有三角剖分中，它使该曲面的粗糙度最小，粗糙度由以下 Sobolev 半范数衡量：

$$\int_T [(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2]\, dx\, dt \tag{2}$$

这是一个有趣的结果，因为它不取决于数据的实际形式。这也表明，Delaunay 三角剖分能够很好地逼近使该 Sobolev 范数最小化的函数。例如，满足拉普拉斯方程及适当边界条件的调和函数恰好使该范数最小化。已证明，Delaunay 三角剖分能够保证离散拉普拉斯近似（采用线性单元）满足最大值原理。

基本 Delaunay 三角剖分的生成 需要指出的是，大多数 Delaunay 三角剖分算法都以这样一个前提为起点：用于三角剖分的节点集合（节点云）已预先指定。

其思路是从一条已知边界边开始，将它的两个端点分别与某个内部节点相连，形成一个新的三角形。只要新生成的边尚未属于其他三角形，这样最多可以再生成两条边。所有边界边都纳入三角形后，新边就会形成一条边界。这条不断移动的边界通常称为推进前沿（不要与 Peraire 的推进前沿法混淆，后者不会生成 Delaunay 三角剖分）。这可以利用

空圆性质，即 Delaunay 三角形的外接圆内不包含其他点。这样便可以如图14所示，通过迭代选取合适的点。

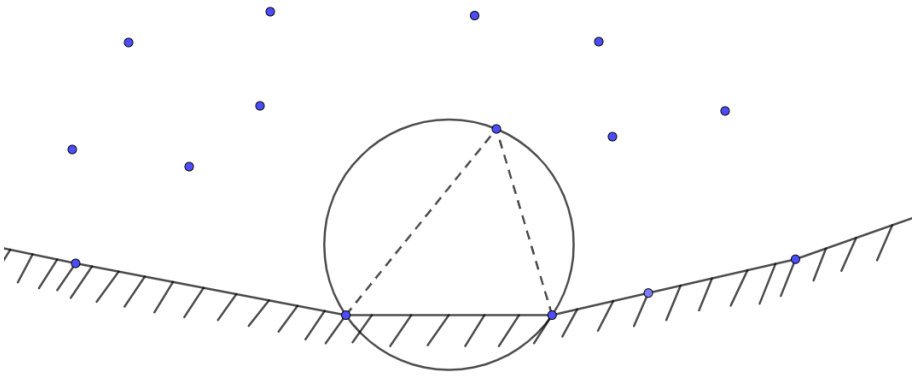


图14：一个直接的迭代过程会选取一个节点，使其与给定边构成的外接圆最小。外接圆内没有节点，即表明算法收敛。

迭代从选取一个位于给定边所需一侧的任意节点开始。如果没有这样的节点，则给定边是凸包的一部分。接着，构造一个经过该边两个端点和所选节点的外接圆。最后，检查圆内是否有节点。如果有，则将所选节点替换为离外心最近的节点，并重复此过程。当外接圆内没有节点时，将边的两个端点与所选节点连接起来。

这个看似简单的算法实际上是一项非常繁重的过程，因为它需要进行大量搜索并运行控制例程，而且这些操作要扩展到所有节点。这个三角剖分算法由 Tanemura、Ogawa 和 Ogita 于 1983年 提出[43]，后来又由 Merriam 于 1991年 重新发现[85]。另见参考文献 [89]

重新网格划分技术 为了便于理解后文介绍的一些算法，我们将介绍几种最常用的重新网格划分技术。许多生成算法会在三角剖分过程中或之后采用这些技术，以改善网格质量。此处我们只介绍其基本操作原理，稍后再说明它们在不同算法中的应用。

边交换 对角线交换会改变网格中节点之间的连接关系，而不改变节点的位置。此过程需要遍历所有单元边，但边界上的边除外。对于三角形 ABC 和 ADB 的公共边 AB（图 15），我们考虑是否可用 CI 替换 AB，从而将三角形 ABC 和 ADB 替换为三角形 ADC 和 BCD。如果新的配置比原有配置更符合规定的正则性准则，则执行交换。在我们的实现中，只有当新配置中的最小角大于原配置中的最小角时，才会执行边交换操作。边交换技术仅适用于凸四边形 [93]。

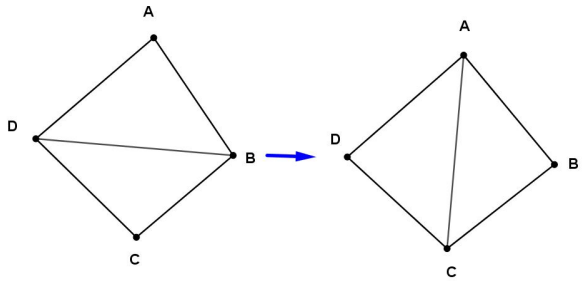


图 15：边交换

节点插入 正如我们将看到的，节点插入是生成算法用于网格优化的最常见技术之一。我们甚至可以根据不同生成算法的节点插入过程对它们进行分类

根据其节点插入流程。

预先插入：节点是预先指定的。也就是说，可以通过节点云（可能是随机分布的节点= ）或由其他网格固定的节点来指定节点（例如，基本德劳内算法、边交换算法等）。

生成过程中插入 ： 初始三角剖分域中的节点是自由的，网格按节点逐个生成，遵循确 定的判据。（例如，推进法等。）

后置插入：在已生成的网格中添加附加节点，以改善网格质量或提高解的精度（例如，采用不同的Delaunay算法、Steiner算法等）

请注意，所提及的许多算法都结合了这些插入过程。

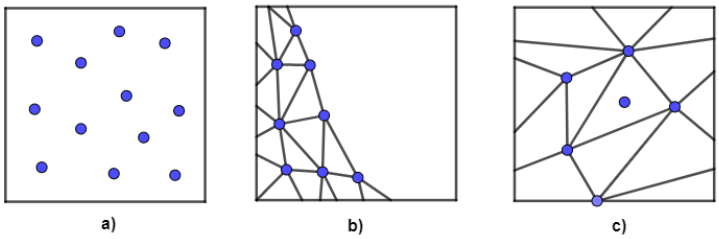


图16：节点插入（a）之前、（b）期间和（c）之后

网格光顺 网格平滑会改变内部节点的位置，但不会改变网格的拓扑。单元边被视为弹簧，其刚度与边的长度成正比。节点不断移动，直到弹簧系统达到平衡。通过迭代求得平衡位置。每次迭代都要遍历内部点，并将其坐标移动到与相邻点的质心重合的位置。通常执行三到五次迭代。

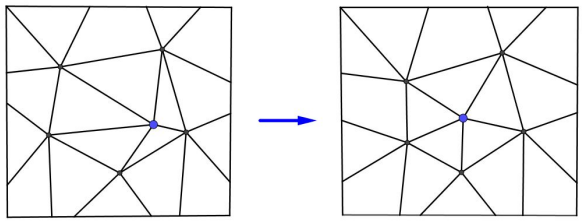


图17：平滑

组合的ap 事实证明，应用这三种后处理算法可以非常有效地改善smooth度和 生成网格的规则性。

2.3.2 Delaunay网格生成算法

现在我们来讨论几种用于二维 Delaunay 三角剖分的技术。选择这些方法是因为它们在截然不同的情形下都能达到最佳性能。Delaunay算法的讨论安排如下：

1. 全局边交换（Lawson） 2. 分治
算法 3. 空间推进法 4. 增量插入算法
(a) Bowyer算法 (b) Watson算法
(c) Green和Sibson算法

通过边交换进行 Delaunay 三角剖分 该算法由 Lawson 于1977年提出， [93] 假设已有一个三角剖分（但它不是 Delaunay 三角剖分），然后通过边交换使其成为 Delaunay 三角剖分。三角剖分 T 的等角性 A(T) 定义为角的排序 A(T) $= [\alpha_1, \alpha_2, \alpha_3, ...\alpha_{3n(c)_3}]$ 使得$\alpha_i \leq \alpha_j$ 如果 $i < j$。我们记作 A(T*) $<$ A(T)，当 $\alpha_j^* \leq \alpha_j$ 且 $\alpha_i^* = \alpha_i$ 时，适用于 $1 \leq i < j$。如果对于点集的所有三角剖分 T*，都有 A(T*) $\leq$ A(T)，则三角剖分 T 是全局等角的。Lawson 算法会检查网格的所有内部边。每条内部边都是由两个相邻三角形的并集形成的四边形的对角线。一般来说，首先必须检查该四边形是否为凸四边形，以确保交换对角线时不会产生边交叉。如果四边形是凸的，就选择能够优化局部准则（此处为局部等角性）的对角线位置。这相当于使两个相邻三角形中的最小角最大化。Lawson 算法持续执行，直到网格在局部得到优化，并且处处满足局部等角性（见图18）。容易证明，局部等角性条件等价于满足前文所述的内切圆测试（见 Delaunay 三角剖分的性质）。因此，处处满足局部等角性的网格就是 Delaunay 三角剖分。请注意，每次新的边交换（得到三角剖分 T*）都会确保全局等角性提高，即 A(T*)¿A(T)。由于三角剖分的维度有限，这保证了该过程会在有限步内终止。

迭代算法：通过 Lawson 算法进行三角剖分

- 交换边 = 真
- 当（交换边）时执行– 交

换边=假– 对（所有内部边）

执行

  - 如果（相邻三角形构成凸四边形），则
  - 交换对角线以形成 T*
    - 如果（满足优化准则），则
    - T=T*
    - 交换边 = 为真
    - 结束条件判断
  - 结束条件

判断– 结束循环

- 结束循环

使用 Lawson 算法构建 Delaunay 三角剖分时，无需进行四边形凸性测试。可以证明，由三角形对形成的非凸四边形从不会违反外接圆测试。当使用更一般的优化准则时（稍后将讨论），则必须进行凸性检查。

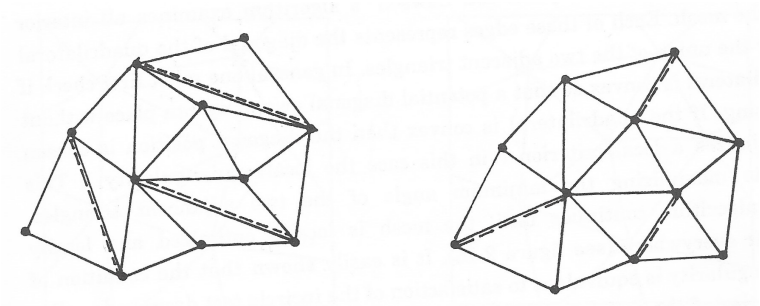


图 18：通过边交换进行 Delaunay 三角剖分。

分治算法 在此算法中，假定节点已预先指定。其基本思想是将点云 T（沿某个适当的轴排序）划分为左侧（L）和右侧（R）半平面。然后分别对每个半平面递归地进行 Delaunay 三角剖分。接着必须将这两部分合并，形成一个完整的 Delaunay 三角剖分。请注意，在以下讨论中，我们假定点已沿 x 轴排序（这可以通过 O(N log N) 的复杂度完成，其中 N 为节点数）。

算法：通过分治法进行 Delaunay 三角剖分。

1. 将 T 划分为大小近乎相等的两个子集 T_L 和 T_R。

2. 将 T_L 和 T_R 进行 Delaunay 三角剖分，合并为一个 Delaunay 三角剖分。

3. 将 T_L 和 T_R 合并为一个 Delaunay 三角剖分。

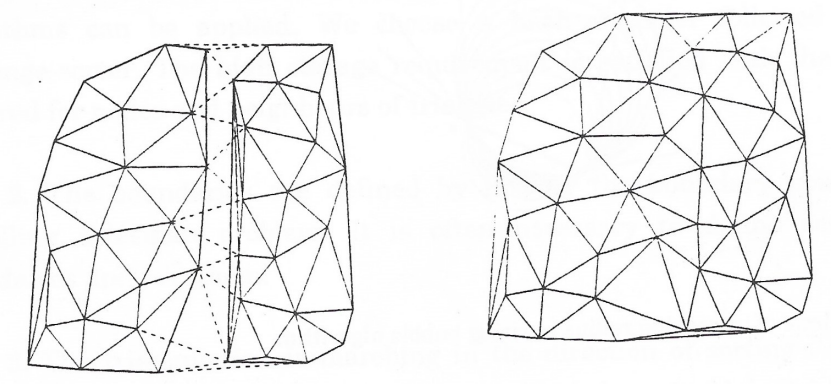


图19：分治法。

分治算法中唯一困难的步骤是合并左右两侧的三角剖分。注意到合并过程的两个性质后，这一过程便简化了：

- 合并过程中只会创建跨边（L-R 或 R-L）。由于合并过程中既不添加也不删除顶点，因此，若需要新建 R-R 边或 L-L 边，就表明原来的右侧或左侧三角剖分存在缺陷。（请注意，合并过程需要删除 L-L 和/或 R-R 边。）

- 左侧和右侧三角剖分中 y 值最小（最大）的顶点总是以跨边相连。鉴于 Delaunay 三角剖分会生成点云的凸包，这一点显而易见。

根据这些性质，我们现在概述“上升气泡” [82] 合并算法。该算法按 y 值升序生成跨边。算法首先连接左侧和右侧三角剖分中 y 值最小的顶点，形成一条跨边（性质 2）。这便构成了上升气泡算法的初始跨边。更一般地，考虑这样一种情况：A 和 B 之间有一条跨边，其端点位于一条穿过 A-B 的直线所形成的半平面上方，见图20。

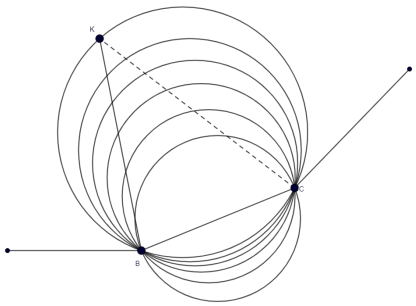


图20：上升气泡算法中半径不断增大的圆。

此图描绘了一个半径不断增大的圆连续变换，并经过点 A 和点 B。最终，圆不断扩大，并从左侧或右侧三角剖分中遇到一个点 C（在本例中，点 C 位于左侧三角剖分中）。随后，将该点与另一半三角剖分中 A-B 上的一个顶点相连，形成一条新的跨边（图20中的虚线）。有了这条新的跨边后，便可重复这一过程；当添加跨边期间或之后可以形成 L-L 边或 R-R 边时，过程即告终止。正确实现后，合并可在线性时间 O(N) 内完成。令 T(N) 表示总运行时间，则步骤 2 大约需要 2T(N/2)。因此，总运行时间由递推关系$T(N) = 2T(N/2) + O(N) = O(NlogN)$描述。

空间推进法 空间推进法由 Vilsmeier/H¨anel（1991年）详细阐述，主要基于基本 Delaunay 三角剖分，并提供了前者的增强版本。

上述方法的缺点是搜索范围较大，导致计算时间较长。该时间与 N^2 成正比，其中 N 为点的数量。因此，对于大量点，此方法并不适用。为避免大范围搜索，可以对该方法进行修改：先对所有点排序，再沿空间推进方向进行三角剖分。

算法： 空间推进法

1. 对所有点（包括边界节点）沿笛卡尔方向进行排序。选择较宽的方向会更合适。根据问题的不同，可以采用不同的排序算法。我们选择先使用哈希预排序器，再使用简单的交换排序器。较高的存储需求可由随后存储三角形节点及其邻居所需的存储空间满足。

2. 通过连接边界点来确定边界，这相当于求解旅行商问题。通常还需要在边界上定义若干部分并将其连接起来。

3. 三角剖分沿排序方向推进。设 N1 为尚未越过推进前沿的点的最小地址，N2 为大于 N1 的一个限定地址。需要寻找新三角形的前沿边，是指其两个端点的地址均小于或等于 N2 的边。作为生成新三角形的候选点，待检查的点范围限定为地址介于 N1 和 N2 之间的点。用于寻找合适点的判据与基本 Delaunay 版本相同。

4. 应对新三角形进行一些检查，尤其要避免新三角形的边与前沿中的其他边相交。这种情况可能发生在内部边界上，也可能出现在德劳内准则不唯一的情况下（同一圆上有三个以上的点）。如果出现问题，则跳过该新三角形。不过，要避免相交，只需检查所有至少有一个端点地址小于或等于 N2 的前沿边。

5. 将新边插入前沿边列表。两条相接的边从前沿列表中删除，同时调整所形成三角形的邻域。

6. 持续执行该过程，直至前沿列表为空；同时根据需要依次调整地址 N1 和 N2。

上述方法的输出是一个闭合三角剖分，其中所有节点及三角形的邻居均已确定，但由于形成新三角形时对点的选择受限，因此不一定满足德劳内准则（图 21）。因此需要进行重新网格划分（参见重新网格划分技术）。

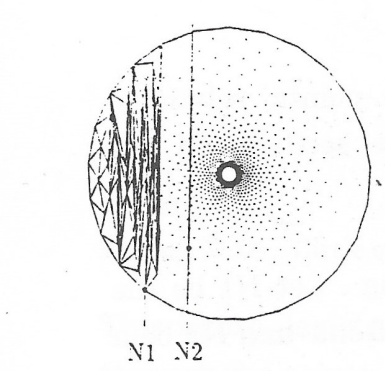


图 21：空间推进算法

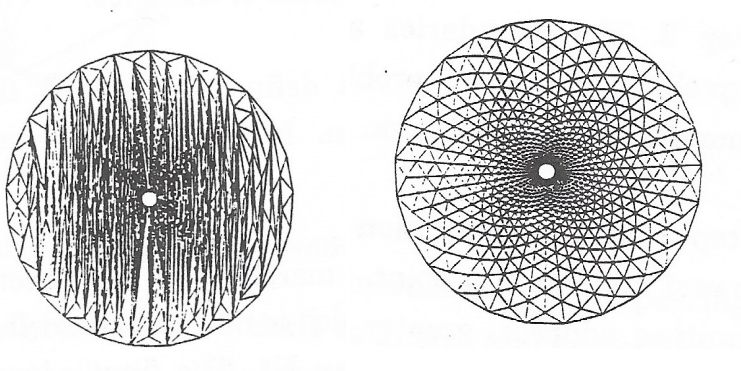


图 22：最终网格

计算工作的大部分，即使采用空间推进法，也由三角剖分造成。持续搜索的工作量与差值 N2-N1 成正比。假设三角剖分前沿上的点数为 O($\sqrt{N}$) 量级，并取相同数量的尚未进行三角剖分的新点。此时，N2-N1 仍为 O($\sqrt{N}$) 量级，三角剖分所需时间则为 O($N\sqrt{N}$) 量级。减少新点的数量会降低 N2-N1 的值，并减少新点；只有在无法仅使用前沿上的点构建新三角形时，才这样做

前沿中地址小于或等于 N2 的边数很快也会达到 O(1) 量级，N2-N1 的值也会如此，因此三角剖分的复杂度趋近于 O(N)。

增量插入算法 为简单起见，假设待添加的节点位于现有三角剖分的边界多边形内。如果我们希望对一组新节点进行三角剖分，可以始终添加三个初始虚拟点，构成一个足以包围所有待插入点的三角形。此外，内部边界通常也足以包围所有待插入点。此外，在 Delaunay 三角剖分中，内部边界通常会暂时忽略。这就引出了三角剖分中的点定位任务。一些增量算法需要知道外接圆包含该节点的任意一个三角形。在这两种情况下，点定位都有两个极端情形。典型的网格自适应和细化算法会将待插入节点的具体单元作为网格自适应算法的一部分来确定，从而减轻点定位的负担。另一种极端情况是，对随机分布节点进行初始三角剖分时，通常需要先进的点定位搜索技术，才能实现渐近最优复杂度 O(N log N)。基于四叉树和分裂树数据结构的搜索算法在这种情况下效果非常好。另一种常用方法是基于“行走”算法的搜索技术，因为它们简单易用。当依次添加的点彼此接近时，这些方法效果非常好。其基本思路是从前一个插入点在网格中的位置开始，沿着边（或单元）逐步朝新添加点的大致方向移动。最坏情况下，每次节点插入都需要 O(N) 次行走，因此整体最坏情况复杂度为 O(N^2)。对于随机分布的点，平均每次插入点需要 O($N^{0.5}$) 次行走，整体复杂度为 O($N^{1.5}$)。对于许多依次添加的点趋于彼此接近的应用，行走次数大致为常数，因此这些简单算法具有很强的竞争力。采用上述任一技术后，我们就可以进行插入算法。

- **鲍耶算法**

  Bowyer 算法的基本思想是在现有 Voronoi 图中插入一个新节点（例如图 23 中的节点 Q），确定其区域，删除完全位于该区域内的边，然后添加新边并重新配置图中的现有边。以下是 Bowyer 算法的基本内容，主要依据 Bowyer 的原始表述（完整细节请参见参考文献 [9] ）。

