博 士 学 位 论 文

自适应有限元网格生成算法研究与应用

Research and Application of
Adaptive Finite Element Mesh Generation Algorithm

作 者 姓 名：单菊林

学科、专业 ：计算机应用技术

学　　号 ：10403023

指 导 教 师：张洪武 关振群 郭英乔 顾元宪

完 成 日 期：

大连理工大学

Dalian University of Technology

独创性说明

作者郑重声明：本博士学位论文是我个人在导师指导下进行的研究工作及取得研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得大连理工大学或者其他单位的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

作者签名：_____日期：_____

摘　　要

有限单元法(Finite Element Methods, FEM)是当今最成熟、应用最广泛的一种数值分析方法。然而面对越来越复杂的工程与科学问题，有限元法还经常显得力不从心，限制其发挥更大作用的一个重要因素是网格生成问题。网格生成是计算机辅助工程(Computer Aided Engineering，CAE)的共性支撑技术之一，全自动高质量网格生成方法研究一直是 CAE 领域的研究热点。

本文研究、改进和实现了自适应三角形和四面体网格生成方法。包括基于黎曼度量的二维自适应三角形网格生成方法、三维复杂组合曲面的自适应网格生成方法以及三维四面体自适应网格生成方法。本文还介绍并实现了网格剖分程序和多种 CAD 软件通信的 B-Rep 接口、基于拓扑连接的数据结构以及和剖分程序密切相关的点的定位查找算法。具体来说，本文是从以下几个方面展开论述的：

首先是绪论部分，主要介绍了国内外有限元网格研究的现状和发展的方向。随后介绍了本文提出的有限网格生成模块与多种造型软件通信的接口，以及有限元网格生成的整体框架。给出了一种有限元网格生成的公共几何接口，这个接口把不同造型软件的对模型几何和拓扑信息的描述的差异性进行封装，以统一的形式提供给剖分程序使用。采用简化的边界描述法（Boundary Representations，B-Rep）来组织模型的几何和拓扑信息，减少了剖分程序实现的复杂性，提高了剖分算法的可复用性。给出了网格生成的一般框架，使得扩展功能后的算法更容易成为现有框架的一部分，以相同的方式为网格剖分算法提供新的服务。

第 3 章是网格生成算法的数据结构部分，介绍了基于拓扑连接的网格数据结构。传统的节点-单元这种紧凑型表达虽然简单而且能满足某些应用，但是由于其缺少必要的拓扑连接信息，因而在诸如网格优化、自适应等程序实现中是非常复杂的，因为这些程序需要用到连接信息的时候不得不重新建立连接信息表，这就增加了这些应用程序实现的复杂性。该部分针对面剖分和体剖分两种常用的应用，给出并实现了基于拓扑连接的数据结构，这一数据结构广泛应用到本文的网格剖分和优化过程中。

第 4 章介绍二维复杂域的自适应有限元网格生成。工程中很多实际问题都可以用二维有限元网格来模拟。另外，三维参数曲面的自适应网格生成也是首先在其二维的参数域上生成网格，然后投影到参数曲面的三维物理域上。本章研究和实现了基于黎曼度量的二维自适应网格生成算法：对在黎曼空间下网格剖分中常用的数值计算方法进行了总结；介绍了用来描述单元尺寸和形状信息的四叉树背景网格的建立和平衡过程；给出了基于平衡四叉树的梯度黎曼度量场的建立算法；对基于黎曼度量的二维自适应波前推进

算法和基于二分的二维自适应网格生成算法给出了详细实现步骤。本文的二维自适应网格生成方法不但能够生成各向同性的有限元网格，而且能够生成各向异性的有限元网格，从而满足流体动力学分析的需要。

第5章将上述二维自适有限元网格生成算法和基于拓扑连接的数据结构应用于金属破坏过程的模拟。这个部分主要介绍了在实现过程中所遇到的一些问题，给出了一些解决方法，包括新旧网格间状态变量的转换；相交边界线段快速侦测和修复以及非流网格的删除等。

第6章给出了三维空间的黎曼度量和曲面自身的黎曼度量相结合的三维复杂参数曲面自适应网格生成的改进波前推进算法。针对周期性曲面的特殊性，提出了三维参数曲面的迁移波前推进算法，有效避免了由于虚边界而导致的网格质量变差的问题。详细阐述了曲面参数域上任意一点的黎曼度量的计算和插值方法；提出按层推进和按最短边推进相结合的方法，在保证边界网格质量的同时，提高曲面内部网格的质量。算例表明，该算法对复杂曲面能够生成高质量的网格，而且整个算法具有很高的效率和可靠性。

第7章将基于几何和拓扑的网格剖分框架和三维组合曲面自适应有限元网格生成算法应用于金属冲压过程模拟。在金属冲压成形模拟中附加面及其有限元网格的生成是模拟的关键步骤之一。这个部分主要介绍了附加面的生成，B-Rep 模型的构建以及有组合冲压件和附加面构成的组合曲面的有限元网格生成。

第 8 章针对三维四面体推进波前算法（AFT-Advancing Front Technique）存在的效率与收敛性问题，提出了一整套改进方案。基于拓扑连接的网格数据结构使用大大提高了整个算法的效率。通过在网格生成过程中动态维护前沿的尺寸信息，提高四面体单元的整体质量。在内核回退时通过引入前沿优先因子，改变前沿推进的路径，大大增加了成功回退的概率；对于极少数不能回退的内核采用基于线性规划的插点方法加以解决，这样就保证了整个算法的收敛性。在网格生成以后，通过删除不必要的内部节点、合并相关四面体单元以及对所有内部节点进行基于角度的优化，从而进一步有效地提高了网格质量。数值算例表明，本文实现的三维四面体推进波前算法的改进算法具有接近线性的时间复杂度，生成网格质量好。

第 9 章介绍了作者在全六面体网格生成方面研究和实现的成果。以四面体-六面体基本转换模板为基础，提出了一系列具有伸缩性的扩展转换模板，可将四面体分解为不同数量、不同密度过渡形式的的六面体单元；提出了基于几何造型的边界节点坐标修正方法，使边界网格能够更好地拟合几何模型边界，给出了基于转换模板的三维实体全六面体网格生成的算法流程和剖分算例。

文章最后的第 10 章介绍了本文提出的改进的可回退的点定位搜索算法，在这个算法有效避免了搜索路径的环的形成，使得算法不但能够应用到二维的单连通凸域，而且可以运用到三维任意复杂域的凸多边形剖分中。在不损失效率的前提下提高了算法的应用范围。

本文得到了国家自然科学基金（10572032，10421002），国家杰出青年科学基金（10225212），法国外交部尔菲尔博士奖学金(Bourse d'Excellence EIFFEL)以及大连市科学技术基金联合资助。

关键词：网格生成；自适应；几何接口；网格数据结构；组合曲面；四面体；有限元

Abstract

The Finite Element Methods (FEM) is one of the general engineering numerical methods. However, for most complex applications, the engineer should have a lot of practical experience and time to succefully use the FEM to solve engineering problems. For this reason many researchers are investigating ways to automate the finite element method, thus allowing an improved productivity, more accurate solution, and used by less trained personnel. The finite element method can be divided into several sub-steps, often the most time consuming and experience requiring task faced by an analyst is the discretization of a general geometric definition of the problem into a valid and well conditioned finite element mesh. Finite Element Analysis of the accuracy and the cost depend directly on the element size, shape and number. Automatic quality finite element mesh generation can make FEM easier to be a powerful tool for the general engineer. Therefore, automation of the mesh generation is an important prerequisite for the complete integration of the FEM with design processes in computer aided engineering (CAE) and manufacturing (CAM) systems.

In this paper, an improved adaptive triangle and tetrahedral adaptive mesh generator has been developed, which includes two-dimensional Riemannian metric based adaptive mesh generation, complex 3D surface adaptive mesh generation and adaptive 3-D tetrahedral mesh generation. It also introduces the interfaces of B-Rep which was used to exchange data between mesh generator and various CAD software. Topology-Based mesh data structures and the procedures of subdivision point location algorithm are also given here.

The Riemannian metric based adaptive mesh generation algorithm can generate not only two-dimensional isotropic elements but also the anisotropic elements, thereby satisfying the needs of computational fluid analysis. For the combination of three-dimensional complex surfaces, an extended Advancing Front Technique with shift operations and Riemann metric named as shifting-AFT is presented for generating finite element meshes on 3D surfaces, especially 3D closed surfaces. Riemann metric is used to govern the size and shape of the triangles in parametric space. The shift operators are employed to insert a floating space between real space and parametric space during 2D parametric space mesh generation. Combining the shift operators, the advancing front technique kernel is extended to overcome the mesh quality-worsening problem in closed surface mesh generation due to introducing virtual boundaries into 2D open parametric domains generally mapped from closed surfaces. The shifting-AFT can generate high-quality meshes and guarantee convergence in both open surfaces and closed surfaces. For the shifting-AFT, it is not necessary to introduce virtual boundaries manually or automatically while meshing a closed surface, so that the boundary discretization procedure is simplified very much, and moreover, better-shaped triangles will

be generated because there are no additional interior constrains yielded by virtual boundaries. Comparing with direct methods, the shifting-AFT avoids carrying out costly and unstable 3D geometrical computations in real space. The examples demonstrate the advantages of the shift-AFT in 3D surface mesh generation, especially for closed surfaces.

In Chapter 8, a reliable and effective tetrahedral meshing algorithm is also proposed based on advancing front method. The operators such as insert query and delete like a database are implemented by using topology based mesh data structures which accelerates the whole algorithm. Instead of preparing a background mesh for mesh spacing control, this information is estimated at the beginning of each layer at each node from the area of connecting triangles on the front and a user-specified stretching factor. A cell searcher is prepared to correct the mesh spacing information and to perform geometric search efficiently. During rolling back the advancing path is changed by changing preferential factor of front, as a result the times of rolling back is decreased significantly. Node inserting based on linear programming technique guarantees the convergence of the algorithm. At the end of the mesh generation process, unwanted node removing and angle-based smoothing are employed to enhance the resulting mesh quality. The examples demonstrate that high quality tetrahedral meshes can be generated within a reasonable time limit

In Chapter 9, Based on the basic transform template from tetrahedron to hexahedron, a series of flexible extended transform template is presented. The number and the density of the hexahedral mesh transformed from the tetrahedral mesh can be controlled using the varied templates and their assemblies. Thus, it is needless that the initial tetrahedron mesh is generated very finely. According to different type of boundary mesh nodes, using different map method to modify new generated nodes' coordinates. This finds that it could make the new generated nodes, which are nearest to the boundary; locate the entity's boundary accurately. The essential steps of this scheme are described by means of flow, which based on CAD platform

Point location is one of the most basic searching problems in the computational geometry. It has been largely studied on the aspect of the query time in the worst-case and many methods have been proposed such as Counter Clockwise Wise Search and Barycentric Coordinates Search. However most of them aim at the convex field. For a non convex problem such as a concave field or a convex field with holes, these searching schemes may fail. The numerical experience shows that even for convex problem, the searching path may lead to an infinite loop for some special case and can not find the element containing the query point. In Chapter 10 a robust backward search method based on Walk-through algorithm is proposed to deal with the searching problems in non-convex fields and to avoid the problems of infinite loop. Another important improvement is to locate the query point on a 3D surface mesh. Several

examples demonstrate that the present method is efficient and robust for the workpieces of complex geometry.

The author would like to appreciate the joint supports to this project by the National Natural Science Foundation of China (10572032, 10421002), Outstanding Young Scientists Foundation (10225212) and French Foreign Ministry Eiffel PhD scholarship (Bourse d 'Excellence EIFFEL).

Key Words：Mesh generation; Adaptive; Mesh data structures; Surface mesh generation; Tethedral mesh generation; Finite element method

目　　录

1. 绪论

1.1 引言

从 20 世纪 70 年代起，经过众多科学家长期不断的研究和努力，有限单元法(Finite Element Method, FEM)已经成为解决众多工程分析问题的通用方法[1]。应用有限单元法的一个基本前提是将表示一个结构或连续体的求解域离散成若干有限个子域（单元），即有限元网格生成，并通过它们边界节点相互联结成为组合体，用这样的组合体来模拟或逼近求解区域。有限单元法是一种近似方法，它和求解问题的真实解的逼近程度依赖于单元的数目、单元的形状、以及节点的位置等等。单元的尺寸和节点的位置对问题解的精度有相当大的影响，减小单元尺寸和增加节点的数目一般会提高求解的精度，但这样同样会增加求解的费用(时间、内存占用）。为了能够在较短的时间内得到较高精度的解，就要在单元和节点数目、节点的位置与求解精度之间加以权衡和调整。这样就需要网格生成程序能够根据目标域的几何形状或者上一次分析的结果自适应地生成有限元网格，也就是在求解域中需要精细刻画的部分(如曲线和曲面的曲率较大部分，应力集中区域)生成的网格应该更密一些，而其它部分(曲率较小，应力变化较为均匀)的网格可以相对稀疏一些，这样可以在不增加节点的前提下提高求解的精度，满足工程实际分析的需要。

网格生成是有限单元法的关键步骤，从有限单元法诞生以来就有很多科学工作这致力于这方面算法的研究。现在这项技术已经比较成熟，并且很多商业计算机辅助工程（Computer Aided Engineering, CAE）软件都有相应的模块，一般来说这些 CAE 软件都提供二维、曲面、三维体网格生成功能。目前 CAE 商业软件的生成的网格对线性问题都能得出一个很好的解决，但是对于那些要求生成更高质量的复杂或者非线性问题这些商业软件往往很难以全自动的方式生成用户所需要的网格。对于这些问题，往往需要花去分析工程师大量的时间，以手工方式调整已有的网格来满足工程分析的需要。另外，有限元分析的精度很大程度上依赖于网格生成的质量，而现成的商业软件在处理这些复杂问题时往往不能得到用户满意的网格，这就需要相关领域的科学工作者进一步深入研究新的网格自适应算法。

在对模型进行网格生成以前，首先要获取已经定义的待分析域几何模型的几何和拓扑信息。几何模型和有限元模型之间数据的交换是 CAE 软件的重要组成部分。另外，数据结构的设计在网格生成过程中的重要性也显得尤为突出，成熟的数据结构不但要考虑计算时间和空间成本(内存占用)，同时也要考虑整个数据结构的易维护性。

网格生成和计算几何密切相关，Shewchuk, J. R.[2, 3]对一些基础的计算几何算法进行了总结并对几何计算算法的健壮性进行了讨论。P. J. Schneider 和 D. H. Eberly [4]的关于计算几何的论著在本文的算法实现中起了很大的作用。需要指出的是，除了这些算法以外，空间点的定位和查找算法在网格生成算法中应用很广，文章最后将介绍这两个算法。

1.2 自适应网格生成算法研究的现状

许多学者对有限元网格生成方法研究进行了概括和总结，如 P. L .George、S. H. Lo 、H. T. Y. Yang 和关振群等[5-11]。在这里主要针对上面所介绍的网格剖分所涉及的主要部分，回顾一下国内外其它同行在这方面的研究的成果与现状。

1.2.1 模型几何与连接信息的获取

网格生成中非常重要的部分是模型的几何和拓扑信息的获取。忽视模型的几何和拓扑信息会导致网格剖分后的有限元模型和几何模型之间的孤立，这样就给网格生成后对模型边界的恢复和进一步的优化带来困难。在网格剖分过程中模型的几何信息对于网格生成是非常重要的，因为它是待分析域的初始描述。在单元的细化和对模型的其它后序操作中都要用到模型的几何信息。实际上几何和拓扑信息描述和获取也是网格生成和其它后序操作中最主要的瓶颈之一。目前在网格生成中，对 CAD 模型几何信息的获取通常有以下四种方法[12]，它们是：

(1) 转换和还原；

(2) 离散化的表示；

(3) 直接法;

(4) 统一的几何和拓扑接口

转换和还原是最常用的方法，它通过像 IGES、STEP、Parasolid 等工业标准文件格式来传递数据，这种方法最大的弊端在于由于不同几何造型软件的精度的不同，从而导致转换和还原过程中对模型的污染（模型转化过程中导入在裂缝和重叠）[13, 14]。离散化的表示技术是基于 CAD 系统生成的小面模型（实体的表面用一些小三角面表示）。这种技术通常用来修正由于使用不同 CAD 系统造成模型的污染，但是小面模型的最大的问题是在面和面之间有时存在不相容问题，如图 1.1 所示。直接法是用 CAD 软件提供的应用程序接口(Application Program Interface, API)来访问模型的几何拓扑数据，由于这种方法直接使用 CAD 软件提供的 API，从而可以和 CAD 软件达到无缝结合。这种方法的问题在于不同 CAD 软件提供的接口函数是不同的，目前还没有一个统一的标准 API 可以使用。现在大部分 CAD 软件的几何建模功能基本是一致的，实际上这些功能都是

由处于核心地位的几何引擎提供的,成熟的 CAD 软件都提供可访问几何信息和拓扑信息的 API，比如，ACIS，Parasolid 等，所有这些引擎都提供模型的几何和拓扑信息访问的 API，同时也提供对几何信息进行修改的 API。但是，不同的几何造型软件之间无论在语义上还是在底层的拓扑模型都存在这样那样的差别，这样，对于类似网格剖分这样的对模型几何和拓扑信息有很大依赖的模块就迫切需要一个公用的几何接口来消除这些差异，以使得第三方应用程序就可以通过该公用的几何接口来获取模型的几何和拓扑信息，而不需要关心模型是什么 CAD 软件构建的。如 Tautges 等[15] 基于 ACIS 开发出公共几何模块，该研究成果被用在 Sandia.试验室的 CUBIT 软件中。实现的大致思路如图 1.2 所示 。在此以前 Panthaki 等人[16]也做了许多杰出的工作，建立了 VGI 库，简化了网格生成过程对不同 CAD 平台对几何信息的访问。S. Gopalsamy 等[17]也给出了基于拓扑模型的网格生成的接口，使得几何模型和网格模型之间可以相互通信。类似的还有 Robert Haimes[18]提出的为分析和设计服务的公共几何信息获取方法等。

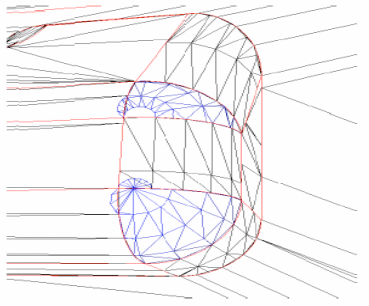


图 1.1 基于小面的模型及面不相容问题

Fig.1.1 Facet based model and Incompatibility

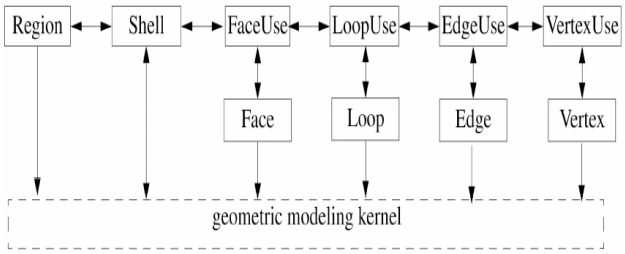


图 1.2 一致的拓扑模型

Fig1.2 Compatible Topology based model

造型软件的几何引擎通常以库的形式存在于造型软件中，许多第三方的应用程序在几何引擎的基础上进行开发，网格生成就是其中的一种。通常这些第三方的应用程序需要基于不同几何造型软件（引擎）来实现增值开发，一般来说在开发这些第三方的应用程序以前就需要对不同的 CAD 软件提供的 API 进行封装，提供统一的几何接口来消除不同 CAD 软件 API 集合之间的差异，以简化 API 的使用，并且把封装后的接口集合进行归并，形成更高级的开发包供类似于网格剖分的第三方应用程序使用。本文就采取类似的形式，首先给出了在网格剖分中常用的几何和拓扑接口，它作为整个前处理框架的一部分，为其它有限元网格生成算法提供统一的软件接口。

1.2.2 有限元网格数据结构

从几何的角度出发，有限元网格是由节点和单元组成的。其中，节点是指定区域内选定点的集合，单元则是这些节点之间的连接关系。由于节点之间连接关系的不同，在

一定的区域内，相同的节点集合会生成不同类型的单元。例如在平面区域的网格生成中，既可以产生三角形单元，也可以产生四边形单元；在空间结构的网格生成中，既可以产生四面体单元，也可以产生六面体单元。虽然由于节点之间的连接关系不同可以生成不同的单元，但区域内的节点和单元之间要满足拓扑有效和几何有效条件。拓扑有效是指，在 d 维问题中，两个单元的 $d-1$ 维公共部分仅为这二者共享，不会与其它单元共享；几何有效是指，所有单元的集合覆盖整个求解区域，并且每个单元的 Jacobian 行列式都为正值。

即使最简单的有限单元法的应用也要用到节点和单元及这两个对象的集合。节点-单元这种紧凑型表达虽然简单且能满足某些应用，但是由于其缺少必要的拓扑连接信息，因而在诸如网格优化、自适应等算法中，如果仅使用这种紧凑型的数据结构，会大大增加实现的复杂度。因为这些算法在需要用到其它连接信息的时候不得不需要建立新的连接信息表，而很多连接信息的使用频率非常高，所以一般在网格生成中会拓展这种紧凑型数据结构。另外，在后处理系统显示有限元模型的时候为了加快显示速度，需要模型的节点表、边表、面表以及各拓扑对象间的连接信息[19]。

网格数据结构的设计和实现也一直是网格剖分领域研究的热点，针对不同的应用已经有很多基于拓扑连接的网格数据结构。对于面网格，比较有代表性的是半边结构(Half-Edge)，M. Botsch[20]等给出了一个 C++语言的具体的实现。然而对于三维体剖分（如四面体/六面体）这种半边结构就不适用了。为了使网格数据结构能够适应更多具体的应用，更一般的方式是借鉴表述实体模型的边界描述方法(Boundary Representation, B-Rep)[21, 22]定义基于拓扑连接的网格数据结构[17, 20-23]。这种数据结构包括两个方面：拓扑对象（Region，Face，Edge，Vertex）和连接关系（adjacencies）。在这种数据结构中 Region 可以表示像四面体/六面体等三维单元，Face 是包围这些体单元的外表面，可以是三角面(四面体)，四边形(六面体)等，而 Face 是由边(Edge)构成的，Edge 是由两个点(Vertex)构成的。对于上述拓扑对象不同的连接方式 Rao V. Garimella[23]在内存和操作效率上进行了仔细的比较，结果表明采用 R2,R4 和 F4 三种连接方式性能较好。然而这些结构也只是相对几个不同应用和操作的统计结果。在现代有限元分析中，需要根据用户不同的应用进行面网格生成、体网格生成、网格优化及自适应、分析计算、后处理等过程，这些不同的应用和过程所需的拓扑对象集合以及对象间的连接关系是不同的，为此 J.F Remacle 等[24]提出了面向算法的网格数据结构。在文章的第 3 章，本文针对面剖分和体剖分两种常用的应用，介绍并实现了基于拓扑连接的网格数据结构，这一数据结构广泛应用到本文的网格剖分和优化算法中。