  算法：增量 Delaunay 三角剖分，Bowyer [9]。

  1. 在 Voronoi 图中找到任意一个顶点，使其到新点的距离小于到其生成点的距离。该顶点将在新的 Voronoi 图中被删除。2. 执行树搜索，找到其余可删除的顶点；这些顶点到新点的距离小于到其生成点的距离。3. 找出与可删除顶点对应的生成点集合 P。4. 删除 Voronoi 图中可用集合内顶点对描述的边，前提是待删除边的两个生成点都包含在 P 中。5. 计算 Voronoi 图的新顶点，求出其生成点和相邻顶点，并更新沃罗诺伊数据结构。

  Bowyer 的论文中介绍了实现细节和建议使用的数据结构。

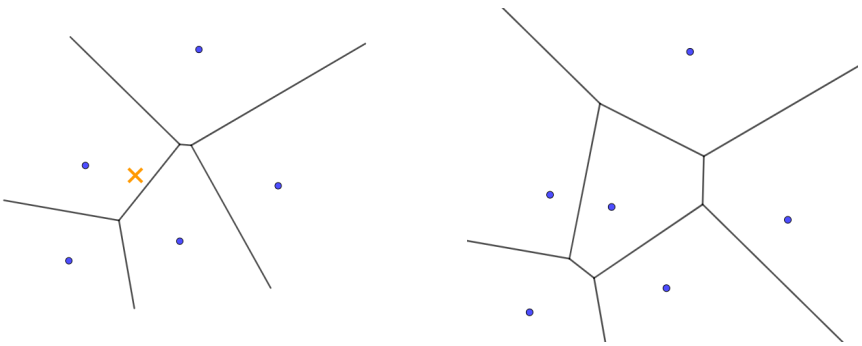


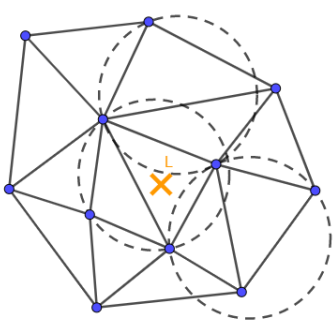
图 23：Bowyer 算法。

- **WATSON 算法**

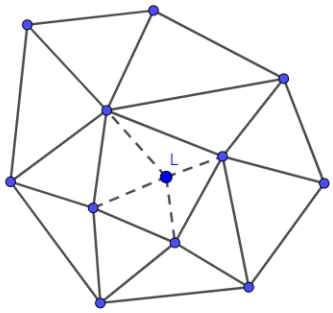
  Watson [10] 算法的实现相对直接。第一步是将一个新节点插入现有的 Delaunay 三角剖分中，并找到一个三角形（根），使新节点位于该三角形外接圆的内部。从根开始，通过树搜索查找所有外接圆包含新节点的三角形。具体做法是递归检查相邻三角形。由此得到的可删除三角形集合会显露出一个围绕节点 Q 的多边形空腔，该多边形的所有顶点对于节点 Q 都是可见的。随后，通过将多边形的顶点与节点 Q 相连，对空腔内部重新三角剖分（见图 24）。至此，算法完成。Baker [68] 对 Watson 算法作了详尽介绍，并讨论了与约束三角剖分相关的问题。

  算法：增量式 Delaunay 插入，Watson [10]

  1. 将新节点 Q 插入现有的 Delaunay 三角剖分中。2. 找到任意一个外接圆包含节点 Q 的三角形。3. 执行树搜索，找出其余外接圆包含节点 Q 的可删除三角形集合。4. 构建与可删除三角形相关联的边列表。删除列表中出现次数超过一次的所有边。5. 将剩余边连接到节点 Q，并更新 Delaunay 数据结构。



（a）Delaunay 三角剖分。 with node 添加在十字处。

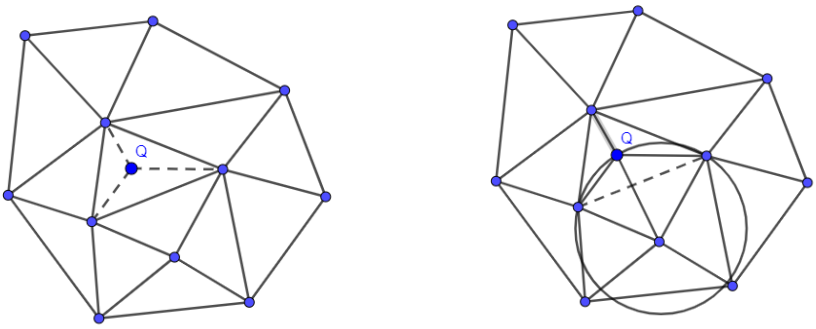


(b) 删除无效边后的三角剖分，并重新有效边和重新连接。

图 24：Watson 算法。

- **Green 和 Sibson 算法**

  Green 和 Sibson [67] 提出的算法与 Watson 算法非常相似。主要区别在于，该算法使用局部边交换来重新配置三角剖分。第一步是确定位置，即找到包含点 Q 的三角形。完成后，如图 25 所示，创建三条边，将 Q 与该三角形的顶点相连。如果该点落在一条边上，则删除该边，并创建四条边，将 Q 与新形成的四边形的顶点相连。根据外接圆判据可以证明，新创建的边（3 条或 4 条）会自动满足 Delaunay 条件。遗憾的是，某些原有边现在不再正确。我们需要设法找出所有可能无法通过圆测试的“可疑边”。如下文所述，这一点可以做到：将每条可疑边视为由两个相邻三角形构成的四边形的一条对角线。对该四边形的两个相邻三角形之一应用外接圆测试。如果四边形的第四个点位于外接圆内部，则交换这条可疑边，如图 25 (b) 所示；随后还会有另外两条边成为可疑边。我们可以随时立即识别所有可疑边。为此，首先考虑所有以 Q 为顶点的三角形构成的子集。可以始终保证，所有与 Q 相连的初始边都满足 Delaunay 条件，而通过交换新形成的任何与 Q 相连的边也必定满足 Delaunay 条件。因此，我们只需将该子集中其余构成 Q 周围多边形的边视为可疑边，并对其进行内切圆测试。当所有可疑边都通过外接圆测试时，该过程终止。



(a) 插入顶点。 (b) 交换可疑边。

图 25：Green 算法

算法：增量 Delaunay 三角剖分，Green 和 Sibson [2]

1. 定位包围点 Q 的现有单元。 2. 插入点，并将其连接到 3 个或 4 个周围顶点。 3. 确定新的可疑边。 4. 对所有未通过内切圆测试的可疑边执行边交换。 5. 确定新的可疑边。 6. 如果产生了新的可疑边，则转到步骤 3。

Green 和 Sibson 算法可以使用标准递归编程技术实现。该算法的核心是递归过程，对于图 26 所示的配置，该过程形式如下。

– 交换过程 $[v_q, v_1, v_2, v_3, edges]$– 如果 (incircle $[v_q, v_1, v_2, v_3] =$ TRUE)，则

- 调用重新配置边 $[v_q, v_1, v_2, v_3, edges]$
- 调用交换 $[v_q, v_1, v_4, v_2, edges]$
- 调用交换 $[v_q, v_1, v_5, v_3, edges]$–

结束 if– 结束过程

此示例说明了一个重要问题。Delaunay 三角剖分的性质保证，与 Q 相邻的交换边将成为 Delaunay 递归过程的最终边。在后面的章节中，我们将探讨如何基于其他交换准则，通过增量插入和边交叉来生成非 Delaunay 三角剖分。该算法也可以递归编程，但递归实现需要反向传播。对于 Delaunay三角剖分算法，插入算法可简化为以下三个步骤：

递归算法：增量 Delaunay 三角剖分，Greeen 和 Sibson

1. 定位包含点 Q 的现有单元。2. 插入点，并将其与周围顶点连接。
3. 对新形成的单元（3 个或 4 个）执行递归边交换

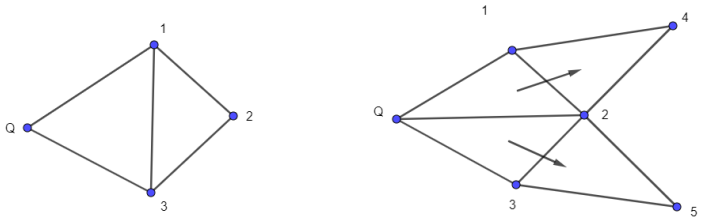


图26：通过向前传播进行边交换。

Delaunay算法的实现

- Baker（1990年），[71]：介绍了一种生成四面体网格的方法。该算法基于 Delaunay 三角剖分，可处理复杂程度基本不受限制的对象。
- Mitty（1991年）[86]：将三维非结构化网格上的自适应网格加密应用于精确解析斜激波。通过应用Bowye算法生成网格，从而得到空间的Delaunay镶嵌。
- Mavriplis（1991年），[84]：本文提出了一种生成并自适应细化高度拉伸的非结构网格的方法，该网格适用于计算任意二维几何体周围的高雷诺数黏性流。该方法基于对预先确定的一组点进行Delaunay 三角剖分。
- Taniguchi（1991年），[87]：针对具有复杂边界几何的三维物体进行基于Delaunay的网格生成。
- Meshkat（1991年），[88]：基于Delaunay四面体化的三维自动非结构网格生成。
- Rebay（1993年），[95]：通过Delaunay 三角剖分和Bowyer-Watson算法实现高效的非结构网格生成。

2.3.3 推进前沿算法

现在，我们将深入研究不同的推进前沿法（AFM）。这是计算流体力学（CFD）中构建三角形网格和四面体网格的第二种方法。在本节中，我们将把Huet的“前沿法”作为一种推进前沿算法加以介绍。推进前沿算法的讨论安排如下：

- 原始推进前沿法
- 推进前沿法变体
- ”Methode de front”
- 推进前沿算法的实现

原始推进前沿法 发展最完善、也很可能是应用最广泛的二维和三维网格生成算法，是 Peraire 提出的推进前沿法（AFM），该方法首次发表于 [35]。与基于 Delaunay 的网格生成方法不同，它依据一系列启发式原则来放置和连接顶点，从而生成平滑且规则的网格。此外，AFM 还能够生成单元在局部沿某一方向拉伸的网格；许多采用基于 Delaunay 概念的方法的研究者都难以实现这一特性 [41]。随后，AFM 成功扩展至三维四面体剖分 [13],[23]。

我们将首先介绍这一“原始”版本，稍后再讨论各种改进之处

背景网格 对于下文将要介绍的方法，首先需要手动构建一个由三节点三角形单元组成的粗背景网格，使其完全覆盖所关注的求解域，从而启动在任意形状的二维区域上生成网格的过程。图27对此进行了说明，其中展示了一个可能的背景网格，该网格仅由四个单元组成，用于研究流动绕过拐角时的膨胀问题。

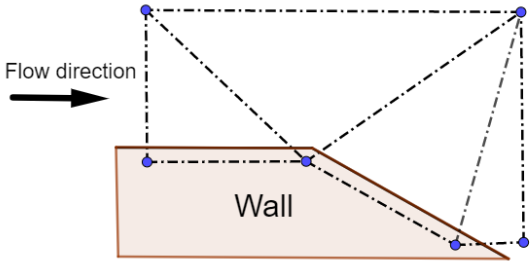


图27：用于覆盖计算域（阴影部分）的粗背景网格，适用于绕拐角超声速膨胀问题。

为了生成单元，方便起见，可定义节点间距 δ、拉伸参数的值s 以及拉伸方向 α。生成的单元在平行于 α 的方向上具有典型长度 $s\delta$，在垂直于 α的方向上具有典型长度 δ （见图28）。背景网格用于为待生成的网格提供这些参数的分段线性空间分布。因此，必须指定背景网格上每个节点处的节点值 δ、 s和 α 。在生成过程中，将通过对背景网格三角形上的指定节点值进行线性插值，获得这些量的局部值。对于初始网格，一维特征的位置通常未知，因此通常将值 $s = 1$ （即不进行拉伸）设为指定值。节点间距 δ 也可以设为均匀分布；但如果显然流动域的某些区域需要提高网格分辨率（例如图28所示问题中的拐角附近），则可通过适当构造背景网格来实现 δ 的变化。请注意，如果初始时要求 δ 均匀且未指定拉伸，则背景网格只需由一个完全覆盖求解域的单元构成。

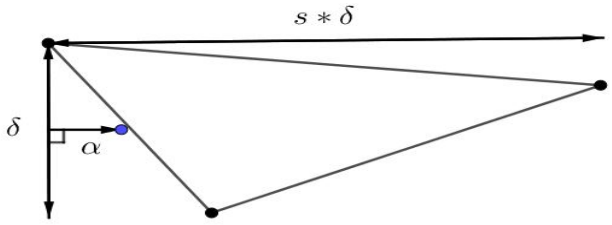


图28：网格三角形中 δ、 α 和 s 的定义。

边界节点的生成 求解域的边界由曲线段组成的闭合环的并集表示，边界节点则布置在这些线段的交点处。对于单连通区域，只有一个闭合环；而对于多连通区域，内部环的数量将等于区域内部开口的数量。外边界的线段按逆时针方向定义。这意味着，沿边界曲线遍历时，待进行三角剖分的区域始终位于左侧。在开始于感兴趣区域内生成三角形之前，必须先在区域的边界上布置附加节点。依次处理每个边界段，并在边界段上生成节点，其间距由 δ、 s 和 α 的插值值确定。

三角形生成 网格生成算法采用了生成前沿这一概念，其形式与 S.H. Lo [45] 提出的方法非常相似。在过程开始时，前沿由连接相邻边界节点的一系列直线段组成。在生成过程中，

任何可用于构成单元边的直线段都称为活动线段，而任何不再处于活动状态的线段都会从前沿中移除。因此，尽管计算域边界始终保持不变，生成前沿却会持续变化，并且每当形成一个新单元时，都必须对其进行更新。

生成网格中新三角形的过程包括以下步骤:

1. 选择前沿上的一条活动线段来生成新三角形。选择方式可以是选取较短的边（如果间距增量沿区域变化，这样可以提高网格质量），也可以选取第一条活动线段（始终遵循创建循环）。

2. 选定线段后，就需要对区域进行抽象，以生成新三角形。具体做法是在初始线段（AB）的中点（M）处计算 δ_M、 α_M 和 s_M，该线段将作为新三角形的底边。如前所述，这通过在背景网格上插值实现。此外，还会进行局部坐标旋转，使 α_M 沿着 x_1 轴。在这个新坐标系中，将生成尽可能规则的三角形。

3. 为避免生成新三角形时出现过度畸变，新的 delta δ_1 按照以下公式计算

$$\delta_1 = \begin{cases} 0.55AB & \delta_m < 0.55AB \\ \delta_M & 0.55AB < \delta_m < 2AB \\ 2AB & 2AB < \delta_m \end{cases} \tag{3}$$

借助 δ_1，将创建三角形的第三个顶点，即点 C。

4. 为考虑与前沿可能存在的不兼容情况，找出所有位于前沿且处于以 C 为中心、半径为 nAB 的圆内的点；按照 Peraire 的设定， $n = 5$，但 n 也可以取其他任何值。将圆内所有点按其到 C 的距离从小到大依次列入列表，其中 N_1 是距离最近的点。

5. 将 C 置于列表首位，除非:

$$AN_1 < 1.5\delta_1 \text{ and } BN_1 < 1.5\delta_1 \tag{4}$$

6. 在后一种情况下，选取新点 N_j 作为新的顶点，同时确保三角形内不包含其他节点（只有 C），且直线 MN_j 不与其他任何活动线段相交。随后形成新单元，将坐标变换回原始空间，并按照前述方式更新前沿。

当前沿中的活动边数量减至零时，三角形生成过程即告结束。

搜索算法 在上述阶段 (b) 中，每次需要用到 δ_M， s_M和 α_M 的值时，都必须通过在背景网格的某个单元中定位点 M 来获得。我们实现了一种高效的搜索算法：对于背景网格中的每个单元“e”，都需要知道与单元“e”共边的三个相邻单元。给定坐标 M 和背景网格中的一个起始单元，即可确定 M 的三个面积坐标 [29] 。

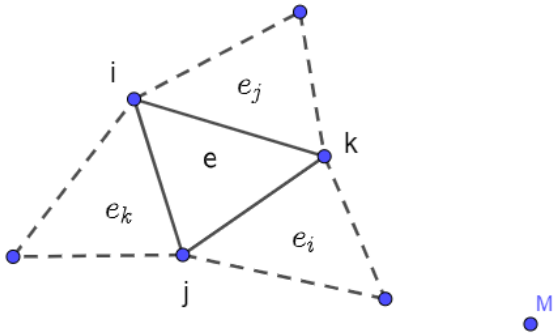


图29：用于在背景网格上定位点 M 的搜索算法。单元 e的面积坐标 L_i、 L_j和 L_k在 M处求得。这里 $L_i(M) < L_j(M) < L_k(M)$，下一个待检查的单元是单元 e_i。

如果每个面积坐标都介于零和一之间，那么该单元就包含点 M。如果不是，则找出面积坐标最小的节点（见图29），这表明接下来要检查哪个单元。这样便避免了搜索背景网格中的所有单元。

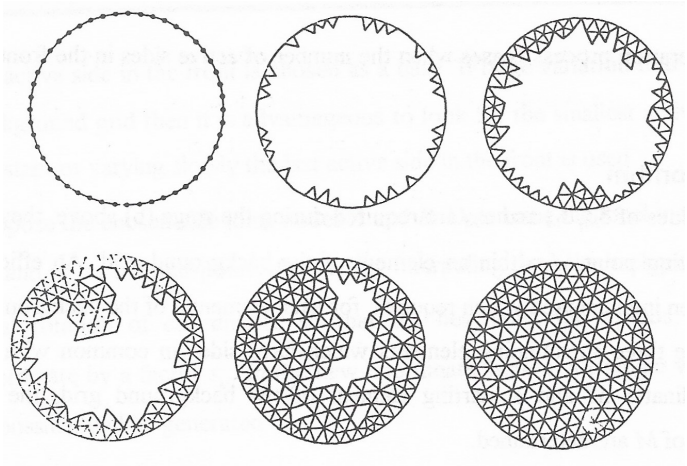


图30：推进前沿技术展示了三角剖分过程中的不同阶段。

推进前沿法以下将要介绍的网格生成算法过程基于前文最初提出的方法 [35]。Peraire（1992年） [63]在此提出了一种改进的三角形生成概念，以辅助网格生成过程，并定义了一个变换 T，它是 α_i 和 δ_i的函数。该变换由一个对称的 N × N 矩阵表示（其中 N（=2 或 3）为维度数），它将物理空间映射到一个空间中，使待处理点邻域内的单元近似为等边形，且单位平均尺寸为 1。这个新空间称为归一化空间。对于一般网格，该变换是位置的函数。变换 T 是 N 个缩放操作叠加的结果，各缩放操作分别沿每个 α_i 方向，缩放因子为 $!/\delta_i$。因此

$$T(\alpha_i, \delta_i) = \sum_{i=1}^{N} \frac{1}{\delta_i} \alpha_i \bigotimes \alpha_i \tag{5}$$

其中， $\bigotimes$ 表示两个向量的张量积。

生成新三角形的过程包括以下步骤（见图31） ：

- 选择前沿的一条边 AB 作为待生成三角形的底边。这里的判据是选择最短的边。生成非结构网格时，这种做法尤其有利。

- 从背景网格中插值得到边的中心 M 处的变换 t，并将其应用于前沿中与三角剖分相关的节点。在此实现中，Peraire 将相关点定义为所有位于以 M 为中心、半径为所考虑边长度三倍的圆内的点。令 A、B、M 分别表示点 A、B 和 M 在归一化空间中的位置。

- 在归一化空间中确定三角形单元顶点的理想位置 P_1 。点 P_1 位于垂直于边且经过点 M 的直线上，距点 A 和 B 的距离为 δ_1 。生成 P_1 的方向由该边的方向决定。值 δ_1 根据基本过程中的不等式选取（见上文）；这里的 A 和 B 分别对应这些不等式中的 A 和 B。只有当边 AB 的特征与背景网格所指定的特征大相径庭时， δ_1 的值才会不同于 1。不过，必须考虑上述不等式，以确保几何可比性。此表达式纯属经验性，不同的不等式也能达到相同目的。

- 选择其他 可能的顶点候选点，并将其按序列入列表。考虑两类点：

  ˆ ˆ– 当前生成前沿中，在归一化空间内位于以 P_1 为中心、半径为 r $= \delta_1$的圆内部的所有节点 Q_1, Q_2,...，以及 ˆ ˆ ˆ ˆ– 沿高 P_1M 生成的点集 P_1,...，P_5 。

  ˆ ˆ ˆ ˆ 对于每个点Q_i，构造一个圆，其中心为 Q_i，位于由点 P_1 和 M 确定的直线上，并且经过点 Q_i、A 和 B。这些圆的中心 Q_i在直线 P_1M 上的位置定义了这些点 Q_i 的排序。创建一个列表，其中包含所有点 Q_i ，离 P_1 最远的点出现在列表首位。将点 P_1,...，P_5 添加到列表末尾。

- 选择最佳连接点。它是在有序列表中第一个能够构成一致三角形的点。通过确保新创建的边均不与前沿中的任何现有边相交，即可保证一致性。

- 最后，如果创建了新节点，则使用逆变换 T^{-1} 得到其在物理空间中的坐标。

- 存储新三角形，并通过添加或移除相关边来更新前沿。

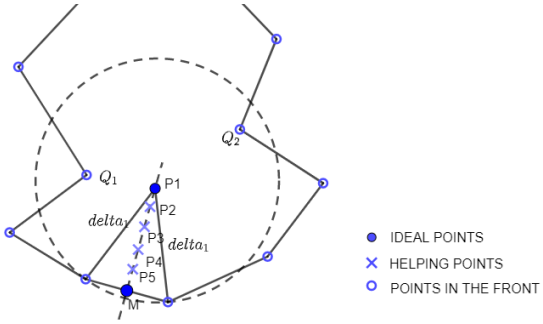


图31：新三角形的生成。

L¨oner（1992年） [52]在其推进前沿法算法版本中提出了一些非常有用的三维改进，这些改进也可以应用于二维。

检查面的相交

推进式生成器最重要的组成部分，是一种可靠且快速的算法，用于检查两个面是否相交。实际应用经验表明，生成器这一部分哪怕只有细微变化，也会对最终网格产生重大影响。与计算几何中的许多其他问题一样，用肉眼判断两个面是否相交似乎轻而易举，但编写代码实现却很复杂

。检查算法基于以下观察：如果任一面的边都不与另一个面相交，那么两个三角形面就不会相交。因此，接下来要为任意两个面构建所有可能的边-面组合，并逐一检查。如果没有发现相交，那么这两个面就不会交叉。对于任意两个面，共有六种边-面组合。考虑到平均需要检查约 40 个相邻面，这种检查面是否交叉的方法会大量占用 CPU 资源。首次实现时，这部分网格生成代码消耗了所需 CPU 时间的 80% 以上。为减少工作量，随后采用了三层方法：

- 最小值/最大值搜索： 其思路是忽略所有面间距离超过预设最小距离的面-面组合。可通过检查每个面的坐标最大值和最小值来实现。

- 局部单元坐标： 检查面交叉的目的是确定新生成的四面体是否破坏了已有的面。其思路是将前述最小值/最大值准则与新四面体的形函数结合起来。

- 对边-面组合进行深入分析：经过步骤 a) 和 b) 的过滤过程后，剩余的所有面都将按照上述方法，利用边-面组合进行分析。

这三种过滤器所需的 CPU 时间依次约增加一个数量级。采用这种实现方式时，面交叉检查仅占网格生成总时间的 25%。

用于尽量减少搜索开销的数据结构
可能将算法效率降低到 O($N^{1.5}$) 甚至 O(N^2) 的操作包括：

a) 查找下一个要删除的面

b) 查找距离给定新点最近的点

c) 查找与给定点相邻的面

d) 对于任意给定位置，从背景网格中查找生成参数的值。这是非结构网格上的一个插值问题。

所有这些操作中都出现了“查找”这个动词。主要任务是设计最佳数据结构，以尽可能高效地执行 a)–d) 中的搜索操作。这些数据结构通常是二叉树或更复杂的树。它们于 20 世纪 60 年代为计算机科学应用而开发。有许多可能的变体 [38]。与流场求解器一样，似乎不存在一种所有当前网格生成器都采用的、定义明确的最优数据结构。对于目前采用的每种数据结构，都可以找到树搜索性能显著下降的病态情况。所使用的数据结构包括：

- Heam-list（[38], [18], [20], [22]），用于查找前沿中下一个要删除的面。
- 四叉树（二维）和八叉树（三维）（[38], [18], [20]），用于定位靠近任意给定位置的点。
- N树，用于确定与某些点相邻的面。

结合这些数据结构，还可以为非结构网格推导出一种最优插值算法 [14]。

提高速度的其他技术
有一些其他技术可用于提升推进前沿网格生成器的性能。其中最重要的包括：

a 过滤：通常，邻近点和面的数量远远过于保守，也就是说，数量偏多。以搜索邻近点为例：一个八分区内最多可能有八个点，但其中只有一个点可能靠近需要移除的面。其思路是过滤掉这些“较远”的面和点，以避免后续的额外工作。虽然搜索操作难以向量化，但这些过滤操作很容易采用向量化方式，从而显著降低整体CPU需求。

b 自动减少未使用的点：随着前沿向区域内部推进并生成更多四面体，树的层数也会增加。这会自动导致 CPU 时间增加，因为需要更多步骤才能到达树的较低层。为了尽可能减少 CPU 时间的增加，所有树都会自动重新构建。所有被四面体完全包围的点都会从树中移除。事实证明，这一过程极为有效。它将网格生成器的渐近复杂度降低到低于 O(N log N)。事实上，在大多数实际情况下，可以观察到线性 O(N) 渐近复杂度；对于较大的问题，CPU 开销在子程序调用开销与平均邻近面数量减少之间取得了平衡。

c 全局 h 加密：虽然基本的推进前沿算法是一种标量算法，但 h 加密可以完全向量化。因此，可以先生成较粗的拉伸网格，再使用经典的 h 加密（[30], [39]）对该初始网格进行全局细化，从而显著加快网格生成过程。采用这种方法通常可实现 1:6 至 1:7 的加速比。

Methode de front Huet 于 1990 年提出的[77]“Methode de front”是一种特殊的推进前沿算法。在这种方法中，节点同样是逐步插入的，但节点和三角形的生成过程遵循不同的判据。这种算法也已在三维中实现。

对于有向曲线前沿 C 上的任意节点 x，都可以定义一个法向方向 Nx，它平分节点 x 相邻边所形成的角度；还可以定义一个标量值 CVx，用于衡量凹度，即形状复杂度（见图 32 (a)）。从前沿上的节点 I 出发，沿 Nx 方向以节点间距 dist 创建一个新节点 II。该新节点 II 与节点 I 以及前沿上位于节点 I 两侧的两个节点相连（见图 32 (b)）。

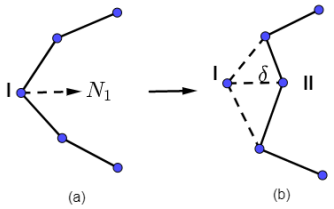


图32：推进前沿法。

节点 II 取代活动前沿上的节点 I，并更新相应的数据。首先，对节点的创建进行控制。选定节点间距 dist，以确保生成尽可能规则的三角形。引入两个不同的系数 Qmin 和 Qmax，用于确定可接受的三角剖分单元的最小和最大尺寸（见图33）。

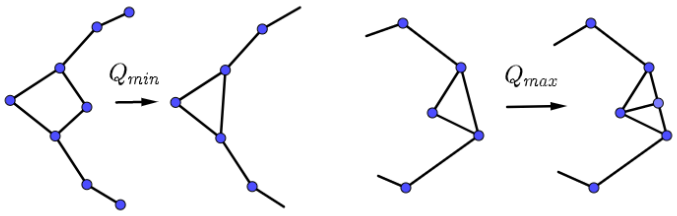


图33：生成单元的最大和最小尺寸。

推进前沿法的实现

- L¨ohner（1990年），[78]：介绍了适用于模拟三维几何形状复杂物体周围强非定常流的网格生成和自适应加密技术。
- Formaggia（1991年），[80]：推进前沿法在不同三维构型中的实现。

- Cabello（1991年）， [79]: 一种用于优化由 AFM 生成的方向拉伸单元的变分法。
- Siden 81990=, [75]: 使用解自适应非结构网格（AFM）对叶片栅中的二维黏性可压缩流进行数值模拟。
- Hwang（1990年）， [83]: 非结构三角网格上的局部隐式总变差递减格式（AFM）。
- Chang（1993年）， [94]: 包含任意数量各向异性材料的任意区域内瞬态热传导的高分辨率温度分布（AFM）。

2.3.4 施泰纳算法

定义： 施泰纳三角剖分是指在现有三角剖分中添加额外的点，以改善某项网格质量指标的任意一种三角剖分。

严格来说，这些笔记中讨论的推进前沿网格生成方法属于一种特殊类型的 Steiner 三角剖分。前文介绍的插入算法也提供了一种生成 Steiner 三角剖分的简单机制。Holmes [65] 证明了，将节点插入现有二维三角剖分中 Delaunay 三角形的外心处，可以改善网格质量指标。这种做法能达到预期效果：将新节点放置在一个位置，使得三角剖分中的其他节点都不可能位于距该节点小于外接圆半径的范围内，见图 34。宽泛地说，新节点的位置会尽可能保守地远离附近的其他节点。

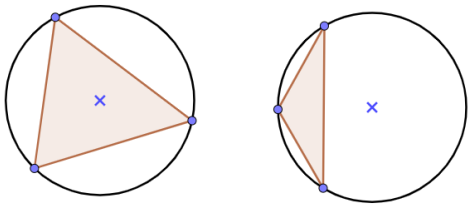


图 34：在锐角三角形和钝角三角形的外心处插入节点。

Warren [81] 和 Anderson [91] 进一步证明了，这类 Steiner 三角剖分在二维网格生成和自适应细化中的实用性。Wiltberger [90]提出的算法也支持基于最小最大（见后文）或最大最小（Delaunay）插入的 Steiner 三角剖分。只有在后一种情况下，在三角形外心处插入节点才真正合理。

（[65], [81], [91]）中介绍的二维施泰纳点网格生成算法包含以下步骤。第一步是对边界数据进行 Delaunay 三角剖分。通常会在远场放置三到四个点，使其凸包包围所有边界点。首先对这些点进行三角剖分，然后按照图 35 所示，使用（[65], [81], [91]）中的 Watson 算法和 [2] 中的 Green 和 Sibson 算法，逐步插入对应于边界曲线的节点。初始三角剖分并不能保证所有边界边都属于该三角剖分。对此可以采用多种方法解决。一种方法是在三角剖分中添加额外的点，以确保生成的 Delaunay 三角剖分包含所有所需的边界边，参见文献 [68]。另一种方法是执行局部边交换，从而生成约束 Delaunay 三角剖分，确保所有边界边都是该三角剖分中的实际边。

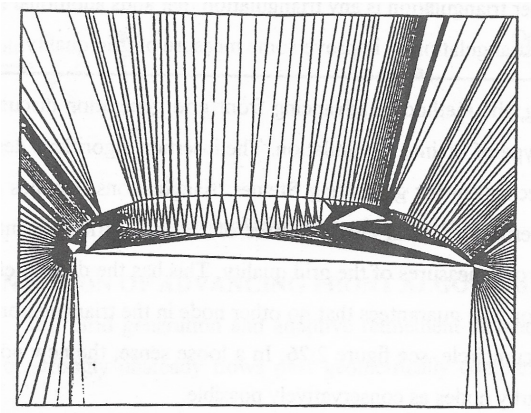


图 35：边界点的初始三角剖分。

无论采用哪种方法，都要标记边界边，使其在三角剖分细化过程中不会被移除。（[65], [81], [91]）中介绍的算法会以任意顺序检查三角形（因此生成的三角剖分并不唯一）。用户必须为三角形细化指定一些质量度量（例如长宽比、面积、内切圆半径），并为该度量设定一个阈值。如果某个三角形未达到阈值，则通过 Watson 算法在该三角形的外心处放置一个新节点来细化三角剖分。必须谨慎选择质量度量，确保在细化时这些度量会降低。以这种方式使用阈值，无法让用户直接控制 Steiner 细化过程中实际生成的三角形数量。Wiltberger 采用了另一种方法，即维护一个按质量度量组织的动态堆数据结构。（堆结构非常适合维护有序列表；对于包含 N 个条目的列表，插入和查询的时间复杂度为 O(logN)。）在整个三角剖分过程中，指定度量值最大的三角形始终位于堆顶，因此能够非常高效地实现使该度量最大值最小化的 Steiner 三角剖分（且结果唯一）。在这种实现中，用户可以指定要生成的三角形数量，或指定该度量的阈值。请注意，可以按字典序对多个度量进行细化。图 36 展示了使用 Wiltberger 算法、采用 MaxMin 插入并基于最大长宽比进行细化的 Steiner 三角剖分。

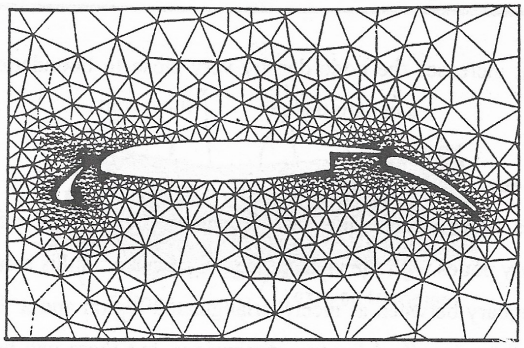


图 36：Steine　r 三角剖分，在外心处插入节点以降低单元的最大长宽比　　io.

Holmes 算法单独使用 Delaunay 三角剖分并不能生成内部点云。Holmes [64] 概述了一种克服这一缺陷的特定方法。将边界节点的 Delaunay 三角剖分作为初始网格。图 37 展示了一个三元件翼型构型的此类三角剖分。初始三角剖分由面积很大、偏斜度很高的三角形组成，这些三角形超过了最大面积或最大偏斜度的某些阈值。Holmes 提议用外接圆半径与内切圆半径两倍之比来衡量偏斜度。

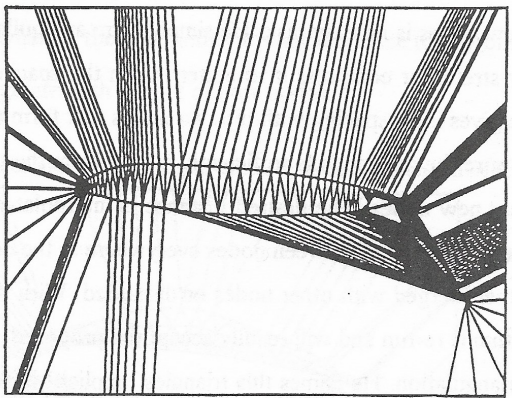


图 37：边界节点的三角剖分。三单元翼型边界上的节点与外边界上的节点相连　　（未显示）。

检测到此类劣质三角形后，将在该三角形的外心处插入一个新节点进行细化。细化从网格中面积最大的三角形开始，直到所有三角形都小于第一个阈值。随后继续细化偏斜三角形，先处理外接圆最大的三角形。所有偏斜三角形的面积都小于第二个面积阈值后，即得到最终网格。按偏斜度进行细化还会隐式地提高边界附近的节点密度，使节点间距更小。

前沿节点生成/ M¨uller M¨uller [92]在此介绍了一种技术，将细化边界节点的 Delaunay 三角剖分与前沿推进的思想结合起来。在这种边界节点的 Delaunay 三角剖分中，前沿将呈现为一个“良好”三角剖分区域与一个“较差”三角剖分区域之间的边界。该方法将针对二维进行描述，但将其实现于三维并不会遇到太多障碍。

观察一组边界节点的 Delaunay 三角剖分（见图 37）可以发现，几乎每个边界面要么是某个具有一个极锐角的三角形的短边，要么是某个具有一个极钝角的三角形的两条短边之一。按照我们关于前沿将优质三角形与劣质三角形分隔开的构想，我们将边界视为此类前沿的初始位置。起初，我们没有优质三角形，但会引入一层位置良好的节点，使前沿得以推进。

新节点的构造很容易。我们只需将前沿面与其组成一个等边三角形，然后将其拉伸或压缩，使之更好地符合背景网格的间距要求；这样，便能为构成该面的两个节点提供理想的间距。还需要进一步检查，以确保新节点与网格中的其余节点以及其他新节点之间的距离足够大。所需距离通过背景网格进行插值，该背景网格唯一确定了区域内任意位置节点之间的距离。间距不合适的节点会与其他节点合并，或被丢弃。放置好这些新节点后，将再次运行Delaunay算法；由于所提出的优质三角形不会给其三角剖分带来明显偏斜，算法可以轻松接受它们。下文将这些三角形称为“隐式”三角形。此外，新节点之间也会形成优质三角形，因为可以保证它们之间具有良好的间距，这些就是“隐式”三角形。在区域的其余部分，Delaunay仍需构造锐角单元，不过其形状已有所改善。

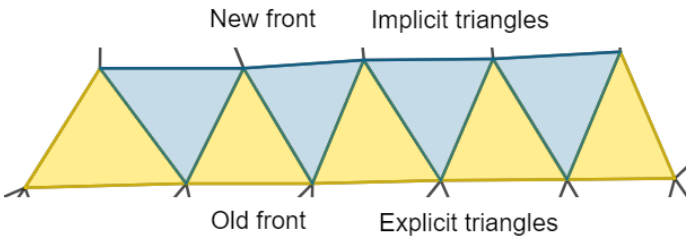


图 38：沿旧前沿形成并构成新前沿的显式三角形（条纹状）和隐式三角形（方格状）。

同样，这些锐角单元的短面标志着优质单元区域与仍在等待细化的区域之间的前沿，见图38；这一过程可以重复进行，直到所有劣质三角形都消失。因此，该算法可以归纳为以下步骤：

1. 检测网格中的所有劣质三角形，并找到其短面。2. 找到一组节点，使其与这些短面形成优质三角形。3. 检查新节点是否与结构中已插入的任何其他节点距离过近。4. 检查新节点之间是否彼此距离过近。5. 使用这组新节点重新进行三角剖分。

这些步骤将重复进行，直到无法再通过节点插入实现进一步改进。

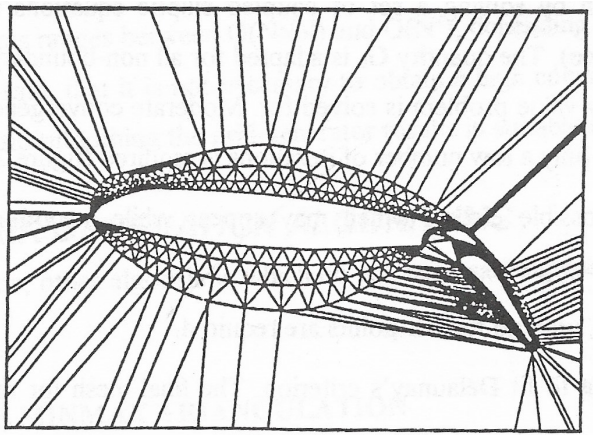


图39：插入三行节点后，三单元翼型周围的网格 .

Vilsmeier/H¨anel算法 该算法是基本Holmes算法的扩展。生成过程从仅包含边界点的粗网格开始，通过细化、重新网格划分和平滑步骤递归地获得最终网格 [89]。具体如下：

1. 使用 Delaunay算法对边界上的给定点进行三角剖分（见图40 a）。2. 对每个边界点，存储两条边界边中较长边的长度，并将其乘以一个大于 1 的统计因子。这个量称为限制边长 Gk。对于边界点，Gk 永远不会改变。3. 在所有至少有一条边长于构成该边的两个点的 Gk 最小值的三角形中心插入一个新点。将构成被细化三角形的三个点的 Gk 值的平均值作为新点的 Gk 预测值。4. 通过形成三个新三角形并删除被细化的旧三角形，将每个新点与周围连接起来。调整所有邻接关系（图40 b）。5. 执行重新网格划分（边交换）。此步骤中进行重新网格划分的目的不是满足德劳内准则，而是获得一个周围有六个三角形的点，同时避免出现极端角。显然，并非所有点都能达到这一目标（图40 c）。6. 通过递归算法求解一组耦合椭圆方程，对网格进行平滑。调整所有非边界点的 Gk 值，也就是说，需要求解一个边值问题。此步骤中只需适度收敛，因此只需要较少的迭代次数（图40 d）。7. 修正可能出现的错误，这些错误可能在步骤 f) 的平滑过程中产生，尤其是在凹多边形中。8. 返回 C；继续执行，直到不再需要添加新点。

9. 执行　　重新网格划分以满足德劳内准则。此示例的最终网格如图40所示　　e.