1.2.3 通用的网格生成方法

在工程应用中，有限元网格主要有两种：

一种网格是结构化网格。在结构化网格中，所有内部节点都有相同数量的邻近节点和相同数量的邻近单元，即所有的内部节点都具有相同的度。结构化网格中的单元全部是四边形单元（二维平面或三维曲面）或六面体单元（三维实体）。结构化网格生成方法用来生成结构化网格，这种方法的代表方法是映射法。映射法的优点是算法简单、速度快、单元质量好、密度可控制，它既可生成四边形单元网格又可生成六面体单元网格。映射法一般可直接处理单连通域问题,但对于复杂多连通域问题,需要首先用手工或自动方法将待剖分域分解成几何形状规则的可映射子区域，然后在每个子区域内应用映射法。然而，在实践中仍有几个难点需要克服[10]：(1) 如何自动地将复杂的不可映射的待剖分域分解成简单的可映射的子区域；(2) 如何满足某些物理问题中对网格疏密过渡的要求；(3) 如何满足子区域之间的网格相容性要求。

另一种网格是非结构化网格。在非结构化网格中，对节点的连结和分布放松了要求，允许任意数量的单元在一个节点处交汇，即所有内部节点的度都可能不同。虽然四边形单元和六面体单元也可以构成非结构化网格，但实际上在非结构化网格中一般采用三角形单元和四面体单元。非结构化网格生成方法的一个突出的优点是它可以适应复杂的几何形体，与几何形体的形状关系不大，结合一定的控制方法，可以生成质量很高的单元，而且利用非结构化网格生成算法可以实现区域网格生成的自动化。非结构化网格生成方法面临的主要解决问题是数据结构的设计和操作，这个问题是所有非结构化网格生成算法的关键问题。需要指出的是，三角形单元和四面体单元不是非结构化有限元网格的唯一选择，由四边形单元或六面体单元构成的有限元网格只要节点的度有所不同，此时的有限元网格仍然是非结构化网格。典型和通用的非结构化有限元网格生成方法是基于栅格法、Delaunay 三角化方法、推进波前法(AFT 方法)以及基于 STL 文件的网格生成方法。下面分别简单介绍一下这四种方法：

(1) 基于栅格法[25-29] (Grid-based Approach)的基本流程是，首先用一组不相交的尺寸相同或不同的栅格（cells）覆盖在目标区域上面，保留完全或部分落在目标区域之内的栅格，删除完全落在目标区域之外的栅格；然后对与物体边界相交的栅格进行调整、剪裁、再分解等操作，使其更准确地逼近目标区域；最后对内部栅格和边界栅格（特别是边界栅格）进行栅格级的网格剖分，进而得到整个目标区域的有限元网格。这种几何适应性强，网格密度可控制，且算法效率较高。该方法的主要问题是边界网格质量较差，为了更好地逼近模型的边界，往往需要在边界上添加三角形(2 维)或四面体(3 维)网格。

(2) Delaunay 三角化方法[30-48]是目前流行的通用的全自动非结构化有限元网格生成方法之一。Delaunay 三角剖分最早由 Delaunay 于 1934 年提出，Delaunay 三角剖分不仅是有限元网格生成中的重要方法，而且在数学、地理、工程等许多领域有着重要应用。在 Delaunay 三角剖分满足一个重要的准则：即空心圆(二维)/空心球(三维)准则。这个准则要求在 Delaunay 三角剖分中任何一个节点不能在网格中任何一个三角形(三维情况下是四面体)的外接圆内。二维 Delaunay 算法的基本步骤是：首先定义一个包含所有节点的初始网格，最简单的情形是单个三角形；向网格中插入一个节点，找出其外接圆包含此节点的所有三角形，删除这些单元形成一个包含插入节点的空腔(Cavity)；将该插入节点与空腔的每条边相连，形成新的三角形单元；上述的节点插入过程重复进行，直到全部节点插入完毕。

由于 Delaunay 三角剖分方法仅对凸域的网格生成有效，对于非凸域的网格生成则不能保证网格中区域初始网格的完整，因而对非凸域应用 Delaunay 三角剖分方法完成网格生成后必须引入恢复边界的步骤。恢复了边界的 Delaunay 三角剖分并不能严格满足 Delaunay 准则，因此称为约束 Delaunay 三角剖分方法。二维平面问题的边界恢复比较简单，并且有明确的理论基础保证边界恢复结果是收敛的[45-47]。三维问题的边界恢复过程要比二维问题复杂得多，在二维问题中行之有效的边界恢复算法不能直接扩展到三维问题中，因而三维 Delaunay 的边界恢复算法[31, 38, 42, 43]近年来一直受到关注。

Delauany 算法应用到三维的另外一个问题是薄元问题（Sliver Element），所谓薄元是指一个四面体的四个节点几乎共面，但这个四面体仍然符合 Delaunay 三角化方法单元外接球为空的剖分准则。这种单元的质量很差，会导致有限元计算结果的严重误差，甚至出现错误。薄元分解法、节点抖动法[48]、以及基于优化的方法是[48]是目前处理薄元问题的常用方法。需要说明的是 Delauany 方法只能生成三角形或四面体网格。

(3) 推进波前法（Advancing Front Technique, AFT）[49-59] 是另外一种通用的全自动非结构化有限元网格生成方法。AFT 方法的基本思路是先离散区域边界，二维平面区域边界离散后是首尾相连的线段集合，三维实体区域边界离散后是拓扑相容的三角形面片的集合，这种离散以后的区域边界称为前沿。接下来的步骤就是从前沿开始，依次插入一个新节点或采用一个已存在节点生成一个新单元。一个新的单元生成以后，前沿要进行更新，即向区域的内部推进。这种插入节点、生成新单元、更新前沿的过程循环进行，当前沿为空时表明整个区域剖分结束。AFT 方法可以完全保持区域边界，并可以控制每一步生成单元的尺寸及形状。AFT 方法不同于 Delaunay 三角剖分算法，它没有后者那样成熟的理论依据，在很多情形下靠经验解决问题，但是这并不妨碍它的成功应用。

AFT 方法的基本思路虽然简单，但是在程序实现上是需要很多技巧的，这是因为在 AFT 方法的实现中，涉及到数据的存储方式，前沿的查找方法，前沿之间的相交判断等，这些环节的实现效率都直接影响 AFT 方法的运行效率。总之，AFT 方法有以下特点：

(a) AFT 方法能够在生成节点的同时生成单元，这样就可以在生成节点时对节点的位置加以控制，从而控制单元形状、尺寸以达到质量控制、局部加密及网格过渡的要求。

(b) AFT 方法生成新单元的同时需要进行大量的相交判断、包含判断、以及为了保证单元的质量而进行的距离判断。相交判断包括线段之间的相交判断，线段与三角形面片之间的相交判断；包含判断主要指单元是否包含前沿节点的判断；距离判断包括线段与线段的距离，线段与前沿节点的距离以及线段与三角形面片的距离。上述判断在整个 AFT 方法实施过程中耗用了大约 80％的机时。因此在实施 AFT 方法时，务必精心设计数据结构，尽量减少需要进行判断的数量，以提高 AFT 方法的效率。

(c) AFT 方法在三维应用中存在收敛问题。所谓收敛性问题是指，在三维问题的某些特殊情况下，一个多面体无法按照三角剖分的要求对其进行三角剖分，一个典型的例子是 Schonhardt 多面体。AFT 方法的收敛性问题有其特殊性，需要插入多个节点才能将其三角剖分。

AFT 方法不但能够生成三角形和四面体网格而且能够生成四边形和六面体网格。本文的二维自适应剖分算法、曲面的自适应剖分算法以及三维的体剖分算法都是基于 AFT 原理的，文中针对 AFT 算法实现过程中的一些难点给出了一些改进意见。

(4) 基于 STL 文件的方法[60-65]。这种方法主要要在曲面网格的自适应生成中。在曲面网格生成过程中并不是所有的 CAD 软件能提供模型中所有曲面的参数化表示，而这些 CAD 软件一般都能提供模型的 STL(Stereolithography)文件，这个时候我们就不能应用直接法[51, 54]或者映射法[41, 53, 56, 66-68]来生成曲面的有限元网格；另外在冲压件的动态模拟中，一般只能得到上一个时间步的有限元网格，而不是模型的几何描述。这时同样需要根据计算结果重新生成新的精度更高的网格，这个时候只能以上一时间步网格作为基础。所以这种方法是一种由网格到网格的方法，如图(1.3)所示，对于 STL 文件，一般来说它只是提供每个三角形的顶点坐标，而且三角形之间可能是不相容的，这样首先需要把 STL 文件转化为相容的有限元网格，然后可以采用节点合并、消除过小的线段、采用长边分裂[69]或者长边分裂和 Dealunay 准则相结合的方法[69, 70]生成新的网格。新网格生成后可以通过网格优化的方法[71-77]来提高新生成网格的质量。用这种方法生成有限元网格的一个关键步骤是求取新的节点位置，也就是当新节点生成时，以及节点位置发生变化时，一般需要把点投射到合适的位置。有两种方法可以采用，一种是

E.Bechet 等[65]给出的方法，也就是保留最近时间步的旧网格，把新节点投影到旧网格中最近的一个单元中去；另外一种是 P.J.Frey[78]给出的方法：首先也是找到和新节点最近的旧网格中的某个单元，然后根据这个单元的三个节点的坐标和平均法线方向，拟合一张和周围三角形用同样方法得到的曲面一阶导数(G1)连续的面，最后把这个新节点投影到这张拟合面上去。很明显第二种方法比较复杂，但是得到的网格和理想状态逼近程度更高。对于 G1 连续的面采用 D.J.Walton 和 D.S.Meek[79]给出的基于贝塞尔曲面的方法或者 Xue, D[80]的方法；对于节点的平均法线方向的计算可以采用 Max[81]的方法。