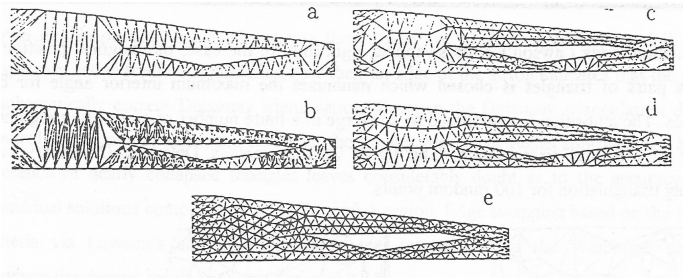


图40：冲压式喷气发动机进气道网格的生成过程。

计算时间
计算时间不仅取决于最终网格的点数，还取决于几何形状。如果需要执行修正（步骤7），计算时间会大幅增加。

对边界点进行三角剖分（步骤1）所花的时间可以忽略不计。引入新点（步骤3）所需的时间，对于整个三角剖分而言为 O(N) 量级；连接新引入的点（步骤4）所需的时间也是如此。重新网格化工具的 CPU 时间很难预测。循环次数通常很少，但每次循环都会影响大量三角形。这里我们认为，重新网格划分的计算时间对于每个三角形仍为 O(1) 量级，因此对于整个三角剖分为 O(N) 量级。

最耗时的工具是椭圆平滑（步骤 6）。求解边值问题的计算量介于 O($N\sqrt{N}$) 和 O(N^2) 之间，具体取决于所采用的加速技术。需要指出的是，平滑时无需达到很高的收敛程度。即使不进行椭圆平滑，该网格生成器也能生成适用于计算流体力学应用的网格。

2.3.5 其他生成算法。

在本节　　中，我们将探讨其他不一定会生成 Delaunay 三角剖分的算法　　d.

最小最大三角剖分 正如 Babuska 和 Aziz [60] 所指出的，从有限元的角度来看，最大最小（Delaunay）三角剖分并非必不可少。关键在于，不能有任何角接近 180°。换言之，能够使最大角最小化的三角剖分更为理想。这类三角剖分称为 MinMax 三角剖分。生成二维 MinMax 三角剖分的一种方法是采用 Lawson边交换算法。在这种情况下，对于凸三角形对，选择使两个三角形的最大内角最小的对角线位置。采用与 Delaunay 三角剖分类似的论证，可以保证该算法在有限步内收敛。图4展示了由 100 个随机点构成的 Delaunay（最大最小）三角剖分和 MinMax 三角剖分。

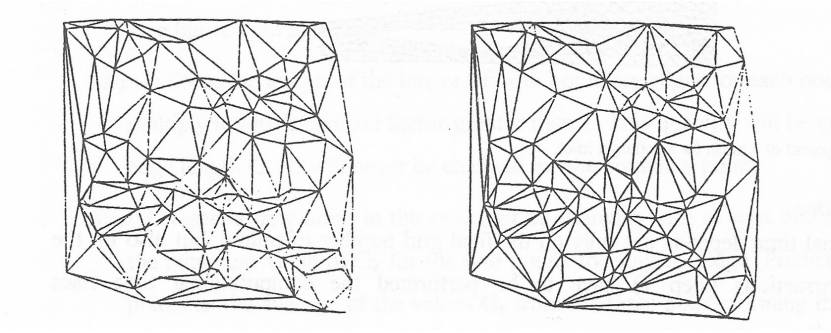


图41：Delaunay 三角剖分和最小最大三角剖分。

请注意，使用 Lawson 的算法进行局部最小最大优化，可能只能得到局部最优的网格，而不一定达到全局最小值。获得全局最优的最小最大三角剖分是一项更加困难的任务。截至目前，最佳算法（Edelsbrunner、Tan 和 Waupotitsch [76]）具有

O($N^2 logN$) 的高复杂度。Wiltberger [90]实现了 Green 和 Sibson 算法[2] 的一个版本，该版本经过修改，可通过在局部边交换上进行增量插入，生成局部最优的 MinMax三角剖分。该算法采用递归编程实现，并进行完整的前向和后向传播。这是生成局部优化网格的必要步骤。MinMax 三角剖分已被证明在计算流体力学中非常有用。图4 2 显示了翼型后缘区域附近的 Delaunay 三角剖分，其中存在极端的点聚集。

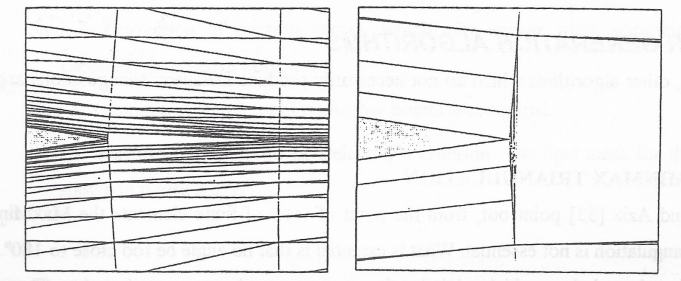


图42：翼型后缘附近的 Delaunay 三角剖分。

初步观察时，翼型后缘附近的网格似乎存在缺陷。进一步检查并对翼型后缘附近区域进行极度放大后发现，该网格在数学上是正确的 Delaunay 三角剖分，但无法控制最大角；后缘附近单元的角度接近 180°。几乎塌缩的三角形的存在，使人在很大程度上怀疑后缘区域中计算所得任何数值解的准确性。通过 Lawson 算法，依据最小最大判据进行边交换，可得到所需结果，如图43所示。

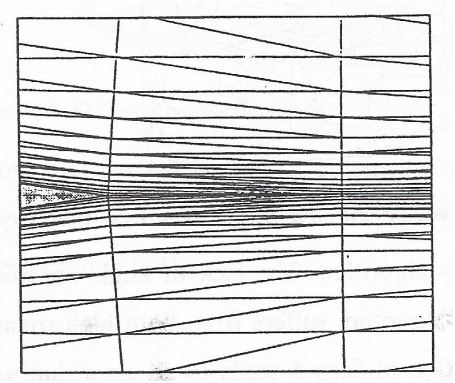


图4　　3：翼型后缘附近的最小最大三角剖分　　。

贪心三角剖分 贪心方法不会撤销先前执行的操作。贪心三角剖分会持续添加与当前三角剖分兼容的边（不允许边交叉），直到三角剖分完成，也就是满足欧拉公式。三角剖分的一个目标可能是选取一组总长度最短的边。贪心算法能做到的最好选择，是采用一种局部准则：每次只考虑当时可用的最短边，并将其加入当前三角剖分。（这并不能得到总长度最短的三角剖分。）请注意，贪心三角剖分很容易适用于包含内部边界且外边界非凸的约束三角剖分。在这种情况下，只需在候选边的排序中将边界边列在最前面即可。完整算法如下所述。

算法： 贪心三角剖分

- 将三角剖分 T 初始化为空。
- 计算 n_2 候选边。
- 对候选边池进行排序。
- 从有序池中移除当前边 e_S

- 如果（ e_S 与 T 的边不相交），则将 e_S 添加到 T。
- 如果（不满足欧拉公式），则转到步骤 4。

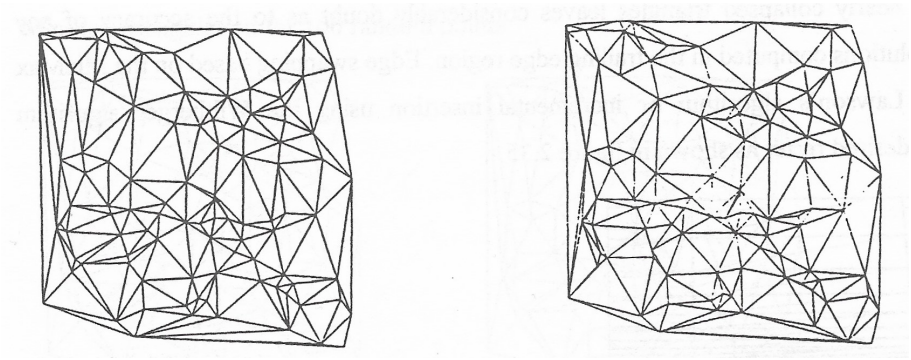


图 44：Delaunay 三角剖分与贪心三角剖分。

图 44 对比了 Delaunay 三角剖分与贪心算法。贪心三角剖分中缺乏角度控制这一点显而易见。贪心算法不仅运行时间长，而且存储开销也大。事实上，步骤 5 的朴素实现会导致算法复杂度为 O(N^3)。Gilbert [6] 给出了高效的实现技术，使复杂度降低至 O(N^2log N)，存储开销为 O(N^2)。

数据依赖三角剖分 与网格自适应不同，数据依赖三角剖分假设顶点的数量和位置固定不变。在这些顶点所有可能的三角剖分中，目标是找到满足数据依赖约束的最佳三角剖分。Nira、Levin 和 Rippa [74], 将若干数据依赖约束与分片线性插值结合起来进行研究。为了判断新网格是否比之前的网格“更好”，需要为每条内部边定义一个局部代价函数。证明尤其有效的两种选择是 JND（法向导数跳跃）和 ABN（法线夹角）。

回想一下，如果选择使相邻三角形之间的最小角最大化的 cons 函数，就会得到 Delaunay 三角剖分（ Lawson 算法）：虽然为所有代价函数求得全局最优解是理想的，但在许多情况下，这样做的代价可能很高。另一种策略是放弃追求全局最优三角剖分，转而寻求局部最优三角剖分。这里再次使用 Lawson 算法。请注意，使用 Lawson 算法时，我们要求每次进行边交换后，全局度量都减小。这不像之前那样简单，因为每条边都有代价函数。尽管如此，其影响范围很小，也很容易确定。图 45 绘出了数据依赖三角剖分。

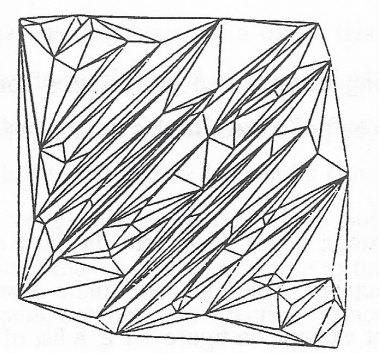


图 45：数据依赖三角剖分。

请注意，使用此方法得到的三角剖分并非全局最优，而且高度依赖边的访问顺序。 [73] 中提到了几种可能的排序策略。

2.3.6 数据结构

用于表示非结构网格的数据结构，其选择因要执行的算法操作类型而有很大差异。本节将讨论几种最常见的数据结构。假设网格中的顶点、边、面等均已编号。在大多数情况下，物理坐标只需按顶点编号列出。“标准”的有限元（FE）数据结构会列出每个单元的连接关系。例如，在图46中，会给出每个三角形的三个顶点列表。

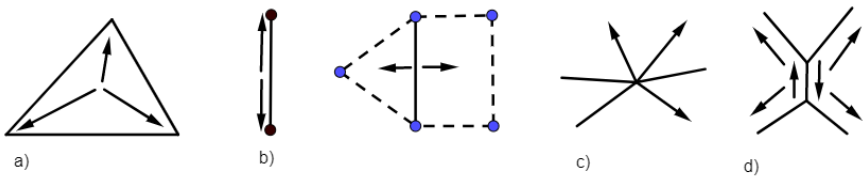


图46：数据结构：a) 有限元数据结构 b) 边结构 c) 出度结构 d) 四边结构。

有限元结构可自然地扩展到三维。有限元结构广泛用于固体和流体的有限元求解器。

对于平面网格，另一种典型结构是边结构（图46 b），它通过为每条边存储一个四元组来列出顶点的连接关系及相邻面；四元组包含每条边的起点和终点，以及共享该边的两个面（单元）。这种结构便于遍历网格，在某些网格生成算法中十分有用。（使用有限元结构不易完成这种遍历。）扩展到三维时，需逐面处理：给出面的顶点以及相邻的两个体，并区分不同的面类型。

第三种数据结构通过顶点列表表示连接关系，如图46 c所示。最直接的方法是为每个顶点列出所有相邻邻点（通常采用链表）。许多稀疏矩阵求解器软件包使用行或列存储方案指定矩阵的非零元，即列出给定行或列中的所有非零项。对于仅涉及网格相邻邻点的离散化，这等同于指定顶点列表。指定所有相邻邻点的另一种方法是对网格的边进行定向。在这种情况下，只列出从某个顶点向外指向的边。下一节将说明，如何通过对平面网格中的图进行定向，使每个顶点至多有三条出边，从而构造出度列表。这在渐近意义上是最优的。将出度结构扩展到三维并非易事，针对非平面图如何获得最优边定向的算法仍在开发中。

这里讨论的最后一种结构是 Guibas 和 Stolfi 提出的四边结构 [82], 见图===。每条边都存储为一对有向边。每条有向边都存储其起点，以及指向其左侧区域中下一条和上一条有向边的指针。四叉树结构在网格生成中非常有用，因为区分拓扑方面和几何方面至关重要。Dobkin 和 Laslo [70] 以及 Brisson [69]已将该结构扩展到三维布局。

从前面的部分可以看出，要成功实现生成算法，就需要使用能够高效执行某些排序和搜索操作的数据结构。例如，AFM 的生成前沿需要一种数据结构，既能高效插入/删除边界边，又能高效识别与空间中指定区域相交的边界边。

在 n 维空间中，确定 N 个节点的集合中哪些节点位于指定子区域内，这个问题称为几何搜索。已有多种算法（[5], [1],[54], [61]）被提出，可解决上述或等价问题，且其计算开销与 log(N) 成正比。当考虑的对象不是点，而是线段、三角形或三维四面体等有限尺寸物体时，问题复杂度会显著增加。这里常见的问题称为几何相交，即找出与所考虑空间中的某个子区域重叠的物体。已有用于解决二维中这一问题的算法 [46]，并已用于确定平面中几何物体之间的相交。据我们所知，唯一能够解决三维中这一问题的算法基于交替数字树（ADT） [72]。

2.3.7 网格自适应

用于分析可压缩流问题的数值方法，必须能够充分描述经常出现的狭窄高梯度区域（例如激波）；这些区域通常嵌在流动变量缓慢变化的大面积区域之中。由于分析人员事先并不知道这些高梯度区域的位置，显而易见，自适应网格方法和后验误差估计器将在开发此类问题的高效求解技术方面发挥重要作用。迄今为止，此类问题已成功采用网格自适应技术。为改进网格移动，这些技术通常与显式时间积分过程结合实施，并专为稳态问题分析而设计。

网格移动 网格移动可以通过推进解使其趋于稳态来实现；在某些时刻，用具有一定刚度的弹簧替换网格边。随后移动节点，直到弹簧系统达到平衡。弹簧的强度通常基于局部解梯度。

在二维中，此类技术已应用于四边形有限体积法 [36] 以及三角形有限元法 [63] 。由于不会添加新节点，节点连接关系也不会改变，因此这种方法的一个缺点是，最终计算的精度受初始网格的结构和分辨率限制。

网格加密 网格加密算法已应用于三角形和四面体有限元法（[58], [56]），以及四边形有限体积法（[55], [62]）。基本方法是在初始（粗= ）网格上推进解使其趋于稳态，然后使用误差或加密指标标记需要细化的计算单元。标记的单元会自动细分，随后继续计算并重复这一过程，直到分析人员对解的质量满意为止（见图47）。

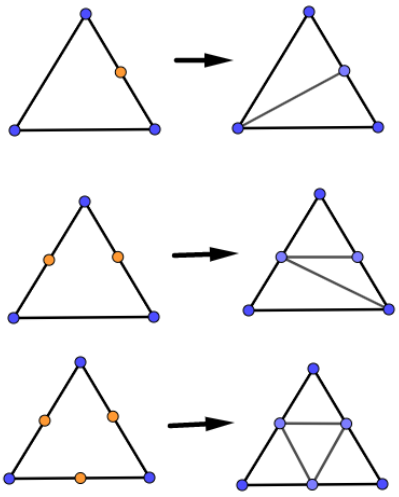


图47：网格加密过程，展示三角形单元的三种可能细化情况

尽管该方法在实践中已被证明有效，但也存在一些缺点。例如，在求解二维问题时，一维流动特征（如激波和边界层）附近的面积无法得到有效细化，而且每次细化都会导致所用单元数量迅速增加（见图 ==）。

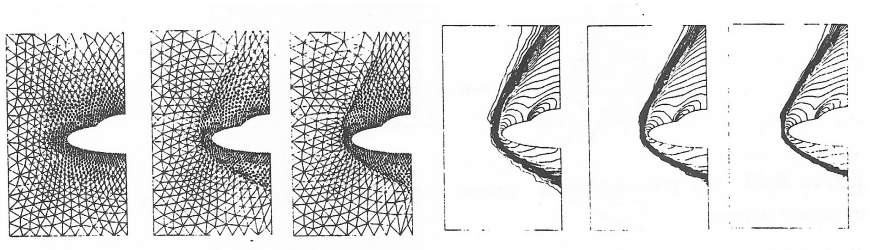


图48：自由来流马赫数为 8.15、攻角为 30 度时，超声速流绕过双椭圆的流动情况，展示了应用自适应网格移动后得到的一系列网格和解。

解决这一问题的一种可能方法是结合网格加密与网格移动程序。图49展示了这一方法：将移动程序应用于图48中的最终加密网格。

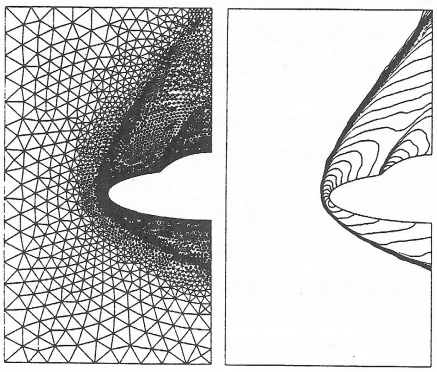


图49：自由来流马赫数为 8.15、攻角为 30 度时，超声速流绕过双椭圆的流动情况，展示了对预先加密的网格应用自适应网格移动后得到的一系列网格和解。

3 MeshGen 2.0

Meshgen 2.0 是一款 Windows 应用程序，由用户创建，用于在用户设计的区域中创建并绘制网格。它使用 C++ 在 Qt Creator 中编写。此面向对象语言可以优化应用程序运行时的内存使用，但也使程序结构更难解释。为此，首先将说明用户界面背后的结构如何运作。

4 Meshgen 结构

应用程序项目由多个类构成，具体如下：

- **MainWindow 类**。这是主要的类，定义了所有界面按钮及其交互。该类会调用其他所有类，所有操作都在此处执行。
- **TextEditReader 类**。该类读取界面右侧文本编辑器中编写的边界段，并将这些信息传递给其他类
- **LienzoCreator 类**。界面中的所有绘图都在 MiddleWidget 中完成。
- **Dialog 类**。这是一个用户界面类，用于创建对话框；用户想要更改边界和网格宽度设置时，该对话框会弹出。
- **Trimpack 类** 是用于在 Fortran 代码和 C++ 代码之间充当“桥梁”的类。
- **TRIMPACK.** 其余文件（.f 或 .lib）是 Juan Manuel Tiz´ón 提供的三角形非结构网格生成软件包。

信息流如图 50 所示。用户打开 .mg 文件（其中定义了域形状）后，文件内容会显示在界面右侧的白色窗口（文本编辑器）中。程序会读取文本编辑器中的所有内容，因此用户可以按需修改。关闭应用程序时，会弹出一个窗口，询问用户是否保存对文件所做的更改。如前所述，信息流从右侧的文本编辑器开始，负责读取这些信息并将其发送到 MainWindow 类的类是 TextEditReader 类。

间距、非结构函数的版本或平滑/边交叉相关信息由 MainWindow 类自行读取。

定义区域和网格划分属性后，用户即可运行程序，此时会调用 TRIM- PACK（Fortran）函数并完成绘图。负责绘制中间小部件的类是 LienzoCreator 类。界面顶部的“配置菜单”允许用户更改网格和边界线的颜色、大小及样式。

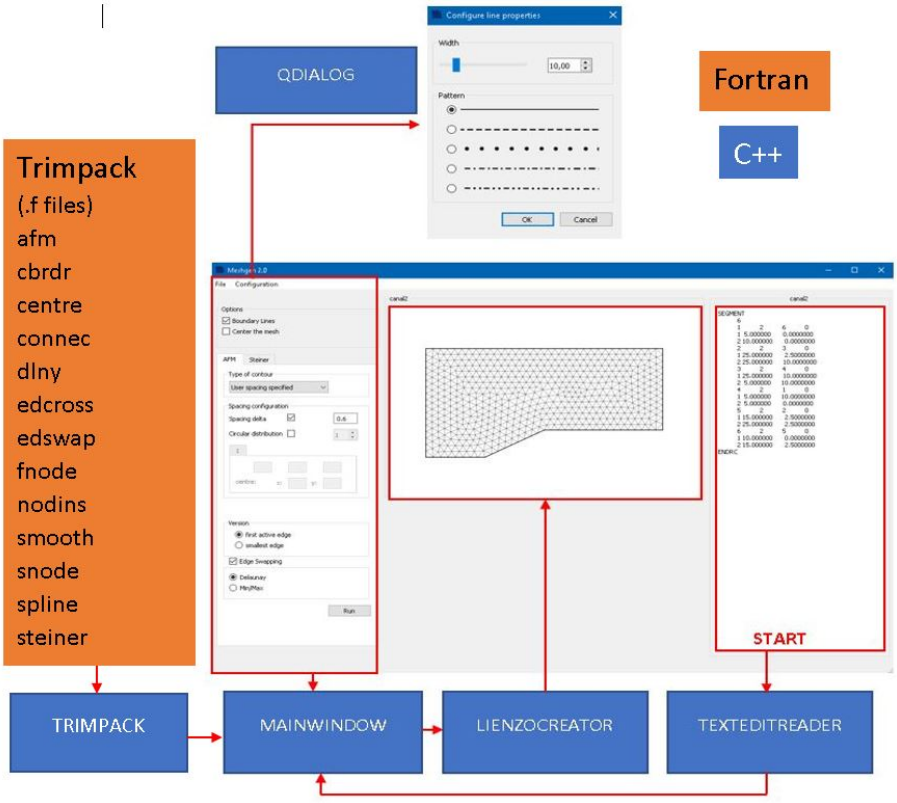


图 50：信息流

5 用户界面

在本节中，我们将　程序的相关内容将得到说明。每个按钮的作用，以及每个按钮会产生的影响
用于创建　非结构网格。

5.1 文件菜单



图51：文件菜单

此菜单位于界面左上角，负责管理包含域边界的 .mg 文件。用户可以创建新文件、打开现有文件、保存其在界面右侧文本编辑器中所做的更改，或将文件另存为其他名称。此外，如果用户在文本编辑控件中修改了文件并尝试关闭应用程序，系统会弹出窗口，询问用户是保存更改还是放弃更改。