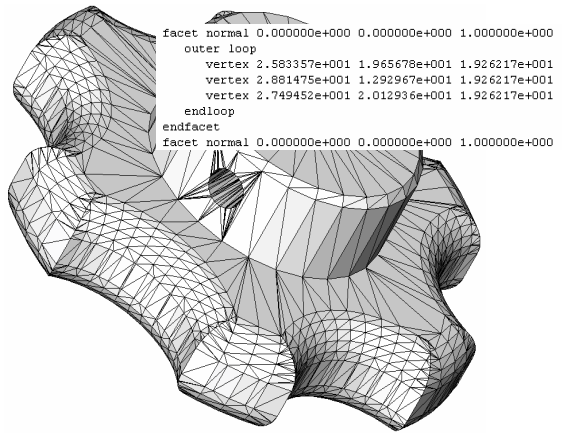


图 1.3 STL 模型及其文件格式
Fig 1.3 STL model and it's format

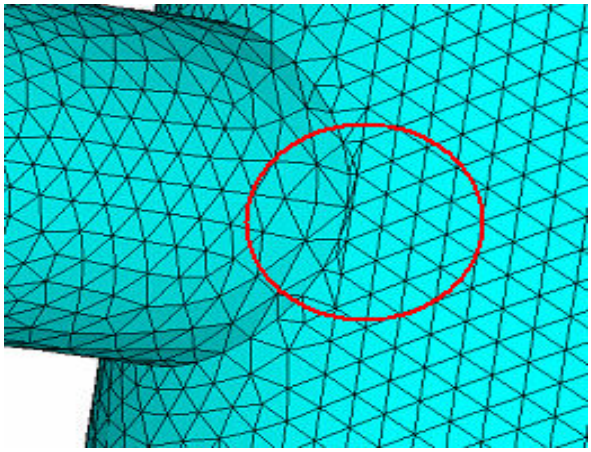


图 1.4 虚边界问题
Fig.1.4 Problem of virtual boundary

1.2.4 曲面有限元网格生成

在有限元网格生成的研究中，曲面的自适应有限元网格生成占有重要的地位，同时也是该领域的一个难点和热点[60, 61, 63, 64, 66, 67, 82-84]。这是因为工程结构中常用的薄壳结构，如球罐、压力容器、冷凝塔、飞机蒙皮、汽车外壳等，都是由圆柱、圆锥、球面等规则曲面以及 Bézier、NURBS 等自由曲面组合而成的，另外曲面网格又是三维实体网格生成的前提和基础。经过不懈努力，曲面网格生成方法研究在近十年来取得了许多重要进展。通常，曲面网格生成方法可以分为三大类：直接法、映射法以及基于 STL 文件的方法。基于 STL 文件的方法上文已经介绍，这里主要介绍直接法和映射法。

直接法[51, 54, 85]是指直接在曲面上生成网格，而无需使用曲面的参数域。Lau 和 Lo[50, 85]采用波前推进法(AFT-Advancing Front Technique)对任意曲面进行直接三角剖分，并详细地描述了如何将三角形单元转化为四边形单元。然而，为了确定前沿推进的方向，该方法需要求出曲面的法向和切向；单元的理想点确定以后，为了使点落到曲面上，还需要多次的投影操作；并且，AFT 方法所必需的各种单元有效性判断也是比较费

时的。因此，直接法的主要问题是程序运行的效率和健壮性，它主要运用在非参数化表达的曲面网格生成上。

映射法是目前曲面网格生成的最流行的方法。这种方法要求待剖分的曲面是一个可以在二维域上用双参数表达的参数化曲面。这样，曲面三维物理域上的点可以和曲面二维参数域上的点建立一一对应关系。这种方法首先使用通用的二维剖分程序在二维参数域生成网格，然后将剖分结果反向映射回曲面的物理空间，从而形成曲面的最终网格。通常，这种直接映射方法对于简单的曲面能够生成质量较高网格，而对于曲率变化剧烈的复杂曲面，由于其映射函数在参数域的两个方向的非正交性，生成的网格的质量往往较差。为此 Lee, C. K[52]提出了一种首先将复杂曲面分解为一系列简单曲面，然后在这些简单曲面上生成网格，最后将这些简单曲面上网格合并形成复杂曲面的最终网格。然而对于一张曲率变化频繁的自由曲面，如何分解为一系列简单曲面本身也是一个十分复杂的过程。

为了使映射法生成的网格投影到曲面的物理域后保持较高质量，必须在参数域上生成网格时对网格的形状和大小进行很好的控制。实践证明，在网格的形状和大小控制方面，采用黎曼度量的方法是一个简单有效的方法。并且，在仔细定义待剖分的黎曼度量后,就可以很好地控制待剖分的网格生成，不但可以按用户指定的尺寸得到各向同性网格，而且可以根据需要生成各向异性网格。另外，黎曼度量方法是一个通用方法，它不但适用于曲面网格的剖分，对于三维实体剖分同样适用。

曲面网格生成的另外一个难点是周期性曲面的网格生成，周期性曲面是指其参数域在 U 向或 V 向或 UV 两个方向是周期性的，也就是在有周期性的方向上，曲面的参数域是不封闭的。对这样的周期性参数曲面，为了在参数域上能够使用二维的剖分程序对参数域进行剖分，文献上一般需要加上所谓的“虚边界”。但是如果虚边界周围有“洞”，“洞”周围单元质量往往很差，有时候甚至不能用于有限元分析。图 1.2 是某知名有限元软件对两个相交的圆柱面的剖分结果，可以看出在圆柱面上明显看到一根直线(程序内部添加的虚边界)，在圈出的区域内网格质量较差。再者虚边界的计算往往也是比较复杂的，这就增加了整个剖分程序的复杂性，降低了剖分程序的稳定性。

1.2.5 实体有限元网格生成

在实体网格生成研究中，由于四面体单元具有良好的几何适应性，而几乎成为所有 CAE 软件的默认首选实体网格剖分类型[86]。实体网格的另外一种类型是六面体单元，但复杂三维实体的全六面体单元网格全自动生成问题始终未能获得真正意义上的解决，全六面体网格由此被称为“神圣网格”（holy grid）[87]。近几年来，全六面体网格自动生成研究取得了一些研究进展。目前有代表性的全六面体网格自动生成方法有：原型

法，映射法，扫描法，基于栅格法，扩展的 AFT 方法和多子区域法。Blacker 总结并且扫描法[88]，用这种方法只能生成形状简单的三维网格，且主要依靠人机交互，因而自动化程度较低；Weiler 和 Schneiders 提出了采用同构技术(Isomorphicsm Technique)改进了八叉树法[26]。这种方法实体内部网格好，但是实体的边界网格不是很好，网格质量有方向性；Li 和 Armstron[89]的提出了中点法，把简单几何体分解为全六面体网格。但是这些简单几何体必须首先满足一些拓扑条件，比如欧拉公式；M.A.Price 和 C.G.Armstrong[90]提出了中面法，这种算法的主要缺点是中面的不稳定性，几何体的微小变化就会导致中面的变化，从而整个几何体的网格随之改变；粘贴算法(Plastering)[91]和编须算法（whisker weaving）[92]都是波前推进法(Advancing Front Method AFT)的扩展形式。在网格粘贴的时候识别交叉的面和考虑已经存在节点的连接形式和面的缝合形式，但是当前沿推进的时候可能会产生复杂的内部孔洞，并且这些孔洞不可完全用六面体来填充，也就是说会留下不可剖分的孔洞。粘贴算法基于局部几何测试来推进网格，对于复杂的几何实体很难闭合网格内部连通性。总之，复杂三维域全六面体网格的自动生成是有限元网格生成算法研究中的难点问题，并且始终未能获得真正意义上的解决，它的研究解决对计算几何与计算数学都具有重要的理论价值。

本文的实体网格剖分部分主要研究了四面体剖分的 AFT 算法，关于应用三维 AFT 方法实现四面体的剖分过程中的一些问题和难点在本文第 8 章将详细介绍，这里就不再赘述。

1.2.6 自适应网格生成方法

自适应有限元网格生成方法主要有三类，分别为 *p* 方法，*r* 方法和 *h* 方法。*p* 方法通过增加单元的多项式插值阶数提高求解精度。*r* 方法既不提高单元的插值阶数，也不改变节点总数，它是通过改变节点的位置达到反映结构特征的目的，这种方法作为网格生成后处理阶段的网格光顺过程与 *h* 方法共同使用可以达到很好的效果。*h* 方法通过改变节点数目和单元尺寸达到提高网格求解精度目的。这三种方法在进行自适应有限元网格生成时可以结合运用，但都以 *h* 方法为主。当 *h* 方法与 *p* 方法结合时，称为 *hp* 方法；当 *h* 方法与 *r* 方法结合时，称为 *hr* 方法。

在这里我们主要讨论 *hr* 方法，也就是通过增加和减少节点和单元的数目以及通过改变节点的位置来产生尺寸梯度变化均匀的网格[93, 94]。产生这种尺寸梯度变化均匀的自适应网格的最重要的一步是在剖分域中指定网格的尺寸和形状。尺寸函数(Node Space Function, NSF)[50, 95]和黎曼度量法[37, 49, 53, 59, 94, 96-104]是两种常用的方法，黎曼度量法也可以认为是一种空间转化法，它把目标空间首先转化为黎曼空间，在黎曼空间

产生单位网格，也就是在考虑黎曼度量的情况下网格中每条边的长度接近于 1。由于黎曼度量不但能够控制单元的尺寸，还能够控制单元生成的方向，而几乎成为一种标准的网格自适应控制方式。自适应源也是多方面的，主要由几何(比如曲线/曲面的曲率、剖分域中几何元素间的近亲关系[105-107])、计算结果[96, 97, 108] 以及用户指定等。自适应有限元网格生成方法同自适应有限元分析关系紧密，它是自适应有限元分析的重要组成部分。对于计算自适应，是以计算结果的误差判据为依据，其关键问题是如何建立有限元分析精度与有限元网格疏密程度之间关系，其核心是有限元分析结果的误差准则问题。这部分的工作很大部分和有限元分析程序的工作密切相关，这方面 Zienkiewicz, Zhu, Peraire[109-111]等人的研究工作具有代表性。对自适应网格程序来说，当建立了有限元分析精度与有限元网格疏密程度之间关系后，就可以把这种计算结果的自适应源和剖分域几何自适应源以及用户指定自适应源同等对待，一般来说可以对这多种自适应源求得的交集作为自适应网格生成标准来生成网格，具体可以参照第 4 章和第 6 章的相关介绍。

2. 有限元网格剖分程序的整体框架

2.1 引言

软件开发的目标是使软件具备很强的自适应性、互操作性、可扩展性和可重用性。从上世纪 80 年代起，面向对象技术在软件开发领域广泛兴起。面向对象技术以对象作为最基本的元素，将软件系统看成是离散对象的集合。一个对象既包括数据，也包括行为。面向对象的优点在于，它能让分析者、设计者及用户更清楚地表达概念，相互交流；同时，它作为描述、分析和软件开发的一种手段，大大提高了软件的易读性、可维护性、可重用性；使得软件分析到软件设计的过渡非常自然，因而可以显著降低软件开发的成本。另外，面向对象技术中的继承、封装、多态等机制，直接为软件重用提供了进一步的支持。

随着抽象技术类型和面向对象技术的出现，体系结构的研究逐渐得到了重视。这是因为：对象封装降低了模块间的耦合，为构建层次上的软件复用提供了可能。现在软件体系结构逐渐成为软件工程的重要研究领域。一般来说，可复用面向对象软件系统为三大类：应用程序、工具箱和框架(Framework)。

框架通常定义了应用体系的整体结构类和对象的关系等等设计参数，以便于具体应用实现者能集中精力于应用本身的特定细节。框架主要记录软件应用中共同的设计决策，框架强调设计复用，框架设计中必然要使用设计模式。

设计模式（Design pattern）[112, 113]是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。另外，设计模式有助于对框架结构的理解，成熟的框架通常使用了多种设计模式。

这部分的内容主要介绍有限元生成程序的整体框架，应用程序是具体算法需要解决的工作。在框架的设计中应用一些设计模式，这里也给与简单的介绍。

2.2 软件设计的应当遵循的一些基本原则

Martin, R. C.[114]介绍了现代软件设计和实现过程中应当遵循的一些原则，如"开、闭"原则(Open Closed Principal)、里氏代换原则、合成复用原则等等。在软件设计开发中遵行这些原则，可以达到了代码复用、增加可维护性的目的。这里对这些原则给与简单的总结，在框架设计中将体现这些原则。

(1) "开-闭"原则：模块应对扩展开放，而对修改关闭(Software entities should be open for extension,but closed for modification)。也就是说在软件设计和实现的时候，模块应尽量在不修改原先代码的情况下进行扩展。所有的软件系统都有一个共同的性质，即对它们的需求会随着时间的推移而发生变化。在软件系统面临新的变化的时候，系统的设计必须是稳定的。满足"开-闭"原则的设计可以使软件：a) 通过扩展已有的软件系统，提供新的行为，满足对软件新的需求，使变化中的软件具有一定的适应性和灵活性；b) 已有的软件模块，特别是重要的抽象层模块不能再修改，使得变化中的软件系统具有一定的稳定性和延续性。

(2) 里氏代换原则：一个软件实体如果使用的是基类的话，那么换成子类也完全可以运行，而且它根本不能觉察出基类对象和子类对象之间的区别。里氏代换原则是继承复用的一个基础。只有当子类对象可以替换基类，软件单位的功能不会受到影响时，基类才能真正被复用，子类才能在基类的基础上添加新的行为。

(3) 依赖倒转原则：抽象不应该依赖于细节，细节应当依赖于抽象。传统的面向过程的设计倾向于使高层的模块依赖于底层的模块；抽象层次依赖于具体层次。依赖倒转原则就是要把这个错误的依赖关系倒转过来，这就是“依赖倒转原则”的缘由。这是因为抽象层是应用系统宏观的、表达了对整个系统重要的、战略性的决定；具体层次是一些次要的、与实现有关的算法和逻辑，以及战术性的决定，带有相当大的偶然性。如果抽象层依赖具体层，就会使许多具体层次的算法细节的变化立即影响到抽象层次的宏观逻辑。导致微观决定宏观、偶然决定必然、战术决定战略的行为。

(4) 接口隔离原则：使用多个专门的接口比使用单一的总接口要好。也就是说一个类对另一个类的依赖性应当建立在最小的接口之上。

(5) 合成复用原则：就是说要少用继承，多用合成关系来实现。需要功能扩展时在新的对象里使用已有对象，使之成为新对象的一部分；新对象通过向这些对象委派达到复用已有功能的目的。

(6) 迪米特法则：也叫最少知识原则。也就是说一个对象应当对其他对象有尽可能少的了解。这个原则的根本目的是减小对象间的耦合关系，如果两个类不彼此通信的话，那么这两个类就没有必要发生直接的相互作用。如果一个类需要调用另一个类的方法的话，可以同过第三者转达这个调用。

上述原则是面向对象程序设计中应该遵循的最基本的准则，当然还有其他一些更为具体的原则如：单一职责原则(SRP)、重用发布等价原则(REP)、共同封闭原则(CCP)、 共同重用原则(CRP)、无环依赖原则(ADP)、稳定依赖原则(SDP)、稳定抽象原则(SAP)等等，很多面向对象的参考文献对此都有介绍，这里就不一一叙述了。

2.3 功能与设计要求

几何对于有限元网格剖分程序来说是非常重要的，因为它既是待分析域的初始描述，又是网格细化及重剖分的重要参照。实际上，几何信息的描述和获取也是网格剖分程序的一个瓶颈。现在的几何造型技术已经相当成熟，一般来说商业的 CAD 都会提供二次开发接口来让用户进行增值开发，然而不同的商业软件提供的二次开发接口是不同的，这样当需要网格剖分程序在这些不同的平台下运行时就显得比较困难。

B-Rep（Boundary Representations）作为一种标准，几乎被所有的商业 CAD 软件用来描述几何模型的特征信息。B-Rep 主要包含了模型的几何、拓扑(几何元素间的连接信息)以及模型的精度等三方面的信息。这些信息对网格剖分程序来说都是相当重要的，它成为 CAD 软件和网格剖分程序的一个桥梁。但是，即使同一个模型，不同的商业软件的 B-Rep 描述也是不一样的。比如一个常见的差异是，有的商业 CAD 软件支持周期性曲面，而有的商业软件就不支持。不支持周期性曲面的商业 CAD 软件通常通过在周期性曲面上添加边界的办法变成一般曲面。不同的商业软件即使存在这些微小的差异，都会影响到剖分程序的编制。为了让网格剖分程序能够在不同的 CAD 软件间平滑过渡，需要对 B-Rep 描述中特定的接口进行封装，让这个接口成为网格剖分程序和不同的 CAD 软件相互通信的标准。

本文的网格剖分框架主要包含以下三个方面的内容：

(1) 几何模型的 B-Rep 接口，这个接口封装了网格剖分程序所需的几何及几何元素间拓扑连接的信息。同时这个接口还封装了不同 CAD 软件提供的 API 的复杂性和差异性，使得几何和拓扑连接信息以统一的接口供网格剖分程序使用。

(2) 网格生成的一般框架，这里主要针对不同的网格生成应用，给出了网格生成过程的一般框架，框架中给出了通用的默认实现。用户可以针对特定的应用扩展其功能，也可以定制实现，屏蔽默认的实现，达到特定功能的扩展。

(3) 公用的网格生成工具箱，这个工具箱提供诸如向量代数运算、常用矩阵运算、常用的几何判断及运算、通用的优化算法等数值计算工具。为不同的网格生成算法服务。

图 2.1 是本文网格剖分程序的整体框架，在整体上这个框架应该满足下列要求：

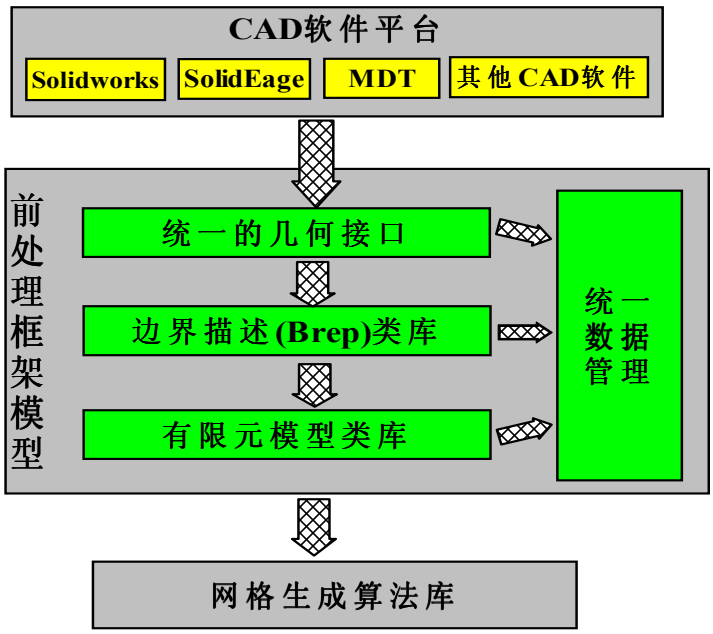


图 2.1 前处理框架的总体设计思路

Fig.2.1 The overall design of pre-process framework

(1) 封装性：也就是说几何模型的 B-Rep 接口不是针对某一个特定 CAD 软件设计的，它只是提出了与网格剖分程序以及其它相关数值模拟程序所必需的几何及拓扑信息接口，不同几何平台的复杂性和差异性通过这个接口来封装。

(2) 相对独立性：作为几何模型的 B-Rep 接口，应该和具体的属性分开。这是因为具体的属性往往表示具体的数据结构，而不同网格剖分算法，其具体的数据结构是不一样的。把接口和属性分开，不但可以减少接口的大小(包括数据本身的大小及接口的数量)，更重要的是这样可以使具体的实现和几何模型的 B-Rep 描述相对独立，这样就可以大大减少网格剖分算法和特定几何模型的 B-Rep 描述之间的耦合，也可以使得在不同平台下的具体实现可以独立的测试，减少开发和维护的成本。

(3) 可扩展性：它作为网格剖分程序的一个基础工具，为网格剖分算法提供几何、拓扑信息的获取，以及其它基础的数值算法。当网格剖分程序提出新的要求时，这个框架也应该扩展相应的功能，以相同的方式为网格剖分算法提供新的服务。

2.4 几何模型的 B-Rep 接口

这部分主要是介绍和网格剖分程序密切关联的几何接口和拓扑信息接口，即几何合拓扑对象的抽象数据类型。

2.4.1 几何接口

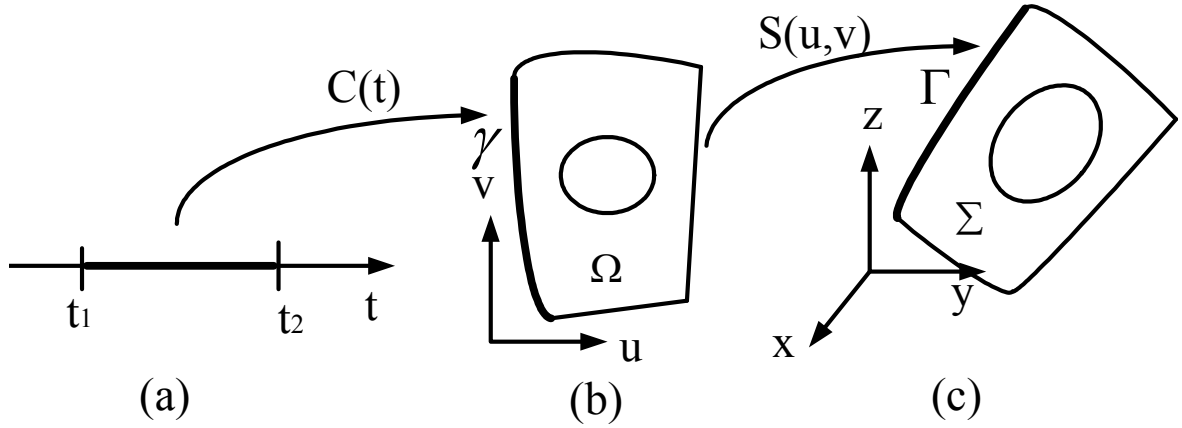


图 2.2 参数曲面与曲线

Fig2.2 Parametric surface and curve

对网格剖分程序来说，参数曲线和参数曲面是两种最重要的几何元素。首先考虑三维空间 R^3 的一张参数曲面 Σ(图 2.2c),它可以用二维空间 R^2 上一个参数空间 Ω (图 2.2c) 的关于 u,v 的一个双参数函数表示：

$$S(u,v)=\begin{bmatrix}x(u,v)\\y(u,v)\\z(u,v)\end{bmatrix}\in\sum\quad\begin{bmatrix}u\\v\end{bmatrix}\in\Omega \tag{2.1}$$

通常曲面的参数空间 Ω 是由一些边界曲线围成，这些边界曲线包括外边界曲线和内边界曲线(如果曲面上存在空洞)， Ω 上每一条边界曲线 γ 和参数曲面 Σ 上的一条三维参数曲线 Γ 对应。 Ω 上的边界曲线可以表示为:

$$C(t)=\begin{bmatrix}u(t)\\v(t)\end{bmatrix}\in\gamma\quad t\in[t_1,t_2] \tag{2.2}$$

有了 $S(u,v)$ 和 $C(t)$，R^3 空间参数曲面 Σ 对应的参数曲线就可以表示为：$S(u(t),v(t))$，$t\in[t_1,t_2]$。 图 2.2 是上述参数曲面 Σ（图 2.2c）、Σ 对应的参数空间 Ω （图 2.2b）以及边界曲线（图 2.2c）的对应示意图。