5.2 输入文件：.mg 格式

用户要使用该应用程序，首先需要打开域形状文件。该文件必须按照特定格式编写，程序才能读取其中的信息。

.mg 文件由不同的“记录段”组成。这种文本文件的编写方式允许程序从文件中读取多条信息。每个部分以程序员选定的键名开头，以“ENDRC”结尾。在本应用程序中，由于其性质所限，只需要一个“记录段”，即定义边界线的 SEGMENT 段。下面将通过一个尽可能简单的示例来解释该文件的结构：一个正方形。

首先需要了解的是，区域的外部直线必须从左向右定向，而内部直线必须从右向左定向（始终按**顺序**定义各条边）。这种方向有助于网格生成器确定应在线段的哪一侧创建网格。

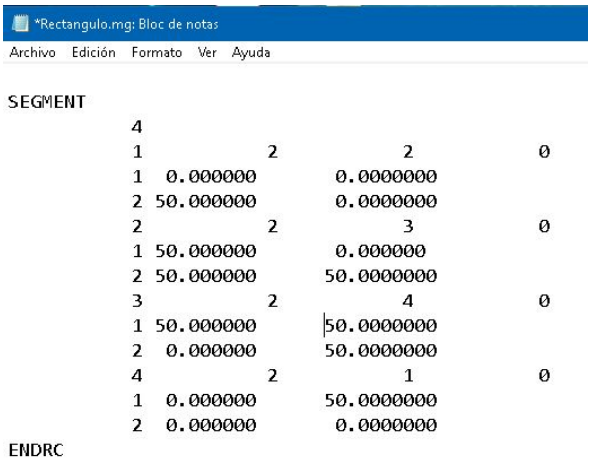


图5：rectangle.mg 文件

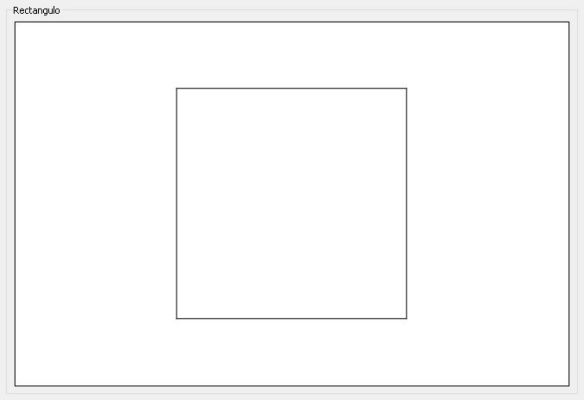


图5：正方形

务必注意，数字之间必须使用空格分隔，**绝不能**使用制表符，否则程序将无法读取文件。

-”SEGMENT” 如前所述，用于开始记录计算域边界部分
- ”4” 计算域边的数量 n.
- ”1” 线段编号。 ”2” 定义该线段的点数（这允许用户在一个线段内定义包含多个点的曲线边）。 ”2” 与当前线段相连接的线段编号。 ”0” 必须始终为 0。

- ”1” 的编号 节点。 ”0.000” 点的 x 坐标。 ”0.00000” 点的 y 坐标。 - ”2” 的编号 节点的。”50.000 ”点的 x 坐标。”0.00000”点的 y 坐标。

这会绘制从点 (0,0) 到 (50,0) 的线段 1。线段不必按顺序书写，只要与其连接的线段正确即可。最后这三行格式会重复另外 3 次，每次对应一条线段。

- “ENDRC”这一行表示线段定义记录结束。

5.3 选项按钮

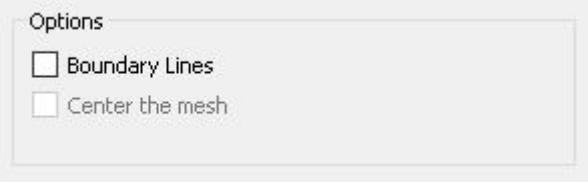


图5：选项区域

打开 .mg 文件后，如果用户点击 firstcheck 按钮，就会绘制区域的边界线。第二个按钮则会将网格居中显示在中间小部件中。

5.4 AFM 页面

它由四个部分组成：

5.4.1 轮廓类型

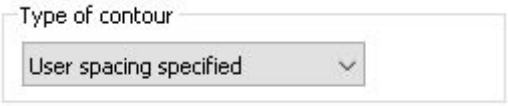


图55：AFM 轮廓类型部分

第一个部分用于指定所使用的轮廓类型。共有三种不同的轮廓类型，但应用程序中只有一种可用。

- **用户指定间距**。这是本应用程序采用的间距。在网格创建过程开始时，程序读取用户定义的间距，并使用该间距划分边界线段。
- **用户定义边界**。此功能已在本应用程序中启用。边界不会由程序划分，而必须由用户进行划分。这样，用户便可以准确指定需要网格点的位置。
- **自由节点**。此功能已在本应用程序中启用。用户定义点云，程序则设置外部边界，并使用用户定义的点创建网格。

5.4.2 间距配置

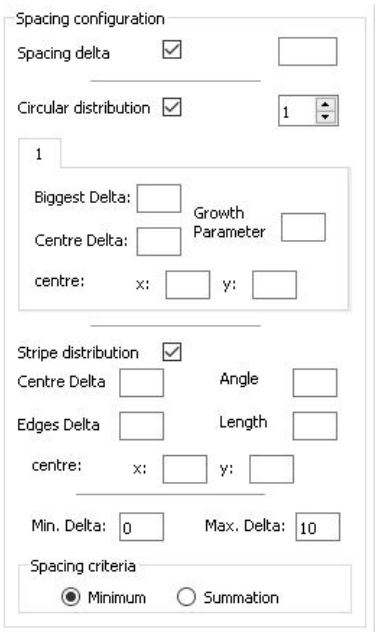


图56：AFM 间距部分

间距部分是用户定义网格尺寸和分布的地方，有多种方式可用于定义所需的网格。本部分将在“间距函数”（6）一节中详细说明。

5.4.3 AFM 版本



图5 57：AFM 版本部分

在第三部分中，用户定义所使用的 AFM 版本，共有两个版本：

- **第一个活动边**。此版本会取出栈中的第一个活动边，并以此进行三角剖分。对于区域中采用恒定间距因子的情况，推荐使用此版本。
- **最小边**。此版本会取出栈中最小的边，并以此进行三角剖分。对于区域中采用变化间距因子的情况，推荐使用此版本。在复杂网格中，这是更好的选择。

这两个版本之间的差异可以很容易地从接下来的图中看出。

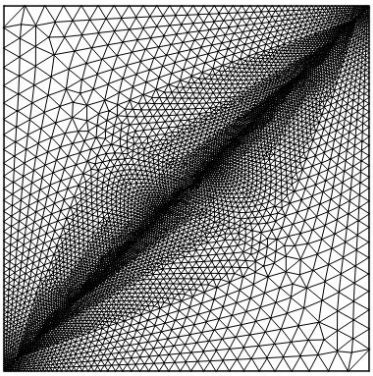


图58：第一个活动边版本

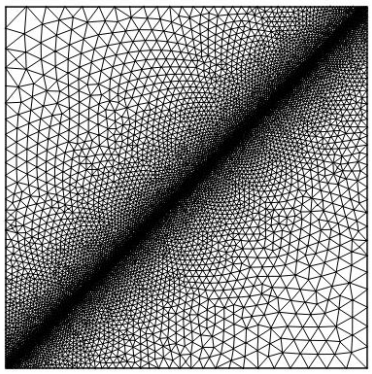


图59：最小边示例

5.4.4 边交换



图60：AFM 边交换部分

AFM 页面最后一部分是边交换部分。用户可以在此选择是否进行边交换以及采用哪种类型，共有两个选项：

- **Delaunay 判据**。此版本会使两种可能的三角剖分配置中的最小角最大化
- **最小/最大准则**。此版本会在两种可能的三角剖分配置中使最大角最小化

5.5 Steiner 页面

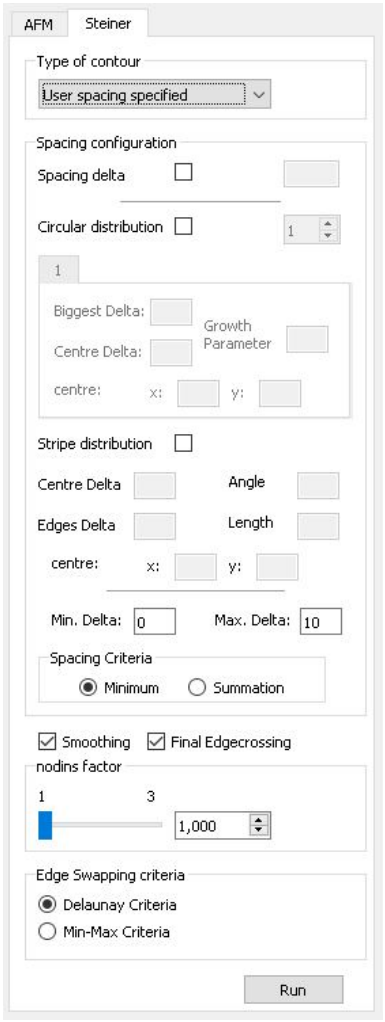


图 61：Steiner 页面

它由五个部分组成:

5.5.1 轮廓类型



图 62：Steiner 轮廓类型部分

第一个用于 sp 指定所用的轮廓类型。轮廓有三种不同类型，但只有一种可在应用程序中使用。它与 AFM 方法相同。

- **用户指定间距**。这是本应用程序中使用的类型。在网格创建过程开始时，读取用户定义的间距，并按此间距对边界线段进行划分。
- **用户定义边界**。这些边界不会由程序划分，必须由用户划分。这样，用户就可以精确指定需要网格点的位置。

- **自由节点**。此应用程序已启用此功能。用户定义点云，程序则设置外部边界，并使用用户定义的点创建网格。

5.5.2 间距配置

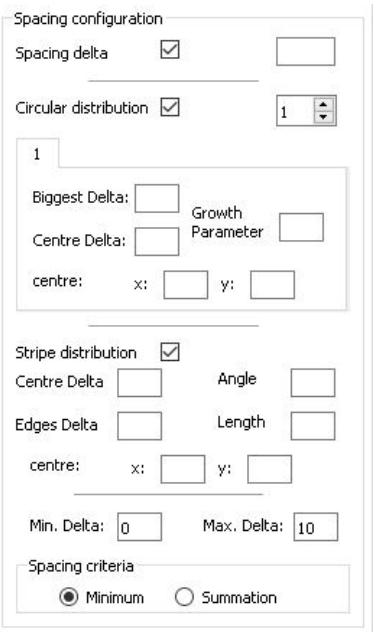


图 63：Steiner 间距部分

间距部分用于定义网格尺寸和分布，定义所需网格的方法有多种。本部分将在“间距函数（6）”部分中进行更详细的说明。

5.5.3 平滑与最终边缘交叉按钮



图 64：Steiner 平滑与边缘交叉部分

第三部分由两个按钮组成 ns:

- **平滑**。网格平滑会改变内部节点的位置，但不会改变网格的拓扑。

- **最终边缘交叉**。检查边之间是否可能发生交叉，并修复交叉。

5.5.4 Steiner 因子



图 65：Steiner 因子部分

此因子控制网格中三角形的大小。如果因子值为 1，所有三角形都将小于增量三角形（即网格中每个点的间距部分所定义的三角形）；如果因子值为 3，所有三角形都将大于增量三角形。

5.5.5 边交换判据



图 66：Steiner 边交换部分

Steiner 页面最后一部分是边交换部分。在这里，用户可以选择是否进行边交换以及交换类型，共有两个选项：

- **Delaunay 判据**。此版本会在两种可能的三角剖分配置中，使最小角达到最大值
- **最小/最大准则**。此版本会在两种可能的三角剖分配置中，使最大角达到最小值

6 间距函数

在非结构化网格中，可以通过简单的函数定义区域内每个点处三角形的尺寸。本程序定义了三种不同的间距分布：

- 均匀间距 Delta。
- 圆形分布。
- 条带分布。

均匀间距 Delta

此分布 ge 在整个区域生成均匀网格。用户在间距增量部分中设置值。



图 67：间距增量部分

三角形大小没有上限，但下限为 0.001。设置此下限是因为绘图精度无法达到更多小数位。不过，程序可以生成这种网格。对于质量良好的网格，delta 的常用值比区域边界边的长度小一个数量级。

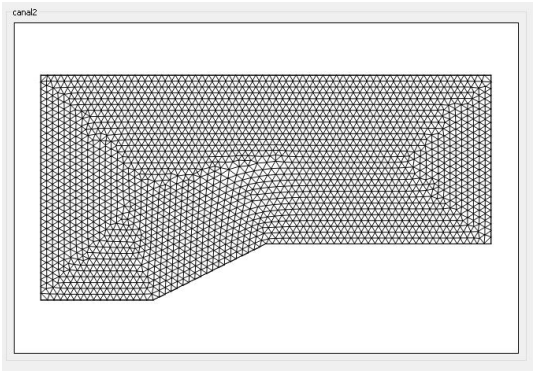


图 68：AFM 均匀间距 Canal2.mg delta 值：0.3

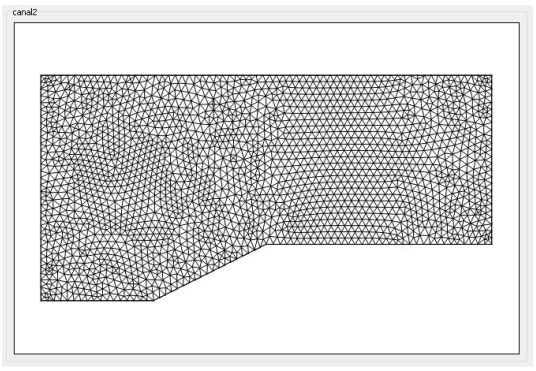


图 69：Steiner 均匀间距 Canal2.mg delta 值：0.3

圆形分布

这种分布会生成一个从用户选定的中心开始、网格大小逐渐增大的网格。

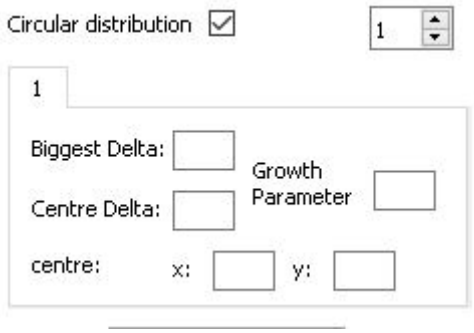


图70：圆形分布部分

此分布使用的输入为：

- δ_A：“最大间距”。这是网格达到的最大值。
- δ_B：“中心 Delta”。这是圆形分布中心处的值。
- β：”增长参数”。它定义了该方法的增长参数。当 β 增大时，压缩后的网格会变小。
- X_s：”中心：x”。它设置分布中心的 x 坐标。
- Y_s：”中心：y”。它设置分布中心的 y 坐标。

控制此分布的方程为：

$$\delta = \delta_A + (\delta_B - \delta_A) * e^{-\beta * r^2} \tag{6}$$

其中 $r = \sqrt{(X - X_s)^2 + (Y - Y_s)^2}$, 表示正在绘制网格的点与分布中心之间的距离。使用此函数时，delta 被限制在 δ_A 和 δ_B之间。

分析此函数的极限：

$$\lim_{r \to 0} \delta = \delta_B \tag{7}$$

$$\lim_{-\beta * r^2 \to \infty} \delta = \delta_A \tag{8}$$

如前所述， β 变量控制网格的增长速度。通过绘制同一个函数并更改 β 的值，可以轻松看出这一点。

r（横坐标轴）- delta（纵坐标轴） $\beta = 1$

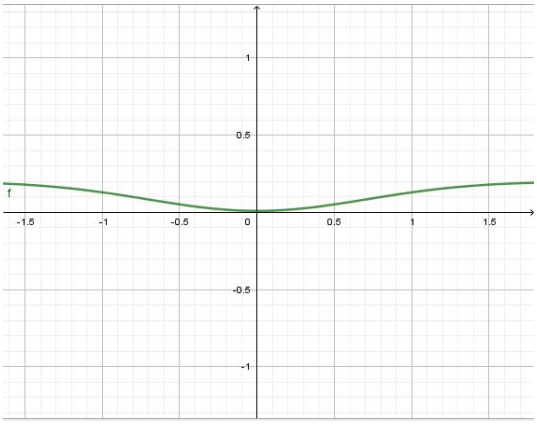


图 71：圆形分布函数， δ_A =0.2, $\delta_B = 0.01$

r（横坐标轴）- delta（纵坐标轴） $\beta = 0.1$

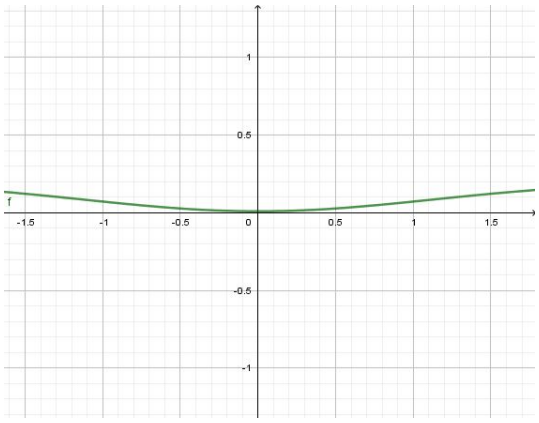


图 72：圆形分布函数， δ_A =0.2, $\delta_B = 0.01$

这两个示例生成了以下网格：

带有 $\beta = 1$ 的圆形分布

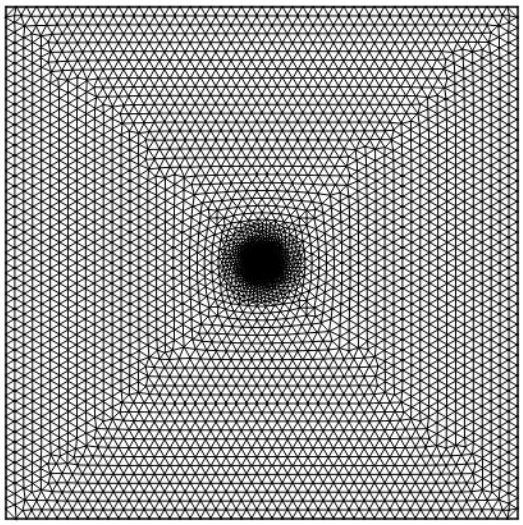


图 73： $\delta_A = 0.2$， $\delta_B = 0.01$

带有 $\beta = 0.1$ 的圆形分布

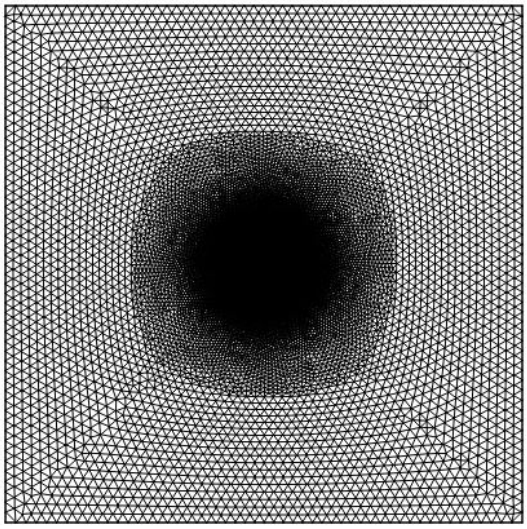


图 74： $\delta_A = 0.2$， $\delta_B = 0.01$

条带分布

This 分布会根据用户指定的倾角，从一条直线生成增长网格。

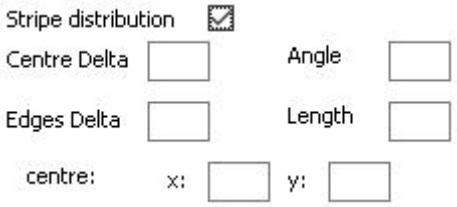


图 75：条带分布截面

此分布使用的输入包括：

- δ_A：“中心 Delta”。这是圆形分布中心处的值 .
- δ_B：“边缘增量”。这是网格达到的最大值。
- α：“角度”。它定义了紧凑条带的角度
- L：“长度”。设置网格更密集的区域大小。
- X_C：“中心：x”。设置分布中心的 x 坐标。
- Y_C：“中心：y”。设置分布中心的 y 坐标。