在曲面自适应剖分算法中为了计算曲率信息，除了知道 $S(u,v)$ 和 $C(t)$ 外还需要知道它们的一阶导数和二阶导数，也就是：$\frac{\partial S(u,v)}{\partial u}$，$\frac{\partial S(u,v)}{\partial v}$，$\frac{\partial C(t)}{\partial t}$ 和 $\frac{\partial S^2(u,v)}{\partial^2 u}$，$\frac{\partial S^2(u,v)}{\partial u\partial v}$，

$\frac{\partial S^2(u,v)}{\partial^2 v}$，$\frac{\partial C^2(t)}{\partial^2 t}$。为了便于网格剖分程序编制针对特定类型的曲线和曲面的加速算法，曲面和曲线的接口中还应当包含曲线的类型(线段、二次曲线、Nurbs 曲线等)和曲面类型信息(平面、二次面、Nurbs 曲面等)，以及曲面和曲线是否有周期性的信息。有了这些关于曲线和曲面的基本信息以后，其它信息可以有这些基本信息计算出来。本文算法程序都是采用面向对象的 C++来实现的。可以这样定义参数曲面的 C++接口：

```
1. class SlGeSurface {
2. public:
3.     virtual void          interval(SlGeInterval& intU, SlGeInterval& intV )const = 0;
4.     virtual SlGePoint3d   eval(double u, double v) const = 0;
5.     virtual void          deriv1(double u, double v,
                                    SlGeVector3d& du,
                                    SlGeVector3d& du) const = 0;
6.     virtual void          deriv2(double u, double v,
                                    SlGeVector3d& duu,
                                    SlGeVector3d& duv,
                                    SlGeVector3d& dvv) const = 0;
7.     virtual GeoType       type()const = 0;
8.     virtual void          isPeriod(bool& inU,double& periodU,
                                      bool& inV,double& periodV)const=0;
9. };
```

这里 SlGeSurface 表示参数曲面对象，SlGeInterval 表示一个区间对象，SlGePoint3d 表示三维空间的一个点对象，SlGeVector3d 表示一个三维向量对象。第 3 行是取得曲面参数空间u,v的范围；第 4 行是曲面的表达式接口，也就是输入一个曲面参数域的u，v值，该方法返回其曲面上对应的R^3坐标；第 4、5 行分别表示曲面的一阶导数和二阶导数。GeoType 是一个关于几何类型的枚举(曲面类型有平面、二次面、Nurbs 曲面等；曲线类型有：线段、二次曲线、Nurbs 等)，第 7 行返回曲面的几何类型；第 8 行返回该曲面 **U** 向或 **V** 向是否是周期性的，如果是周期性曲面，返回相应的周期。

三维空间R^3上的一条参数曲线的 C++接口可以这样定义：

```
1. class SlGeCurve3d {
2. public:
3.    virtual void          interval(SlGeInterval& inv)const = 0;
4.    virtual SlGePoint3d   eval(double t) const = 0;
5.    virtual SlGeVector3d  deriv1(double t) const = 0;
```

```
6.  virtual SlGeVector3d    deriv2 (double t) const = 0;
7.  virtual GeoType         type()const = 0;
8.  virtual bool            isPeriod(double& period)const=0;
9. };
```

SlGeCurve3d 表示上述 R^3 上的参数曲面的边界曲线 Γ。第 3 行取得曲线参数空间的范围；第 4 行曲线的表达式接口，也就是输入一个曲线参数域的 t 值，返回曲线上对应的坐标；第 4、5 行分别表示曲线的一阶导数和二阶导数。第 7 行返回曲线的几何类型；第 8 行返回该曲线是否是周期性的，如果是，返回相应的周期。类似，对于曲面二维参数空间 R^2 上的曲线 γ，C++接口可以定义为:

```
1. class SlGeCurve2d {
2. public:
3.      virtual void    interval(SlGeInterval& intv)const = 0;
4.      virtual SlGePoint2d     eval(double t) const = 0;
5.      virtual SlGeVector2d    deriv1(double t) const = 0;
6.      virtual SlGeVector2d    deriv2 (double t) const = 0;
7.      …
8. };
```

SlGePoint2d 表示二维空间的一个点对象，SlGeVector2d 标示一个二维向量对象，SlGeCurve2d 表示二维参数空间 R^2 上的曲线 γ，其它接口和 SlGeCurve3d 对象的功能类似。

2.4.2 跨平台的几何接口的实现

本文采用设计模式中的桥梁模式(Bridge)和工厂模式(Factory)实现了几何接口的跨平台。这里以参数曲面为例来说明具体的实现过程。

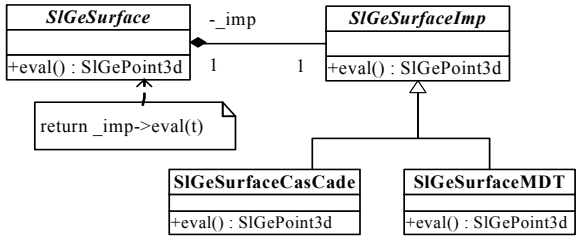


图 2.3 桥梁模式

Fig.2.3 Surface's Bridge

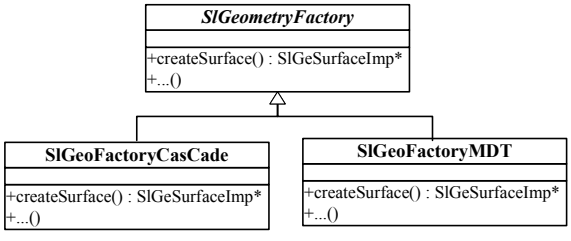


图 2.4 几何类的工厂模式

Fig.2.4 The factory of geometry class

桥梁模式的主要作用是将抽象的数据类型与具体实现脱耦。本文通过接口的方式在传递几何数据，在不同的 CAD 平台下几何引擎是不同的，也就是针对上述几何接口的具体实现是不同的。设计的目标是要让几何接口成为网格生成模块和不同的具体的 CAD

平台通信的中介，使几何接口的具体实现对网格生成模块不再可见。参数化曲面的几何接口和在 Open CASCADE®和 Autodeak MDT®两种平台下具体实现的桥梁模式如图 2.3 所示以 Open CASCADE®为例我们可以这样特化曲面类：

```
1. class SlGeSurfaceCasCade : public SlGeSurface{
2. public:
3.     SlGeSurfaceCasCade(const TopoDS_Face& face);
4.     SlGePoint3d    eval(double u, double v) const;
5.     …
6. private:
7.     Handle_Geom_Surface      _surfacePtr;
8.     SlGeInterval    _intervalU,_intervalV;
9. };
// in the CPP file
10. SlGeSurfaceCasCade:: SlGeSurfaceCasCade (const TopoDS_Face& face){
11.      _surfacePtr = BRep_Tool::Surface(face);
12.      Standard_Real    u0,u1,v0,v1;
13.      BRepTools::UVBounds(face, u0,u1,v0,v1);
14.      _intervalU.set(u0,u1);
16.      _intervalU.set(v0,v1);
17. }
```

在 Open CASCADE 中以 Geom_Surface 表示一个参数曲面，Handle_Geom_Surface 表示指向一个参数曲面的智能指针，但是在这个平台下有的参数曲面(如平面)的参数域是无穷大的，只有加上拓扑面的约束后才和模型中实际面的大小一致。下面给出一个曲面函数 $S(u,v)$ 的实现，其它如一、二阶导数等接口可以用类似的方法实现。曲线接口的实现类似，这里就不再赘述。

```
1. SlGePoint3d    SlGeSurfaceCasCade::eval(double u, double v) const{
2.      gp_Pnt pt = _surfacePtr ->Value(u,v);
3.      return SlGePoint3d (pt.X(),pt.Y(),pt.Z());
      4.}
```

在具体的 CAD 平台下，当需要使用几何对象时，本文通过具体的几何类的工厂来“生产”相应得几何对象，如图 2.4 所示，在 Open CASCADE 平台下，当需要生成曲面对象时，可以用 SlGeoFactoryCasCade 类的 createSurface 方法具体对象的指针。

2.4.3 算法对象的策略模式

有了上述的抽象数据类型以后，就可以针对这些抽象数据类型实现一些常用的算法了。虽然我们可以把针对特定数据类型的算法作为对象本身的方法(行为)来处理，但是这样设计会带来一些问题：首先是算法的可扩展性，当需要扩展算法的功能或者用新的算法替换现有算法的时候就需要修改现有代码或者重新特化一个子类重载该类的相关算法。这就违反了上述的 2.2 节介绍的“开-闭”原则和里氏代换原则。一个比较好的方法是用设计模式中的策略模式来解决这个问题。策略模式是对算法的包装，是使算法的责任和算法本身隔开，委派给不同的对象管理，策略模式通常把一系列的算法包装到一系列的策略类中，作为一个子类，使得它们可以互换。图 2.5 是典型的策略类的 UML 类图。

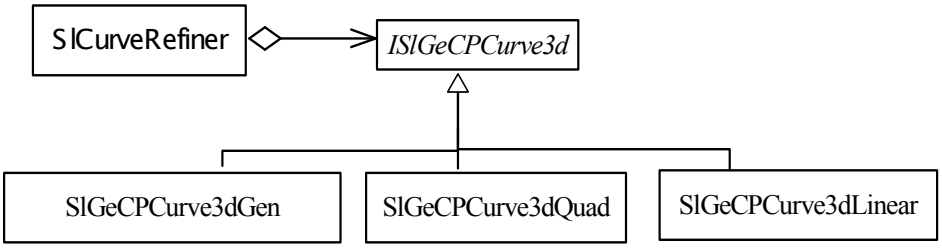


图 2.5 策略模式的 UML 类图

Fig.2.5 The UML architecture of strategy

在上述视图中是一个网格加密算法的曲线上单元边细化应用程序的一个实例。在自适应网格剖分过程中，通常需要对曲面上的网格进行加密，加密后对新增加的点需要投影到相应的曲线或曲面上。这个算法可以抽象为：已知一个点 P 和一条曲线 $C(t)$ 或曲面 $S(u,v)$，求 $C(t)$ 或 $S(u,v)$ 上和 P 距离最近的点 P'。像这样的类似算法可能随着需求的扩展需要不断的补充。为了保证接口的独立性，对于类似的接口一般不直接添加在 $C(t)$ 或 $S(u,v)$ 的接口中，而是设计新的对象来扩充功能。如上述求 $C(t)$ 上和 P 距离最近的点 P' 的算法，本文是这样设计接口的：

```
1. class ISlGeCPCurve3d {
2. public:
3.     ISlGeCPCurve3d (const SlGeCurve3d& curve, const SlGePoint3d&);
4.     virtual  SlGePoint3d  pointOnCurve(double& t)const=0;
5.     …
6. }
```

上述接口的第 3 行是该算法对象的构造函数，通过输入一个曲线和一个点来构造这个对象，第 4 行是根据已有点和曲线，求该曲线上和已知点距离最近的一个点，同时返

回曲线上这个最近点的参数坐标。对这样的已知目标函数及导数的问题，通过调用通用的优化程序就可以得到相应的解，这样的通用解法(在 SlGeCPCurve3dGen 类中实现)只能作为一个默认的方法。一方面对于不同类型的曲线(比如直线，二次曲线等)可能存在更快的解法(SlGeCPCurve3dLinear 和 SlGeCPCurve3dQuad 中实现)，另一方面随着开发的深入，可能找到更好的解，因此这样的算法接口也应该设计成可扩展的。在 SlCurveRefine 类中当需要得到曲线上最近点时，就可以根据曲线的类型，调用特定 ISlGeCPCurve3 的子类算法来完成。其他算法对象的设计与此类似，这里就不一一介绍。

2.4.4 拓扑信息接口

边界表示法（B-Rep）[21, 22]是以物体的边界为基础，定义和描述三维物体几何外形的方法。这种方法给出了物体几何外形的完整、显式的边界描述。用于 CAD 造型的 B-Brep 比较复杂，对网格剖分程序来说其中的很多内容可以进行适当的简化，本文简化后的用于网格剖分的 B-Rep 是：每个物体(SlBrBody)都由有限个面(SlBrface)构成，每个面（平面或曲面）由有限条边(SlBrEdge)围成的有限个封闭域构成，这些封闭区域成为环(SlBrLoop)。这些环又分为外环和内环，外环中所有的边按逆时针排列，内环中所有的边按顺时针排列。边是由两个顶点(SlBrVertex)构成。这样整个 B-Rep 把三维物体的体、面、环、边、点的信息分层记录，并建立层与层之间的关系，如图 2.6 所示。

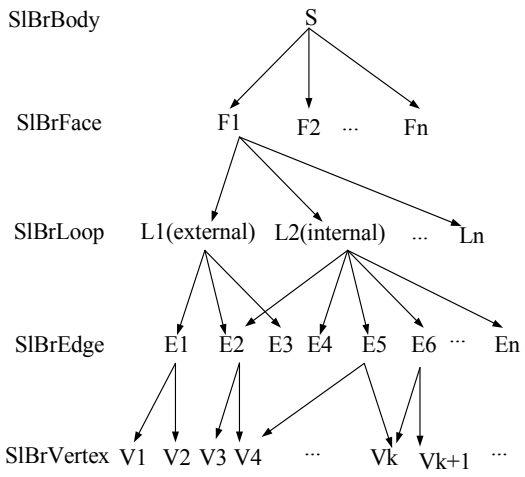


图 2.6 B-Brep 结构

Fig.2.6. Hierarchy of B-Rep

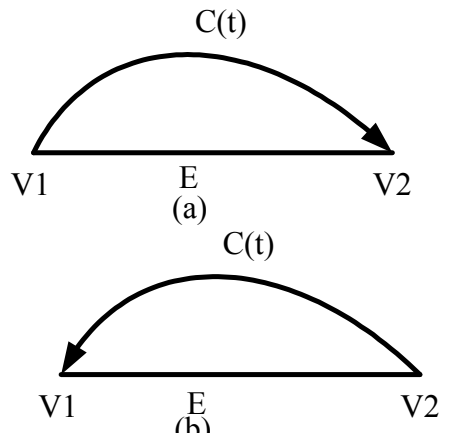


图 2.7 曲线与边的相对方向

Fig.2.7. The orientations of edge and curve

边界表示法一个很重要的特点是在该方法中，描述物体的信息包括几何信息和拓扑信息两个方面。几何信息是指在形体表示中，用于描述边界表示对象的几何性质的数据，如边界对象尺寸、位置、形状等；拓扑信息是指在形体表示中，用来描述表示对象之间的连接关系的数据，如几何体顶点、边、面的数目、类型以及形体上各个顶点之间的线

邻接关系、各条轮廓线之间的邻接关系和各个局部表面之间的邻接关系与线邻接关系等。边界表示法详细记录了构成形体的所有几何信息以及拓扑信息，以便直接存取构成形体的各个面、各条边的边界以及各个顶点的定义参数，有利于以面、边、点为基础的各种几何运算和操作。拓扑元素的作用相当于“粘结剂”，将该模型组合在一起，而几何数据则来自于拓扑元素。打个比方，如果说拓扑关系形成物体边界表示的“骨架”，那么物体的几何信息就犹如附着在这一“骨架”上的“肌肉”。在本文用于网格生成的 B-Rep 中，有三个拓扑对象“粘结”对应的几何对象，这种关联关系可以通过在拓扑对象存储对应的几何对象的引用(C++中的指针)来建立。这三种关联是：

(1) 拓扑面对象(SlBrFace)关联对应的几何参数曲面对象(SlGeSurface)。

(2) 拓扑边对象(SlBrEdge)关联对应的几何参数曲线对象(SlGeCurve3d 或 SlGeCurve2d)。

(3) 拓扑点对象(SlBrVertex)关联对应的几何点对象(SlGePoint3d)。

下面以面向对象的 C++简要介绍一下本文对待剖分模型描述的 B-Rep 的主要接口：

```
1. class    SlBrEntity{
2. public:
3.    bool    isValid() const; /*判断是否是一个有效的对象  */
4.    int      id()const;/*本文以一个全局的整形值(ID)来标志一个对象，返回对象的 ID*/
5. protected:
6.    SlBrEntity(int id);/*构造一个对象必须同时给对象一个 ID, 需要保证 ID 的唯一性*/
7.    …
8. };
```

SlBrEntity 是个基类，其它拓扑对象都是从这个类派生。

```
1. class SlBrVertex : public SlBrEntity{
2. public:
3. SlBrVertex(int id,const SlGePointV3d&);/*拓扑点由一个 ID 和一个空间坐标构成*/
4. …
5. };
```

SlBrVertex 表示一个拓扑点对象，它的基本属性是其位置坐标和标志其唯一性的 ID。

```
1. class SlBrEdge : public SlBrEntity{
2. public:
3.     SlBrEdge(int id,int v1,int v2);
4.     void              setCurve(SlGeCurve3d* curve,bool orientWithEdge);
5.     bool              getCurve(SlGeCurve3d*& curve, bool& orientWithEdge);…
6. };
```

接口第 3 行意思是：这里拓扑边对象(SlBrEdge)是有一个全局的 ID 两个拓扑点的 ID 构成。对于和该拓扑边相关联的几何曲线之间有一个相对方向，如果拓扑边的起始位置和几何曲线的起点是同一点，那么它们的方向是一致的(图 2.7a)，否则是相反的(图 2.7b)，所以在上述代码的第 4 行设置关联曲线时，必须同时设置相对方向；同样代码第 5 行取出关联曲线的同时，也可以取出相对方向。

```
1. class    SlBrLoop : public SlBrEntity{
2. public:
3.      struct LoopEdge{int edgeId;bool orient;}; /*环中的边的结构*/
4.      SlBrLoop(int id);
5.      bool        pushEdge(const LoopEdge& edge);/*往环中添加一条边*/
6.      void        getEdges(SlArray1D<LoopEdge>& edgeArray)const;/*取出环中所有的边*/
7.      ...
8.      };
```

一个环有多条边构成，环本身有一个方向，这样环中的每一条边就有一个相对方向：如果边的起点到终点的方向和环的方向是一致的，那么上述接口 LoopEdge 中 orient 就为 true，否则为 false。如图 2.8 所示，在用上述接口第 5 行往环中添加一条边时，E_j 和环的方向是一致的，对应的 orient 为 true，E_{j+1} 和环的方向是相反的，对应的 orient 为 false。

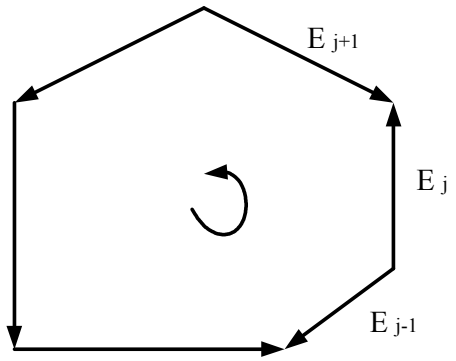


图 2.8 环与边的相对方向

Fig.2.8. The orientations of loop and edges of it

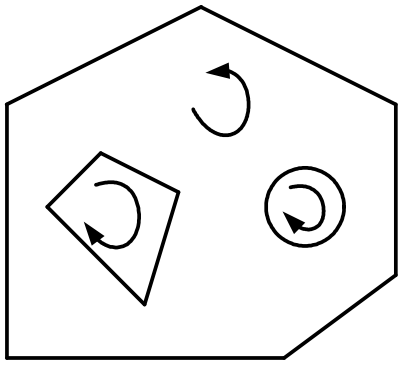


图 2.9 面中环的方向

Fig.2.9. The orientation of loop in face

```
1. class    SlBrFace : public SlBrEntity{
2. public:
    /*环的类型: Exterior 外环；Interior 内环 */
3.    enum LoopType{ Exterior=0,Interior,UnKnown};
4.    struct FaceLoop int loopID ;LoopType type;};/*面环：环的 ID+环的类型*/
```

```
5.  SlBrFace(int id);
6.  void getSurface(SlGeSurface*& surface) const;
/*设置对应的几何面，同时设置相对方向*/
7.  void setSurface(SlGeSurface* surface,bool orient); bool isOrientToSurface()const;
8.  void pushLoop(const FaceLoop&);
9.  void getLoops(SlArray1D<FaceLoop>& loopArray)const;
10.  …
11.};
```

一般来说面中有一个外环和多个内环，这些环都满足左手规则，即外环逆时针方内环顺时针方向(如图 2.9)。本文规定对于实体模型中每一个拓扑面都指向实体的内部，这样几何面和拓扑面就有一个相对方向，如果实体上某点的外法线方向指向实体的外部，那么几何面和拓扑面的相对方向相反。

```
1. class   SlBrBody {
2. public:
3.    SlBrBody(Real eps);
4.    void          append(const SlBrVertex&);   /*添加一个点*/
5.    void          append (const SlBrEdge&);    /*添加一个边*/
6.    void          append (const SlBrLoop&);   /*添加一个环*/
7.    void          append (const SlBrFace&);   /*添加一个面*/
8.    SlBrVertex   getVertex(int id)const;
9.    SlBrEdge      getEdge(int id)const;
10    SlBrLoop     getLoop(int id)const;
11.   SlBrFace      getFace(int id)const;
12    ….
13.};
```

SlBrBody 是个集合类，用来表示整个待剖分域的几何信息和拓扑连接信息，也是网格剖分程序的初始输入。有了以上的几何和拓扑接口后，下面的工作就是怎样根据一个几何模型，来构造一个 B-Rep，也就是根据在特定几何平台上构建的几何模型，通过调用相应的二次开发接口来填充上述数据接口。

2.4.5 B-Rep 的构建

本文参照设计模式中的建造模式，通过一个称为 B-Rep 的构建器来完成 CAD 几何模型信息的提取，这个 B-Rep 的构建器的接口如下：

```
1. class SlBrepBuilder{
2. public:
3.     virtual bool build(SlBrBody&)=0;
4. protected:
5.     SlBreBuilder();
6. …
7. };
```

虽然几乎所有的商业 CAD 软件都有几何模型的 B-Rep 描述，但是不同的 CAD 软件提供提取几何和拓扑信息的接口和过程是不同的，所以很难设计一个通用的过程来完成 B-Rep 的提取。这样，这里设计的 B-Rep 的构建器就提供一个接口，具体的实现需要在不同平台下特化，本文 B-Rep 构建器的继承关系如图 2.10 所示。以 Open CASCADE®* 为例，当需要根据这个平台创建的模型构建 B-Rep 时，就需要设计一个基于这个平台的子类：