这个函数比之前的函数更复杂。其原理是：随着距离用户定义的直线 (x”) 越来越远，间距增量会线性增长。为实现这一点，需要按照图76进行坐标轴变换。

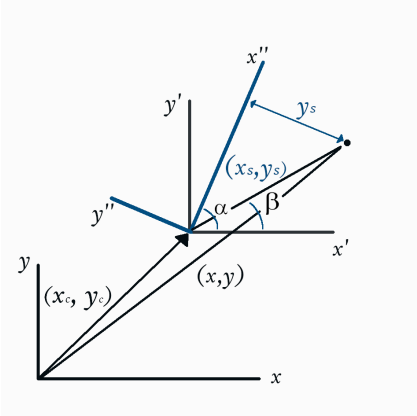


图76：条带分布中使用的轴

控制此分布的方程 is:

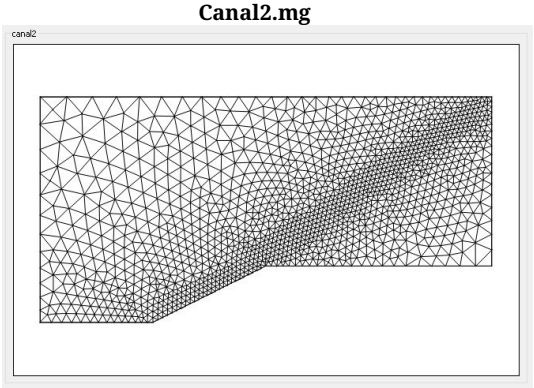
$$\delta = \delta_A + \delta_B * \frac{y_s}{L} \tag{9}$$

为 y_s:

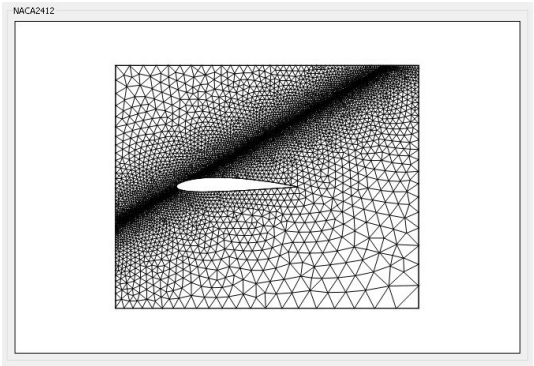
$$y_s = /\sqrt{(x - x_c)^2 + (y - y_c)^2} * sin(\beta - \alpha)/ \tag{10}$$

这种间距增量大小分布是线性的，因此不是顶部界定的，最小值为 δ_A。以下是这种分布的两个示例:

Canal2.mg



NACA 2412.mg



另一种适用于本程序的良好分布是锥形，可用于研究燃烧室内产生的冲击波。

7 TRIMPACK

TRIM-Pack（三**角形非结构网**格生成**程序包**）是一个包含单元（FORTRAN 子程序）的库，用于生成和细化三角形非结构网格。它还包含用于验证和评估这些现有网格的单元。

在设计子程序和数据结构时，我们给予了特别关注。这使得后续扩展成为可能，例如加入新的适应和细化子程序，从而实现高度的使用灵活性。

本节将详细介绍并评估所选的网格生成算法。这些算法与一些必要工具共同构成了三角形非结构网格生成软件包 TRIMPACK。

7.1 TRIMPACK 的内容

7.1.1 生成算法

7.1.1.1 dlny 此子程序为一组节点及其对应的边界实现约束Delaunay三角剖分，但也可以进行无边界三角剖分。

如果给定边界，dlny 子程序会借助边穿越算法来检查例程决策的正确性和可行性。考虑到边界可能具有的复杂性，这一步是必要的。

更详细的说明请参阅 7.2.2

7.1.1.2 afm 此子程序使用推进前沿法 (AFM)，对给定边界之间的区域进行三角剖分。

7.1.1.3 steiner Steiner 使用 Steiner算法生成非结构化三角网格。完成 Delaunay 三角剖分后，此子程序会插入节点、交换边并平滑网格，以提高网格质量。

7.1.2 网格细化算法

7.1.2.1 nodins Nodins 会在已生成网格的三角形中添加新节点，以细化某些部分或整个网格。目前，新节点将插入每个三角形的重心处，但后续扩展很容易实现。

7.1.2.2 edswap 此子程序按照 Delaunay 判据或最小-最大判据执行对角线交换，在不改变给定节点位置的情况下更改网格中节点之间的连接关系。Edswap 假定已存在一个三角剖分（但并非 Delaunay 三角剖分），随后通过边交换将其转换为 Delaunay 三角剖分。

7.1.2.3 smooth 子程序 smooth 会在不改变网格拓扑的情况下调整内部（自由）节点的位置，以改善网格质量。它将每一对相连的节点（只要它们不是边界节点）视为具有相同弹性和基准长度的弹簧。平衡位置即为 smooth 的结果。

7.1.3 工具

7.1.3.1 cbrdr 该子程序使用样条（样条子程序）创建边界，以生成边界节点分布；用户通过边界定义和节点间距子程序提供这些内容。如果待三角剖分的区域没有边界，cbrdr 将创建一条初始边，作为三角剖分的起始边。因此，两个子程序（fnode 和 snode）分别提供定义这条初始边的两个节点。在这种情况下——即未给定边界时——用户必须定义一些自由节点。Cbrdr 可用于创建 Delaunay 三角剖分的边界。

7.1.3.2 connec connec 子程序可选择生成给定三角剖分的以下八个连接矩阵，以供后续计算使用：

- 三角形是三角形的函数。
- 节点是节点的函数。
- 三角形是节点的函数。
- 节点是三角形的函数。
- 边是节点的函数。
- 以边为自变量的节点。
- 以边为自变量的三角形。
- 以三角形为自变量的边。

7.1.3.3 edcros Edcros 提供一个指示器，用于警告两条边发生交叉。

7.1.3.4 spline 通过三次样条，提供子程序 cbrdr，以获得最终的边界节点分布。

7.1.4 其他子程序

7.1.4.1 stmsh（本程序未使用）： 此子程序提供有关现有网格的一些统计信息：

- 节点-节点连接数。
- 三角形-节点连接数。
- 三角形面积占网格三角形面积的百分比。
- 边长占网格边长的百分比。
- 三角形角度占 60o 的百分比。
- 此外，stmsh 会检查一般公式：Nu

7.1.4.2 中心 提供由三个节点定义的圆的中心坐标。

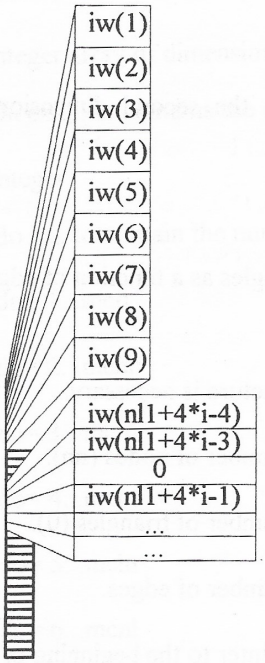
7.1.4.3 fnode 在给定的节点云中查找 y 坐标最小的节点，并将其编号返回给子程序 brdr

7.1.4.4 snode 查找第二个初始节点，该节点与第一个节点共同构成三角剖分的初始边。这个第二节点是与水平线形成最小角度的节点。这样可以相当确定第一条边的位置适合后续的三角剖分。

7.2 描述

7.2.1 数据结构

在整个库中，所有网格信息都保存在一个名为 iw 的数组中，该数组具有以下结构：



iw(1)	Version (ido).
iw(2)	Number of nodes (nn).
iw(3)	Number of triangles (0).
iw(4)	Number of edges.
iw(5)	Length of iw array (liw).
iw(6)	Number of boundary segments (nc).
iw(7)	Number of boundary nodes.
iw(8)	Pointer to the beginning of the table (nl1).
iw(9)	Pointer to the end of the table.
iw(nl1+4*i-4)	First node of edge i.
iw(nl1+4*i-3)	Second node of edge i.
iw(nl1+4*i-2)	Number of triangle 'in front' of edge i (0, because there are no triangles).
iw(nl1+4*i-1)	Number of triangle 'behind' of edge i (number of boundary segment).

图77: *iwstructure*

此外，主代码中还通过参数传入了三个定义为外部函数的子程序。这种结构使用户能够创建或自定义这些子程序的功能。

drw drw子程序可让用户清除屏幕，或在计算网格的同时绘制网格。此子程序的参数如下。

call drw(idrw,x1,x2,y1,y2)

此子程序除屏幕，绘制 在节点1和节点2之间绘制一条直线。整数 *idrw* 的值决定是否可以清除边。

spc 此子程序提供用于控制节点插入的间距因子。用户必须在此为网格中每个三角形的中心指定 delta 的值。该值可以是常数，也可以按照用户所需的方式变化。其规范如下：

call spc(x,y,delta)

其中，x 和 y 确定区域中的点，而在此子程序中通过公式计算得出的 delta 可以是任意值。

bnd bnd 为子程序 *brdr* **(7.2.4.1)** 提供三角剖分域的外部/内部边界。用户必须在此子程序中指定线段的描述节点数量及其 **xb** 和 **yb** 坐标，并指定下一个相连的边界段 **(nj)**（如果存在连接）。

边界段（因此，每条边界边）必须按方向排序，这一点非常重要：从第一个节点到第二个节点遍历线段时，三角剖分域始终位于边界边的**左侧**。其规范如下：

call bnd(in,nb,xb,yb,nj)

变量	输入	输出
in - 整数	边界段数量	
nb - 整数	-	描述边界节点的总数
nj - 整数	空	下一个连接边界段的编号
xb - 实数数组 (nbmax)	所有边界节点的 x 坐标	
yb - 实数数组 (nbmax)	所有边界节点的 y 坐标	

7.2.2 生成算法

DLNY: 该算法采用的是基本 Delaunay 三角剖分算法，其性质和主要思想已在第 2.3.1 章中介绍。

To launch dlny s在调用程序时，用户必须提供一些关于所需区域的先验信息 待进行三角剖分的对象以及 gree 节点（如果存在）：

call dlny(nn,nl,nt,x,y,drw,iw,liw,ido,ifail)

参数值

变量	输入	输出
nn - 整数	区域中的节点数（边界 +现有自由节点）	
nl - 整数	边界边数	所有网格边的数量
nt - 整数	空	网格三角形数
x - 实数数组 (nn)	区域内所有节点（边界节点和自由节点）的 x 坐标	
y - 实数数组 (nn)	区域内所有节点（边界节点和自由节点）的 y 坐标	
drw - 子程序	drw (idrw,x1,x2,y1,y2)	
iw - 整数数组 (liw)	边界结构	完整网格结构
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 5 *$ 最大边数 (nl) $+ 9$	
ido - 整数	版本号	
ifail - 整数	0	错误消息

Ido 值

值	规范
0	无边界
1	有边界
2	由样条构造的有边界

对于 $ido = 1$，2，将执行边交叉检查，因为边界可能具有复杂结构，因此有必要进行此项检查。

Ifail 值

值	规范
1	用户必须检查最终网格，查找错误
0	正常退出
-1	输入数据不正确，可能是 liw 过小

代码说明 主循环开始时， nsk 给出边界边的数量，整数工作数组 iw 包含节点数和三角形数与边的关系（$iw(nl1...nl2-1)$，以及每条边的索引（0：活动，1：非活动）（iw(nl2)）。循环将持续执行，直到 nsk 为 0，即活动前沿为空且所有索引均为 1。选取栈（活动边）的第一条边（节点 $n1$ 和 $n2$），并验证第一个候选节点（$inodo$）是否适合构成 Delaunay 三角形。

因此，需要计算边的直线函数（即穿过边的两个节点的直线），并检查候选节点（$inodo$）是否位于边的正确一侧，从而成为可能的 Delaunay 节点。如果候选节点（$inodo$）未处于正确位置，则选择另一个（下一个）候选节点（$inodo$）。

确认节点（$inodo$）处于正确位置后，便计算临时三角形（$n1$、 $n2$ 和 $inodo$），即经过这三个节点的圆的外心和半径。随后，子程序检查是否有其他节点（$jnodo$）位于所述圆内。如果例程发现圆内有节点，就会舍弃临时三角形，并选取圆内的节点作为下一个候选节点（$inodo$）。

如果临时三角形的圆内没有其他节点，例程就会通过子程序$edcros$ (边交叉)检查两条新边（边 $n1-nodo$ 和边 $inodo-n2$，请记住方向）是否与三角剖分中其他已有的边相交。如果存在边交叉，则不更新临时三角形；否则，该三角形就是 Delaunay 三角形。最后，例程会检查新边是否已经存在，然后更新工作数组（iw），写入新数据（向 iw 添加一条新边，以及与之关联的两个节点 $n1$ 和 $n2$ 及其所属三角形），并从栈中取出下一条活动边作为底边，继续搜索第三个节点以构建 Delaunay 三角形。

在此例程结束时，整数工作数组（iw）会重新组织，现在包含了 7.2.1 中所述的信息

AFM：此处用于推进前沿法的算法基于 Peraire 等人提出的方法，首次发表于 [24]（参见第 2.3.3 节）。

要启动 afm 子程序，用户必须确保正确提供有关区域的必要信息。

$call$ afm(nn,nl,nt,x,y,lnn,drw,iw,liw,ido,ifail)

参数值

变量	输入	输出
nn - 整数	边界节点总数	网格节点总数
nl - 整数	边界边数量	网格边总数
nt - 整数	空	网格三角形数
x - 实数数组 (nn)	边界点的 x 坐标	所有网格点的 x 坐标
y - 实数数组（nn）	边界点的 u 坐标	所有网格点的 y 坐标
lnn - 整数	x 和 y 维度（估计节点数）	
drw - 子程序	drw (idrw,x1,x2,y1,y2)	
iw - 整数数组 (liw)	边界结构	完整网格结构
liw - 整数	drw（iw 数组的维度，至少为 $liw \geq 5*$最大边数（nl）$+9$	
ido - 整数	版本号	
ifail - 整数	0	错误消息

Ido 值

值	规范
1	栈中的第一个活动边用于进行三角剖分（建议使用恒定间距因子）
2	栈中的最小边用于进行三角剖分（建议使用变化间距因子）

Ifail 值

值	规范
0	正常退出
-1	输入数据不正确，可能是 liw 过小
-7	“边交叉栈”过小
-8	近邻节点列表的长度过小

代码说明 在完成一些前期计算（例如变量初始化，以及计算可容纳在 iniw 中的最大可能边数，供后续错误检查使用）后，afm 子程序开始进入主循环。

首先，将搜索栈 iw 中最小的边，或 x 和 y 维度（估计节点数）的第一个活动边，作为第一个三角形的底边。该搜索例程取决于 ido 的值（见上文）。对于这条边，将迭代“完美”三角形的顶点（称为 $P1$；该三角形的底边为 $n1-n2$，顶点与 $n1$ 和 $n2$ 的距离为 $delta1$ ）：在这次迭代中，将一个三角形的高（此处称为 $cheight$）与其所属等边三角形的重心 $spacingfactor$ 构造出的三角形的高进行比较（见图 78）。

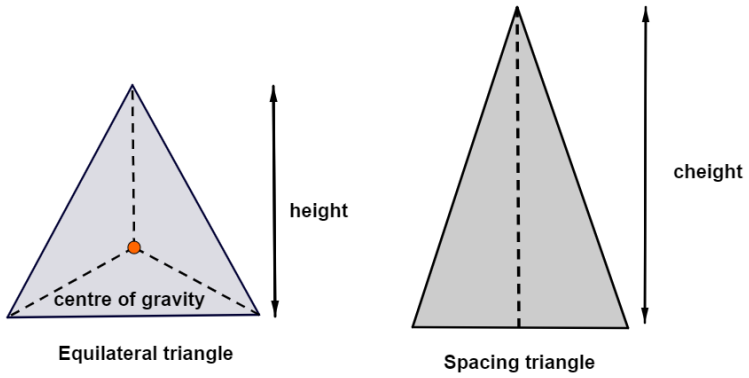


图 78：新三角形的构造

如果这两个高相差超过 1%，则取这个间距三角形的中心，以获得新的间距因子，从而得到新的间距三角形。

间距因子由间距子程序提供，在此处取代了 Peraire 建议的背景网格。这样，提供“背景网格信息”就变得简单而高效。在此间距子程序中，用户必须为每个 x 和 y 值定义一个因子，该因子应大致表示该节点的“平均距离”。不过，也可以轻松实现真正的“背景网格”，以增加搜索算法的计算时间。

如果迭代得到正确的高，并计算出 δ_M ($deltam$)（即间距三角形的边长）的值，就可以根据 Peraire 等人的不等式（参见第 2.3.3 章）得到 δ_1 ($delta1$) 的值。现在， $P1$ 的位置就是该三角形的顶点位置，其底边为 $n1-n2$，边长为 δ_1。

计算完 $P1$ 后，该例程会确定所有位于以 $P1$ 为中心、 δ_1 为半径的圆内的活动节点。如果找到一个活动节点（活动边的第一个节点），且该节点位于基边的左侧（沿边从 $n1$ 到 $n2$ 遍历时的左侧），那么只要它与 $n1$ 和 $n2$ 的距离合适（至多为 $1.5*\delta_1$ ），就可以作为候选连接节点。所有满足这些条件的节点都会按照它们到基边的距离排序并列入列表，而 $P1$ 则添加在列表末尾。

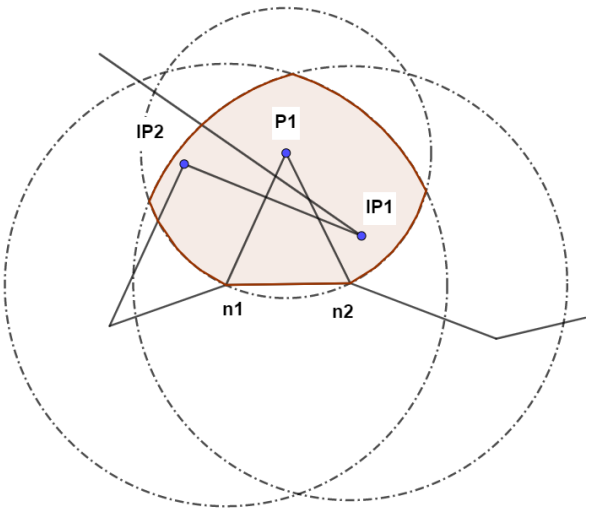


图79：要连接的点的面积（阴影部分）

在主循环中，afm例程会检查列表中第一个节点所构成的边是否与前沿中的任何现有边相交。如果发生边交叉，例程就会选取列表中的下一个节点，并重复进行边交叉控制。如果列表中的最后一个节点也导致边交叉，则此时不会以该底边进行三角剖分。这是该算法与 Peraire 提出的算法的主要区别：在他的算法中，如果列表中的某个节点非常靠近一条现有边，就可能产生狭长且偏斜的三角形（见图80）；此外，对于列表中所有节点都会发生边交叉的情况，也未能得到充分解决（见图81）。

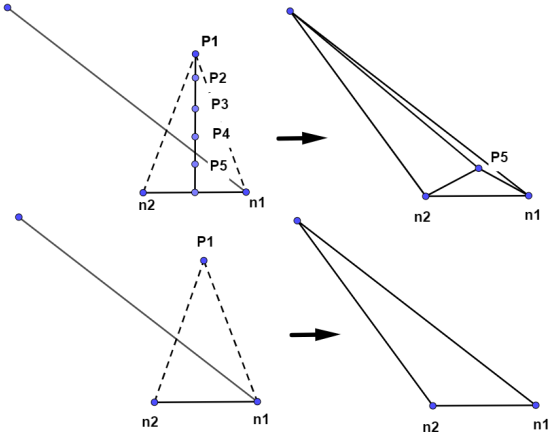


图80：（a）Peraire、（b）所用算法的三角剖分示例

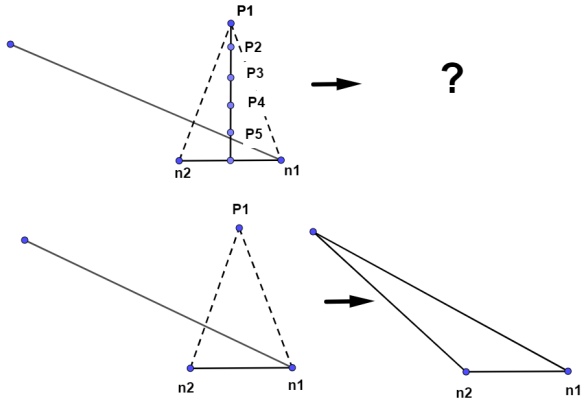


图81：Peraire算法中列表最后一个点导致的边交叉问题（a）及所用算法对此的解决方案（b）

AFM将前沿中下一条较长的边作为基边，并重复前面所述的算法。为此，需要一个新的数组 $cros$ 来保留这条边，以便搜索下一条较长的边。如果该数组的维度（dim）过小（边界几何形状非常复杂时会出现这种情况），例程将返回ifail =-7给主程序。

如果最终能够构造出一个三角形，基边就会变为非活动状态（其在 iw 相应部分中的值变为1），并获得前方下一个三角形的编号。对于这个新三角形，例程会检查其新边是否已经存在（如果生成了新节点），并更新栈。在此处，用户可以按自己的意愿绘制新边（见7.2.1）。

AFM会重复此循环，直到活动边的数量（isk）为 0，且无法继续进行三角剖分。在此例程结束时，整数工作数组（iw）会重新组织，现在包含 7.2.1 中所述的信息。

STEINER： 此处使用的算法与 2.3.4 节中提到的基本 Holmes算法类似。

要启动Steiner子程序，用户必须确保正确提供有关区域的必要信息。

$call$ steiner(nn,nl,nt,x,y,lnn,spc,drw,iw,liw,因子,ido,ifail)

参数值

变量	输入	输出
nn - 整数	自由节点 + 边界节点	自由节点+ 边界节点 + 插入节点
nl - 整数	边界边数	所有网格边的数量
nt - 整数	空	网格三角形数
x - 实数数组 (nn)	区域中所有节点（边界节点和自由节点）的 x 坐标	
y - 实数数组（nn）	区域中所有节点（边界节点和自由节点）的 y 坐标	
lnn - 整数	x 和 y 维度（估计节点数）	
spc - 子程序 7.2.1	spc (x,y,delta)	
drw - 子程序 7.2.1	drw (idrw,x1,x2,y1,y2)	
iw - 整数数组 (liw) 7.2.1	边界结构	完整网格结构
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 5 * 最大边数\ (nl) + 9$	
factor - 实数	控制三角形大小	
ido - 整数	版本号和特定的最终平滑机制 7.2.2	
ifail - 整数	0	错误消息

因子值 此因子可以是 1 到 3 之间的任意数值。它是 Steiner 方法中子程序$nodins$ call 的尺寸因子。

值	规范
1	小于指定 spc 的三角形（边长为 delta 的三角形）
3	所有大于增量三角形的三角形

Ido 值

位置	值	规范
ido(1)	1	有边界
	2	有边界，由样条构造
ido(1)	1	三角形重心处的插图
	2	尚不存在
	3	尚不存在
ido(3)	0	无平滑处理
	1	带平滑
ido(4)	0	无最终边交叉
	1	有最终边交叉

Ifail 值

值	规范
1	用户必须检查最终网格，查找错误
0	正常退出
-1	输入数据不正确，可能是 liw 过小
-2	预期节点数（lnn）过少
-3	边界段中的节点过多
-4	传入的值 nn , nt 或 nl 不正确
-5	传入的连接矩阵维度（列， nmc）过小
-6	传入的连接矩阵维度（行， lmc）过小
-7	最终网格中存在边交叉

代码说明 此子程序包含对 $dlny$、$nodins$、$edswap$ 和 $smooth$ 的调用，并且每次调用后都会进行 ifail 错误检查。

无论 ifail 小于零还是没有发生节点插入，Steiner 都会返回主程序，因此无需进一步细化网格。返回主程序前，还会进行一次边交叉控制以验证网格。如果此处 ifail 为正，例程确实会在屏幕上打印该值，但也会继续运行。

7.2.3 网格细化算法

7.2.3.1 Nodins：这里采用的算法实现了后验节点插入（见第 2.3.1 章），可用于后验网格细化（重新网格划分）。这意味着必须已有一个三角剖分，但其构建方式并不重要。

除了网格信息外，用户还必须传递一些其他信息才能调用此子程序：

$call$ nodins(x,y,nn,lnn,nt,nl,iw,liw,spc,drw,factor,ido,ifail)

参数值

变量	输入	输出
x - 实数数组 (lnn)	区域中所有节点（边界和自由节点）的 x 坐标	
y - 实数数组（lnn）	区域中所有节点的 y 坐标（边界节点和自由节点）	
nn - 整数	区域中的节点数	区域中的更新后节点数
lnn - 整数	必须指定估计的最终 x 和 y 尺寸（nn + 所有新插入的节点）	
nt - 整数	网格三角形数	更新后的网格三角形数
nl - 整数	网格边数	更新后的网格边数量
iw - 整数数组（liw）	边界结构	完整网格结构
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 9 + 4 * \text{nl}$[入口时] $+ 7 * \text{nt}$ [入口时]	
drw - 子程序	drw (idrw,x1,x2,y1,y2)	
spc - 子程序 7.2.1	spc (x,y,delta)	
factor - 实数	控制三角形大小	
ido - 整数	版本号和特定的最终平滑机制 7.2.2	
ifail - 整数	0	错误消息

Ido 值 在此子程序中， ido 值用于设置待插入节点的位置。

值	规范
1	在重心处
2	在外接圆的中心
3	与顶点的距离为 delta（由 spcg 提供）

Ifail 值

值	规范
1	没有空间（在 lw、 x 或 y中）插入节点
0	正常退出
-1	提供的 liw 值太小，无法启动子程序
-4	提供的 nn 值 , nt 或 nl 不正确

代码说明 要正确执行此子程序，整数工作数组（iw）必须具有特定结构和最小长度（liw）。该长度由 connec 子程序创建的两个连接表的长度、对后续计算至关重要的辅助部分的长度，以及上述 iw 部分的长度共同确定。如果 x/y 数组（lnn）或 iw （liw）的长度太小（无法启动或插入所有节点），例程将返回 ifail。子程序还会在创建两个连接表 $mcnt$ 和 $mclt$ 并将其放置在数组 iw末尾之前，对提供的数据执行错误检查。随后开始遍历所有旧三角形，计算当前三角形的中心，并检查是否需要插入节点。因此，外部 spc 会提供一个判据，用于与当前三角形的面积进行比较。如果需要插入节点，例程将确定属于该三角形的每条边的编号和位置，将计算出的节点添加到坐标表 x 和 y 中，并将新的三条边添加到 $mcnl$的末尾。对于待插入节点的位置，用户可以从三个版本中选择（参见 ido 的定义和图 82）。

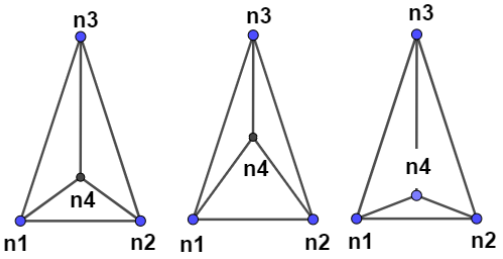


图 82：节点插入位置：a）重心 b）外接圆圆心 c）与 n1、n2 或 n3 的间距

以下描述适用于 $ido = 1$ ，因为另外两个版本尚不存在：新生成的三个三角形插入 $mctl$ 时，其中一个新三角形沿用旧三角形的编号和位置，

而另外两个新三角形则添加到 $mctl$ 末尾的 iw中。此处，用户可以选择是否绘制新边（见 7.2.1）。对 $mctl$ 进行必要的修改后（必须更改相邻三角形），例程会处理下一个旧三角形。子程序即将返回主程序之前，会用新的节点数、三角形数和边数更新 iw信息部分，并将指针指向表格末尾。

7.2.3.2 Edswap：该算法执行对角边交换（见第 2.3.1 章），除了用于网格细化外，也可用于网格生成算法（Lawson，见第 2.3.1 章）。无论哪种情况，都必须已有一个网格，但其中某些部分不符合 Delaunay 三角剖分判据。调用此子程序时，除了网格结构外，还需要提供一些其他信息，以便正确启动：

$$call\ \text{edswap(x,y,nn,nt,nl,iw,liw,drw,ifail)}$$

参数值

变量	输入	输出
x - 实数数组 (lnn)	区域中所有节点（边界节点和自由节点）的 x 坐标	
y - 实数数组 (lnn)	区域中所有节点（边界节点和自由节点）的 y 坐标	
nn - 整数	区域中的节点数	
nt - 整数	网格三角形数	
nl - 整数	网格边数	
iw - 整数数组 (liw) 7.2.1	边界结构	完整网格结构
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 9 + 4 * \text{nl}$[入口处] $+ 7 * \text{nt}$ [入口处]	
drw - 子程序 7.2.1	drw（idrw,x1,x2,y1,y2）	
ifail - 整数	0	错误消息

Ifail 值

值	规范
0	正常退出
-1	提供的 liw 值太小，无法启动子程序
-4	nn的已传递值 , nt 或 nl 不正确

代码说明 要正确执行此子程序，整数工作数组（iw）必须具有特定的结构和最小长度（liw）。该长度由将在 connec 子程序中创建的两个连接表的长度、对后续计算十分重要的辅助部分的长度，以及 iw的各个部分共同定义，后者已在前文提及。

该子程序首先为 iw 工作数组初始化指针，并从 iw中提取信息。如果 iw （liw）的长度过小，例程将返回一个 $ifail$。在创建两个连接表 $mcnt$ 和 $mclt$之前，该子程序还会对已传递的数据执行错误检查。

将这两个连接表放置在 iw 末尾后，算法便开始检查所有四边形，判断其现有对角线是否为边界边。如果不是，则确定该四边形四个节点的编号和位置，并计算对角线“上方”和“下方”的角度。（关于节点、三角形、边和角度的位置，请参见图 83。）如果这两个角度之和大于新构型中对应角度之和，也就是说，旧构型中的角度之和大于 180°，则执行边交换。此判据还可排除非凸四边形，这对于避免边交叉是必要的。

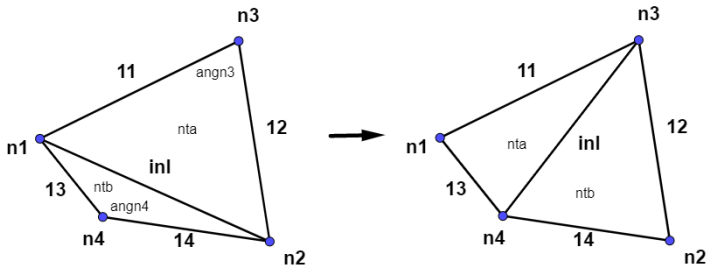


图 83：变量定义

确定该四边形中各边的编号和位置后，例程会更改 iw 中包含的四个连接表，并重复执行算法，直到检查完每条边对应的四边形。该例程结束时，用户可以选择按自己的意愿绘制新网格（参见 7.2.1 子程序），然后返回主程序。

7.2.3.3 平滑： 此子程序根据弹簧模型 实现后验网格平滑，可用于网格自适应的重新网格划分（见第 2.3.1 章）。无论如何，都必须存在一个初始网格，但其创建方式并不重要。

调用此子程序时，除了网格结构外，还需要一些其他信息才能正确启动：

$call$ smooth(x,y,nn,lmc,iw,liw,drw,ifail)

参数值

变量	输入	输出
x - 实数数组 (lnn)	区域内所有节点的 x 坐标（边界节点和自由节点）	
y - 实数数组 (lnn)	区域内所有节点的 y 坐标（边界节点和自由节点）	
nn - 整数	区域中的节点数	
lmc - 整数	网格中的一个节点可以拥有的伙伴节点的最大数量	
iw - 整数数组 (liw) 7.2.1	边界结构	完整网格结构
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 9 + 4 * \text{nl}[\text{on entry}] + 7 * \text{nt}\ [\text{on entry}]$	
drw - 子程序 7.2.1	drw (idrw,x1,x2,y1,y2)	
ifail - 整数	0	错误消息

Ifail 值

值	规范
1	无自由节点
0	正常退出
-1	提供的 liw 值太小，无法启动子程序
-4	提供的值为 nn , nt 或 nl 不正确

代码说明 要正确执行此子程序，整数工作数组 (iw) 必须具有特定结构和最小长度 (liw)。该长度由连接表的长度（将在 connec 子程序中创建）以及上文提到的 iw 部分定义。

如果 iw (liw) 的长度太小、提供的总节点数不正确，或者没有可移动的自由节点，子程序将带有一个 $ifail$ 返回主程序。完成此错误检查并在 iw 工作数组中创建 $mcnn$ 矩阵后，该例程会计算区域内节点的平均二次距离 ($avdist$)，此值随后将用作移动准则。

在遍历区域内所有**自由节点**的循环中，该例程会查找最需要移动的节点，计算其新坐标并更改 x/y 坐标表。

为计算节点的新坐标，使用 $springmodel$，即将与该节点相连的每条边视为刚度**相等** 的弹簧。通过迭代移动节点，直至弹簧系统达到平衡（移动量小于判据）。

在该算法中，弹簧的刚度相等，但也可以根据局部间距因子设置刚度，而且实现起来相当简单。采用局部不同的刚度能让用户更好地控制三角形和网格结构，但会显著降低子程序的执行速度。

事实上，在少数情况下，如果判据设置得过大，平滑可能会引发边交叉（见图84），因此随后应执行边交叉控制。

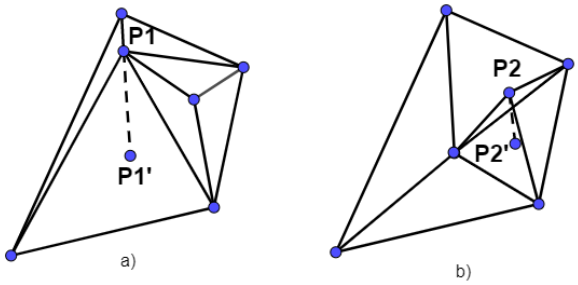


图84：平滑后可能出现的边交叉：a) 距离 P1-P1’ 大于判据，b) 距离 P2-P2’ 不

在返回主程序之前，子程序提供了按用户意愿绘制修改后网格的选项（见7.2.1 子程序）：

7.2.4 工具

7.2.4.1 Brdr：调用 brdr 子程序如下：

$call$ brdr(nn,nl,mb,nc,x,y,lnn,bnd,spc,drw,w,lw,iw,liw,ido,ifail)

参数值

变量	输入	输出
nn - 整数	边界节点数	边界节点总数（旧 + 新）
nl - 整数	边界边的数量	边界边总数（旧 +新）
mb - 整数数组 (nc)	空	作为线段函数的 nn
nc - 整数	边界节点数	
x - 实数数组 (nn)	边界点的 x 坐标	所有边界点的 x 坐标
y - 实数数组 (nn)	边界点的 u 坐标	所有边界点的 y 坐标
lnn - 整数	x 和 y 维度（估计值（nn [入口时] +新增节点））	
bnd - 子程序 7.2.1	bnd（in,nb,xb,yb,nj）	
spc - 子程序 7.2.1	spc（x,y,delta）	
drw - 子程序 7.2.1	drw（idrw,x1,x2,y1,y2）	
w - 实数数组（lw）	用作实数工作空间	
lw - 整数	w 维度 lw $\geq 4*$ 每段最大节点数	
iw - 整数数组 (liw) 7.2.1	边界结构	
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 5 * 最大边数\ (nl) + 9$	
ido - 整数	版本号	
ifail - 整数	0	错误消息

Ido 值 在此子程序中， ido 值包含所需 dlny 三角剖分的版本号。

值	规范
0	无边界
1	具有用户定义的边界
2	具有用户定义的边界，并使用样条插值

Ifail 值

值	规范
0	正常退出
-1	输入数据不正确，liw 过小
-2	预期的节点数（lnn）过少
-3	边界段中的节点过多

代码说明 经过前面的计算和初始化后， ido 的值决定将执行哪种边界分布。

对于 $ido = 0$，用户希望对一个开域进行三角剖分，而开始三角剖分需要一条初始边。这条初始边由第一个节点（由子程序 $fnode$ 提供）和第二个节点（由 $snode$提供）组成。这条初始边被放入 iw 数组中，并更新 iw 数组中用于存储信息部分的变量。

对于 $ido = 1$ 或 $ido = 2$，该例程在遍历所有边界段的循环中接收边界节点的数量和坐标，以及下一个连接边界段的编号。如果 $ido = 2$，则调用样条子程序（spl），该子程序返回当前边界段中所有边界节点的成员信息和坐标。该例程将此信息放入栈（iw ）和 x、y 坐标表中，同时使用一个实数工作数组（w）存储该线段所有边界节点（xb、yb）的坐标。 w 是一个实数工作数组，用于存储：

- w(1... nw2)：定义线段的节点的 x 坐标，由 bnd 子程序提供。
- w(nw2+1...n w3)：定义线段的节点的 y 坐标，由 bnd 子程 e.
- w(nw3+1...nw4)：线段边界节点的 x 坐标。
- w(nw4+1...)：线段边界节点的 y 坐标。

插入边界节点后，其坐标会传递给外部 dew，以便用户查看这些信息（例如，在屏幕或打印机上绘制边界）。

在.....的末尾 此例程结束时，整数工作数组 iw 会重新组织，并包含边界信息 n.

7.2.4.2 Connec: TRIM-Pack 的许多例程需要有关网格及其连接关系的更多信息。因此，已将子程序 connec 建立为独立算法。

要启动此子程序 , 必须先有一个网格，并传入其结构：

$$call\ \text{connec(lmc,nmc,iw,liw,mc,ido,ifail)}$$

参数值

变量	输入	输出
LMC - 整数	mc 的行数	
NMC - 整数	mc 的列数	
iw - 整数数组 (liw)	网格结构	
liw - 整数	drw（数组 iw 的维度，至少为 $liw \geq 5 *$ 最大边数 (nl) $+ 9$	
mc - 整数数组 (lmc,nmc)	包含所需的连接矩阵	
ido - 整数	要创建的目标矩阵的编号	
ifail - 整数	0	错误消息

Ido 值 在此子程序中， ido 值包含要创建的目标矩阵。

值	规范
1	mctt
2	mcnn
3	mctn
4	mcnt
5	mcln
6	mcnl
7	mctl
8	mclt

Ifail 值

值	规范
0	正常退出
-1	liw 过小
-5	NMC 太小
-6	LMC 太小

代码说明 在初始化一些变量、从 iw 的信息部分提取必要信息并对 iw 的最小长度进行错误检查后， ido 的值决定将计算哪个连接表：

• mctt (3,nt) [ido $= 1$]: 节点-三角形连通矩阵。完成维度的例行检查后，遍历数组中的所有边 iw ，并在 mc(mi/m2,nt1/nt2) 中开始存储三个“邻接”三角形，其中 m1/m2 表示当前“邻接”三角形的数量，nt1/nt2 表示当前“中心三角形”。此处， iw 中的辅助空间是必要的，其中包含“邻接”三角形计数器（n1/m2），该计数器表示将三角形放入 mc的位置。

• mcnn (?,nn) [ido $= 2$]: 节点-节点连通矩阵。完成 **nmc** 的维度检查后，例程初始化 mc 的第一行，该行将用作配对节点的计数器，同时也用作将每个配对节点放入 mc的位置。然后，遍历 iw 中的所有边，将每个节点（n1/n2）与其配对节点（n2/n1）关联起来。每添加一个节点，例程都会检查 mc 行的维度（lmc）。

• mctn (?,tn) [ido $= 3$]: 三角形-节点连通矩阵。完成对**NMC**的维度检查后，例程初始化 mc 的第一行，该行将用作共享该节点的三角形计数器，同时也用作位置。例程取两个相连节点及其两个相邻三角形，再通过另一个循环检查其中一个或两个三角形是否已存在于 mc(x,n1/n2) 中；如有必要，最后将它们插入 mc ，并更新 mc(1,n1/n2) 中的“满单元”计数器。每次添加节点时，例程都会检查 mc 的行数（lmc）。

• mcnt (?,nt) [ido $= 4$]: 节点-三角形连通矩阵。检查完两个维度后，开始遍历所有边的循环 iw ，取两个相连节点及其两个相邻三角形，再通过另一个循环检查其中一个或两个节点是否已存在于 mc(x,nt1/nt2) 中；随后更改 iw(ihlp+nt1/nt2) 中辅助空间的值。该空间用作节点计数器，因此也用作位置提示，以确定应将节点编号放入 mc 的位置；最后将节点编号插入 mc。

• mcln (?,ln) [ido $= 5$]: 边-节点连通矩阵。检查完 nmc的维度后，例程会初始化 mc 的第一行，该行将用作共享该节点的边的计数器，同时也用作位置标记，指示应将每条边放在 mc中的位置。随后开始遍历所有边的循环，在 mc(x,n1/n2) 中插入值，同时更新位置标记并检查行数（lmc）的维度。

• mcnl (?,nl) [ido $= 6$]: 节点-边连通矩阵。检查完两个维度后，开始遍历 iw 中的所有边，并将 iw 的值插入 mc中。由于 iw 中已存在此表，因此这个例程相对简单。

• mctl (?,tl) [ido $= 7$]: 三角形-边连通矩阵。检查完两个维度后，开始遍历 iw 中的所有边，并将 iw 的值插入 mc中。由于 iw中已存在此表，因此这个例程相对简单。

• mclt (?,lt) [ido $= 8$]: 边-三角形连通矩阵。检查完两个维度后，开始遍历 iw 中的所有边，将每个编号大于 0 的三角形编号与其边关联起来，而 iw 中的辅助空间则用作计数器，因此也用作 mc的位置标记。

7.2.4.3 Edcross： 此子程序是 dlny 和 afm 子程序的必要组成部分，也可用于检查最终网格结构（例如，执行平滑之后）。调用方式如下：

$call$ edcros(xa,ya,xb,yb,x1,y1,x2,y2,指示变量)

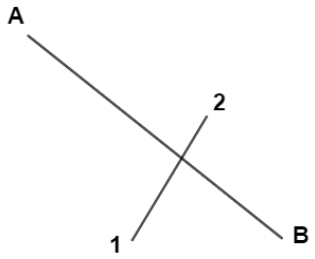


图85：edcros方案

参数值

变量	输入	输出
xa,ya - 实数	节点A的坐标（边A-B）	
xb,yb - 实数	节点B的坐标（边A-B）	
x1,y1 - 实数	节点1的坐标（边1-2）	
x2、y2 - 实数	节点2的坐标（边1-2）	
指示变量 - 整数		边交叉指示器

指示变量值 .

值	规范
0	边A-B与边1-2之间没有边交叉
1	存在边交叉

代码说明 edcros 子程序用于检查边A-B与边1-2之间是否可能发生交叉。因此，程序会进行坐标变换，将坐标转换到 A-B 坐标系中，从而简化控制例程。根据边1-2相对于边A-B的位置，edcros 例程分为三种不同情况：垂直、水平线以及其他情况。如果发生边交叉，edcros 子程序会向主程序返回一个值为 1 的指示器（indcr）。如果没有边交叉，indcr 的值将为 0。

在此子程序中定义了一个特殊参数 zero。zero 是一个非常小的正数，大于机器的舍入误差。zero 和 -zero 定义了一个围绕实数 0 值的带状区域。之所以必须采取这一措施，是因为在 FORTRAN 语言中对实数进行减法运算时会产生误差。

7.2.4.4 Spline: 节点插入子程序。

此处该子程序仅用于边界节点分布，但也可以用于其他用途，例如

调用如下：

$$call\ \mathrm{spl(n,x,y,m,xx,yy,spc)}$$

参数值

变量	输入	输出
n - 整数	实际边界段的边界节点数量	
x,y - 实数数组（n）	描述边界节点的坐标	
m - 整数		创建的最终边界节点数
xx,yy - 实数数组		最终创建的边界节点的坐标
spc - 子程序 7.2.1	spc (x,y,delta)（必须声明为外部）	

7.2.5 其他子程序

7.2.5.1 Stmsh： 生成网格后，可使用此子程序评估网格质量，或获取是否需要重新网格划分的判据。

显然，因此这个子程序需要完整的网格信息，并且需要 connec 子程序的支持：

call stmsh(nn,nt,nl,x,y,liw,iw,kst,ido,ifail)

参数值

变量	输入	输出
nn - 整数	区域中的节点数	
nt - 整数	网格三角形数	
nl - 整数	网格边的数量	
x - 实数数组 (lnn)	区域内所有节点的 x 坐标	
y - 实数数组 (lnn)	区域内所有节点的 y 坐标	
liw - 整数	drw（数组 iw 的维度	
iw - 整数数组（liw）7.2.1	边界结构	完整网格结构
kst - 整数数组（21）	包含统计信息	
ido - 整数	统计信息的数量	
ifail - 整数	0	错误消息

Ido 值

值	规范
1	节点-节点连接数
2	三角形-节点连接数
3	三角形面积相对于网格三角形面积的百分比数量
4	边长相对于网格边长的百分比数量
5	三角形角度相对于 60° 的百分比数量

Ifail 值

值	规范
2	验证 nt + nl = nn + 1 不正确
0	正常退出
-11	kst 的维度太小

7.2.5.2 中心： 此子程序用于计算由三个点确定的圆的中心 .

call 中心(xn1,yn1,xn2,yn2,xn3,yn3,xc,yc,r2,标记)

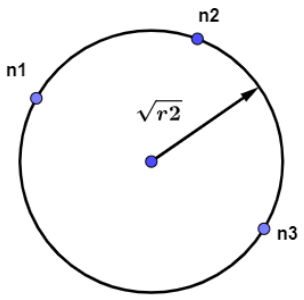


图 86：中心方案

参数值

变量	输入	输出
xn1,yn1	第一个描述节点的坐标	
xn2,yn2	第二个描述节点的坐标	
xn3,yn3	第三个描述节点的坐标	
xc,yc	中心的坐标	
xr2 - 实数	圆的二次半径	
mark	直线上的三个节点的标记（标记 $= 1$）	

7.2.5.3 Fnode: 此子程序在所有 y 坐标中搜索最小值，并将其对应的节点编号作为 n1 返回。

call fnode(v,nn,n1)

参数值

变量	输入	输出
v - 实数数组 (nn)	y 坐标表	
nn - 整数	所有节点的数量	
n1 - 整数	第一个节点的编号	

7.2.5.4 Snode: Snode 查找第二个初始节点，该节点将与第一个节点共同构成三角剖分的初始边。这个第二个节点将是与水平线形成最小角度的节点。这样可以确保第一条边的位置非常适合后续的三角剖分。

call fnode(x,y,nn,n1,n2)

参数值

变量	输入	输出
x，y - 实数数组（nn）	x 和 y 坐标数组	
nn - 整数	所有节点的数量	
n1 - 整数	第一个节点的编号	
n2 - 整数	第二个节点的编号	

8 参考文献

[1] M.I. Shamos 和 D. Hoey。几何相交问题。发表于 IEEE 第七届计算机科学基础年度研讨会，1976年。

[2] P.J. Green 和 R. Sibson。平面上的*Dirichlet*剖分计算。《计算机杂志》，第 21 卷。1977年。[3] C.W. Mastin 和 J.F. Thompson。通过椭圆系统将三维区域映射到矩形区域。《数值数学》。1978年。[4] C.W. Mast´ın 和 J.F. Thompson。椭圆系统与数值变换。《数学分析与应用杂志》。1978年。[5] J.L. Bentley 和 J.H. Friedman。范围搜索的数据结构。《计算调查》，11,4。1979年。[6] P.N. Gilbert。平面三角剖分的新结果。技术报告 ACT-15。1979年。[7] S. Ohring。泊松方程在贴体坐标中的多重网格法应用。《计算物理杂志》，第 37 卷。1980年。[8] G. Srinivasan W. E. Nicolet G. Shanks。升力式再入飞行器的流场预测。AIAA 论文 82-0026。1980年。[9] A. Bowyer。*Dirichlet*剖分计算。《计算机杂志》，第 24 卷。1981年。

[10] D.F. Watson。采用Voronoi多胞体的应用计算n维Delaunay三角剖分。《计算机杂志》，第 24 卷。1981年。[11] M.B. Aston 和 J.W. Thomas。用于水波问题的隐式格式。数值网格生成。1982年。[12] J.V. Rakich、E.Venkatapathy 和 J.C. Tannehill。航天飞机轨道器流场的数值解。AIAA 论文 83-0028。1982年。[13] P.R. Eiseman。 ”正交网格生成。数值网格生成。编者：J.F.Thomson，1982年。[14] P.R. Eiseman。自动代数坐标生成。数值网格生成。编者：J.F.Thompson，1982年。[15] P.R. Eiseman。对网格属性进行精确控制的坐标生成。《计算物理杂志》，第 47 卷。1982年。[16] P.R. Eiseman。精确控制坐标生成的高阶连续性。AIAA 论文 83-1904，《计算物理杂志》，第 47 卷。1982年。[17] L. E. Eriksson。使用超限插值生成翼身组合体周围的边界贴体网格。AIAA期刊，第 20 卷。1982年。[18] W.J. Gordon 和 L.C. Thiel。超限映射及其在网格生成中的应用。数值网格生成。编者：J.F.Thompson，1982年。[19] Z.U.A. Warsi、J.F. Thompson 和 C.W.Mastin。“用于偏微分方程数值求解的边界拟合坐标系。《计算物理杂志》，第 47 卷”。载于：（1982年）。[20] D.D. Knight。曲线坐标生成技术在内流计算中的应用。数值网格生成。编者：J.F.Thompson，1982年。[21] S. Nakamura。使用抛物型偏微分方程的推进式网格生成。数值网格生成。1982年。[22] J.S. Shang 和 R.W. MacCormack。带后体压缩襟翼的双锥构型绕流——数值比较研究。AIAA 论文 83-0098。1982年。[23] R.E. Smith。 ”代数网格生成。数值网格生成。编者：J.F.Thomson，1982年。[24] R.E. Smith。三维代数网格生成。AIAA 论文 83-1904。1982年。[25] J.F. Thompson。椭圆型网格生成。数值网格生成。编者：J.F. Thompson，1982年。[26] J.F. Thomson。一般曲线坐标系。数值网格生成。编者：J.F.Thomson，1982年。[27] J.F. Thomson。数值网格生成。编者：J.F.Thomson，1982年。[28] Z. U .A. Warsi。坐标生成的基本微分模型。数值网格生成。1982年。[29] W.L. Chow、A. Nakayama 和 D. Sharma。任意截面管道中充分发展湍流的计算。《流体力学杂志》，第 128 卷。1983年。[30] R.K Agarval 和 J. Deese。在体贴体曲线网格上求解Euler方程的数值方法。网格生成进展，美国机械工程师学会流体工程会议。1983年。[31] X. Aubert 和 M. Deville。采用贴体坐标中的紧致差分求解定常黏性流动。《计算物理杂志》，第 49 卷。1983年。[32] KR. Heavey、J .L. Steger 和 C.S. Nietubicz。弹丸构型的网格生成技术。1982年陆军数值分析与计算机会议论文集。1983年。[33] R. Camarero 和 M. Reggio。适用于涡轮机械应用的三维贴体坐标多重网格格式。《流体工程杂志》，第 105 卷。1983年。[34] P.G. Buning、D.S. Chaussee 和 D.G.Kirk。Convair 990前机身跨声速流场模拟。AIAA 论文 83-1785。1983年。[35] G.S. Deiwert。带中心推进喷流的锥形后体三维绕流：数值模拟。AIAA 论文 83-1709。1983年。[36] P.A. Gnoffo。-。AIAA J.，21，第 9 期。1983年。[37] J. Grashof。受导弹尾焰干扰的进气道内三维跨声速流动的数值研究。AIAA 论文 83-1854。1983年。

[38] S.A. Al-Nassri、J.J. Trottier、T.Z. Unny 和 M. Chandrasekhar。《用于明渠流动模拟的二维曲线网格》。《应用数学建模》，第7卷。1983年。[39] F.H. Pulliam、L. J. Chow 和 J.L. Steger。《流体动力学方程的一般扰动方法》。《美国航空航天学会》论文83-1903。1983年。[40] D.S. Chausee、M.M. Rai 和 Y.M. Rizk。《带翼体上黏性超声速流的计算》。《美国航空航天学会》论文83-1667。1983年。[41] A. Mark。《激波与目标相互作用的计算》。《美国航空航天学会》论文83-0039。1983年。[42] W.J. Usaband、E.M. Murman。《采用多重网格法求解欧拉方程的嵌入网格方法》。《美国航空航天学会》论文83-1946。1983年。[43] M. Tanemura、T. Ogawa 和 N. Ogita。《一种新的三维沃罗诺伊剖分算法》。《计算物理学杂志》，第51卷。1983年。[44] P. Colella、R.P. Shreeve 和 S. Eidelman。《戈东诺夫方法在叶栅流动建模中的应用》。《美国航空航天学会》论文83-1941。1983年。[45] S. Osher 和 S. Chakravarthy。《迎风格式与边界条件及其在一般几何形状欧拉方程中的应用》。《计算物理学杂志》，第50卷。1983年。[46] R. Sedgewick。《算法》。Addison-Wesley。1983年。[47] R. C. Sharble 和 P. Raj。《结合欧拉方程求解器的代数网格生成方法，用于模拟三维流动》。《美国航空航天学会》论文83-1807。1983年。[48] D.C. Jespersen、R.E. Childs 和 T.H. Pulliam。《欧拉方程隐式程序的增强版本》。《美国航空航天学会》论文83-0344。1983年。[49] S. Vchikawa。《二维轴对称流动问题的贴体曲线坐标系生成》。《计算物理学杂志》，第50卷。1983年。[50] M. Vinokur 和 C.K. Lombard。《具有角点奇异性的代数网格生成》。《网格生成进展》，美国机械工程师学会流体工程会议。1983年。[51] K.J. Weilmuenster。《航天飞机类飞行器大迎角无黏流动计算及与飞行数据的比较》。《美国航空航天学会》论文83-1798。1983年。[52] M.A. Serag-Eldin、Y.M. Rizk 和 A. Mobarak。《轴对称跨声速喷管流动的半椭圆计算》。《应用数学建模》，第7卷。1983年。[53] Joe F. Thomson。《计算流体力学中的网格生成技术》。《美国航空航天学会期刊》，第22卷，第11期。1984年。[54] —。《计算机图形学基础算法》。北约高级研究院系列F，第17卷。1985年。[55] J.F. Dannenhoffer 和 J.R. Baron。—。《美国航空航天学会》论文85-0484。1985年。[56] A. Dervieux、F. Angrand、V Billey 和 B. Stoufflet。—。1985年。[57] Z.U.A. Warsi、J.F. Thompson 和 C.W. Mastin。《数值网格生成：基础与应用》。1985年。[58] J. Peraire、R. Lohner、K. Morgan 和 Zienkiewicz。—。《美国航空航天学会》论文85-1531-CP。1985年。[59] T.J. Baker、A. Jameson 和 N.P. Weatherill。《完整飞机无黏跨声速流动的计算》。《美国航空航天学会》论文86-0102。1986年。[60] l. Babuska 和 A.K Aziz。《有限元法中的角度条件》。《工业与应用数学学会数值分析杂志》，第3卷。1986年。[61] J. Boris。《一种用于确定最近邻的向量化算法》。《计算物理学杂志》，第66卷。1986年。[62] R. L¨ohner 和 K. Morgan。—。《美国航空航天学会》论文86-0499。1986年。[63] K. Morgan、R. Lohner 和 O.C. Zienkiewicz。《欧拉方程的自适应网格细化》。载于《有限元计算中的精度估计与自适应加密》。1986年。[64] K. Morgan、J. Peraire、M. Vahdati 和 O.C. Zienkiewicz。《可压缩流动计算的自适应重新网格划分》。《计算物理学杂志》第72卷。1987年。[65] G. Holmes 和 D. Snyder。《使用Delaunay三角剖分生成非结构三角网格》。《计算流体力学中的数值网格生成》，Pineridge出版社。1988年。

[66] *K. Morgan*、*J. Peraire*、*L. Formaggia* 和 *Zienkiewicz*。《三维有限元欧拉计算》。《国际数值方法工程》，第*26*卷。*1988*年。[67] *K. Morgan*、*L. Formaggia*、*J. Peraire* 和 *J.Peiro*。《在*CRAY*计算机上实现三维显式欧拉求解器》。《第四届*CRAY*超级计算机科学与工程国际研讨会论文集》。*1988*年。[68] *T.J. Baker*。《使用约束*Delaunay*三角剖分自动生成复杂三维区域网格》。《计算机工程》，第*5*卷。*1989*年。[69] *E. Brisson*。《*d* 维几何结构的表示：拓扑与序》。《算法学》，第*4*卷。*1989*年。[70] *D.P. Dobkin* 和 *M.J. Lazlo*。《用于操作三维细分的基本操作》。《算法′学》，第*4*卷。*1989*年。[71] *T.J. Baker*。《使用广义德劳内算法生成非结构网格》。载于《网格生成在复杂三维构型中的应用》，*AGARD*会议论文集第*464*号。*1990*年。[72] *J. Bonet* 和 *J. Peraire*。《用于几何搜索与求交问题的交替数字树（*ADT*）算法》。*1990*年。[73] *S. Rippa*、*D. Nira*、*D. Levin*。《数据依赖三角剖分的构造算法》。《近似算法*II*》，*Chapman and Hall*。*1990*年。[74] *S. Rippa*、*D. Nira*、*D. Levin*。《用于分段线性插值的数据依赖三角剖分》。《数值分析杂志》，第*10*卷，第*1*期。*1990* 年。[75] *W.N. Dawes*、*G.L.D. Siden* 和 *P.J. Albraten*。《使用解自适应非结构网格对叶片栅中的二维黏性可压缩流进行数值模拟》。《涡轮机械杂志》，第*112*卷。*1990*年。[76] *T.S. Tan*、*H. Edelsbrunner* 和 *R. Waupotitsch*。《一种用于最小最大角三角剖分的*O(n^2 log n)*时间算法》。《第六届*ACM*计算几何研讨会论文集》。*1990*年。[77] *F. Huet*。《复杂三维构型中的自动网格生成：采用推进前沿方法》。《*AGARD-CP-464*》。*1990*年。[78] *R. Lohner*。《瞬态问题中三维非结构网格的生成与自适应》。《*AGARD-CP-464-19*》。*1990*年。[79] *J. Cabello* 和 *R. L¨ohner*。《使用*AFM*生成的定向拉伸单元的优化变分法》。《计算流体力学及相关领域中的数值网格生成》。*1991*年。[80] *L. Formaggia*。《用于三维空气动力构型的非结构网格生成算法》。《计算流体力学及相关领域中的数值网格生成》。*1991*年。[81] *J.L. Thomas*、*G. Warren*、*W.K. Anderson* 和 *S.L. Krist*。《自适应方法的网格收敛》。《美国航空航天学会》论文*91-1592-CP*。*1991*年。[82] *L.J. Guibas* 和 *J. Stolfi*。《用于操作一般细分和计算*Voronoi*图的基本操作》。《*ACM*图形学汇刊》，第*4*卷。*1991*年。[83] *C.J. Hwang* 和 *J.L. Liu*。《非结构三角网格上的局部隐式总变差递减格式》。《美国航空航天学会期刊》，第*29*卷，第*10*期。*1991*年。[84] *D.J. Mavriplis*。《用于高雷诺数黏性流的非结构化与自适应网格生成》。《计算流体力学中的数值网格生成及相关领域》。编者：*A.S. Arcilla*，*1991*年。[85] *M.L. Merriam*。《一种高效的*Delaunay*三角剖分推进前沿算法》。《美国航空航天学会》论文*91-0792*。*1991*年。[86] *T.J. Mitty*。《三维非结构网格的生成与自适应修改》。《计算流体力学中的数值网格生成及相关领域》。编者：*A.S. Arcilla*，*1991*年。[87] *T. Taniguchi* 和 *E. Ohta*。《基于*Delaunay*的复杂边界几何三维物体网格生成》。《计算流体力学中的数值网格生成及相关领域》。编者：*A.S. Arcilla*，*1991*年。[88] *J. Ruppert*、*S. Meshkat* 和 *H. Li*。《基于德劳内四面体剖分的三维自动非结构网格生成》。《计算流体力学中的数值网格生成及相关领域》。编者：*A.S. Arcilla*，*1991*年。

[89] R. Vilsmeier 和 D. Hanel。 非结构化网格的生成与自适应。《计算流体力学中的数值网格生成及相关领域》。编者：A.S. Arcilla，1991年。[90] N.L. Wiltberger。 个人通信，*NASA* 艾姆斯研究中心，*M.S. 258-1*，莫菲特机场。1991年。[91] W.K. Anderson。 非结构化网格上 *Euler* 方程的网格生成与流动求解方法。NASA 兰利研究中心，美国，未发表手稿。1992年。[92] H. Deconinck J.D. M¨uller P.L. Roe。 *Delaunay* 三角剖分中节点生成的前沿法。AGARD-R-787。1992年。[93] J. Peraire K Morgan 和 J. Peiro。 可压缩流的非结构化网格方法。《平流主导流的非结构化网格方法》，AGARD-R-787。1992年。[94] l. Chang。 自适应、非结构化有限元多材料热分析。《航天器与火箭》，第30卷，第1期。1993年。[95] S.Rebay。 利用 *Delaunay* 三角剖分和 *Bowyer-Watson* 算法高效生成非结构化网格。《计算物理学杂志》106。1993年。