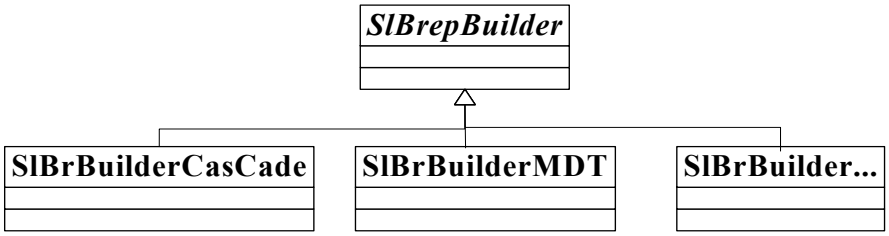


图 2.10 B-Rep 构造器的继承结构

Fig.2.10 Structures of B-Rep Builder

```
1. class SlBrepBuilderCasCade : public SlBrepBuilder {
2. public:
3.     SlBrepBuilderCasCade(const TopoDS_Shape&);
4.     bool build(SlBrBody&);
5. private:
6.     TopoDS_Shape    _theShape;
7. };
```

*一个著名的开源 CAD 造型软件，提供常用的三维实体与曲面造型、模型修复、模型显示、标准格式模型文件(step, iges 等)读取和存贮等常用功能。经过简单的注册就可以获取其全部代码及开发使用手册。由于其平台的开放性，而被世界上很多研究机构用来进行学术研究和商业增值开发，具体信息可以参看其主页：www.opencascade.org

在 Open CASCADE®平台下，TopoDS_Shape 是 B-Rep 类库中，所有 B-Rep 元素的基类，可以通过，它提供一个称为 B-Rep 浏览器的类(TopExp_Explorer)来浏览当前对象下一层次所有的子对象。如在构建 B-Rep 时，可以这样获取模型的中的面信息：

```
1. bool SlBrepBuilderCasCade:: build(SlBrBody&){
2.     TopExp_Explorer exp;
3.     for (exp.Init(_theShape, TopAbs_FACE);exp.More(); exp.Next()){
4.     TopoDS_Face& faceTp=TopoDS::Face(exp.Current())
5.     …
6.     }
7. }
```

同样也可以以类似的方式获取当前面中的环，环中的边等，具体可以参照相应的开发手册。

上面简要介绍了本文用于网格剖分的 B-Rep 接口的设计，并针对 Open CASCADE 平台对具体的实现的关键步骤进行了介绍。其它 CAD 的平台的实现方法与此类似，目前作者已经分别成功地在 AutoDesk MDT、Solidworks、Open CASCADE 三个平台，通过这个接口实现了 CAD 数据信息的获取和组织，并成功应用于网格剖分程序。另外像 Step，Iges 等标准模型文件也可以通过像 Open CASCADE 的开源平台首先转换成对应的模型，然后再使用本文的方法实现网格剖分及数值模拟。

应该说建立上述 B-Rep 接口的主要目的是使得网格剖分以及相关的数值模拟程序不再和某个特定的平台相关，但需要在某个特定平台实现网格剖分时只需要实现上述的几何接口，和 B-Rep 构建器，完成 B-Rep 信息的提取，和参数曲面和曲线的重新特化就可以了，而不需要修改任何网格生成模块的代码，使网格生成模块更容易地作为一个插件来使用。

2.5 网格生成的整体框架

取出 CAD 模型的 B-Rep 数据后，就可以进行网格生成了，网格生成的一般过程是：

(1) 设置(或计算)模型的网格剖分尺寸。

(2) 根据剖分尺寸对模型所有的边界曲线进行离散化。

(3) 遍历模型的所有表面，完成模型表面的网格剖分。

(4) 对于三维实体模型，根据需要生成模型的实体网格。

(5) 输出网格剖分结果到文件或其它模块。

2.5.1 网格尺寸控制

网格尺寸控制，主要包括三个方面：全局尺寸、局部尺寸、自适应尺寸。全局尺寸是通过一个尺寸函数来作用于整个剖分域的；局部尺寸分别通过相关的尺寸函数作用于剖分域的某个几何曲线或曲面上；自适应尺寸一般是由多个尺寸控制源，也可以根据前一次有限元分析的结果计算得到。对于空间中某点的网格尺寸，是这三个方面尺寸的交集。

一般来说用户可以指定全局剖分尺寸，如果用户没有指定，就可以根据模型外包围盒的对角线的长度 L 乘以一个比例因子 α (本文 $\alpha = 1/50$)得到全局剖分尺寸($h = \alpha L$)，最小允许的剖分尺寸 $h_{\min}$ 可以采用类似的计算 $h_{\min} = \beta L$ (本文 $\beta = 1/50$)。

除了全局剖分尺寸外，用户也可以针对特定的曲线或曲面上的某点设置剖分尺寸，默认情况下这些尺寸为 $h = \alpha L$ 。曲线和曲面网格的局部控制的一个重要方面是考虑剖分尺寸的同时，考虑这些曲线/曲面的曲率，这在第 6 章组合曲面剖分中介绍。

2.5.2 网格剖分结果的表示

一般来说网格是由节点和单元的集合构成的，但是不同的网格剖分过程所产生的单元类型是不同的，比如在边界离散化阶段产生的是线单元，在表面剖分阶段产生三角形或四边形面单元，在体剖分阶段产生是四面体单元或六面体单元。另外用户可能有某些特定的需求，比如需要知道剖分结果中某个面或曲线上的所有上的所有节点或单元，或者需要知道某个节点在哪个曲面或曲线上。这样在剖分结果中就必须有剖分过程中每一个过程的结果记录。

本文的剖分结果是用节点和单元的集合表示的，对节点和单元都是用在相应集合中全局唯一的整型值(ID)来标识。节点还另外还附加一个位置属性，这个位置属性分为两个部分：1）节点所关联 B-Rep 元素的类型(Br_Type：Vertex, Edge, Face, Brep)；2）节点所关联 B-Rep 元素的标识(Br_ID)。这样一个节点的基本属性有(图 2.11)：标识、关联 B-Rep 元素的类型(Br_Type)、关联 B-Rep 元素的标识(Br_ID)、位置(Coordinate)。

图 2.11 节点的基本属性
Fig.2.11. The properties of node

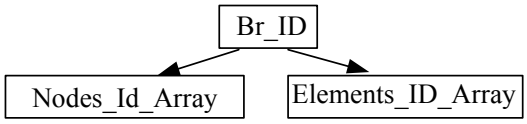


图 2.12 B-Rep 元素和节点与单元标识数组的映射
Fig.2.12 The map of B-Rep entity and nodes and elements'ID array

在网格剖分过程中记录了不同过程产生的节点和单元，这些节点和单元分别和 B-Rep 元素中对应的元素相对应，这种对应关系通过一个映射来实现(图 2.12)。比如在对模型中某一条边(SlBrEdge)的曲线离散化时，会产生新的节点和单元，当更新剖分结

果时，除了把新的节点和单元更新到节点和单元集合时，还需要建立新节点和单元 Id 和边的 Id 的映射关系。在网格剖分的结果中有了上述的信息后，就建立了有限元模型和几何模型间的相互关系，这些关系对自适应网格剖分是非常重要的。

2.5.3 网格生成的整体框架

网格生成的通用过程可以用设计模式中的模版方法来设计。也就是说，首先设计一个抽象的网格生成应用程序类，将部分逻辑以具体的方法以及具体的构造函数实现，并通过声明一些抽象的方法来迫使子类实现剩余的逻辑。这里给出网格生成的通用的过程，具体细节将在第 6 章组合曲面网格生成及实体网格生成中加以介绍。本文用 SlMesherApp 表示网格生成应用程序的模版基类，其设计是这样的:

```
1. class SlMesherApp{
2. public:
3.    virtual    ErrorState    doMesh(SlMeshResult& result); /*网格剖分的主程序*/
4. protected:
5.    virtual    ErrorState constructBRep(SlBrBody& brep)=0;
6.    virtual    ErrorState setupMeshControl(const SlBrBody& brep, SlMeshControl*&);
7.    virtual    ErrorState discreteBoundaryCurves(const SlBrBody&,
                                                   const SlMeshControl&, SlMeshResult&);
8.    virtual    ErrorState doSurfaceMesh(const SlBrBody&,
                                          const SlMeshControl&, SlMeshResult&);
9.    virtual    ErrorState doSolidMesh(const SlBrBody&,
                                          const SlMeshControl&, SlMeshResult&);
10. protected:
11.    SlBrBody              _brep;
12.    SlMeshControl*      _controlPtr;
13. };
```

在此接口中，SlMeshResult 表示网格剖分的结果，SlMeshControl 表示网格剖分控制。对于不同的应用，用户需要设计这个接口的类的子类，来完成 B-Rep 的构建、边界的离散化、面剖分和体剖分。doMesh 方法完成整个剖分过程，在这个接口中给出了默认的实现过程：

```
1. ErrorState    SlMesherApp::doMesh(SlMeshResult& result){
2.      constructBRep(_brep); /*构建 B-Rep*/
3.      setupMeshControl(_brep, _controlPtr); /*取得剖分控制*/
4.      discreteBoundaryCurves(_brep, _controlPtr, result);/*离散化边界曲线*/
```

```
5.    doSurfaceMesh(_brep, _controlPtr, result);/*表面剖分*/
6.    doSolidMesh(_brep, _controlPtr, result);/*实体剖分*/
7.    …
8. }
```

上述 ErrorState 是个错误标志码，在 C++中是个枚举类型，如果程序成功返回，其值为 eOk，否则返回其它具体的错误类型。本文对 discreteBoundaryCurves、doSurfaceMesh、doSolidMesh 的实现是空的，只是有个成功返回的标志(eOk)。

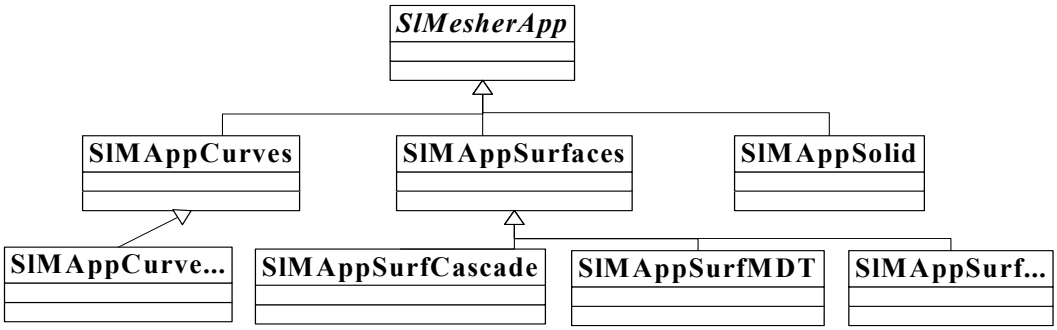


图 2.13 网格剖分应用程序继承关系图
Fig.2.13 Structures of mesh applations

图 2.13 给出了本文网格剖分应用程序的继承关系，SlMAppCurves 表示只是模型的边界的离散化，SlMAppSurfaces 表示模型的曲面剖分，SlMAppSolid 表示模型的实体剖分。并不是所有剖分应用中都需要上述过程，比如在表面剖分应用中就不需要进行实体剖分，这样只要在表面剖分应用程序的实现中，对 doSolidMesh 不做任何的实现即可。在不同的 CAD 平台下，主要的差别是 B-Rep 的构造过程不同，这样也需要特化相应的类(在 Open Cascade 平台下的曲面剖分应用程序类：SlMAppSurfCascade)，调用相应的 B-Rep 构建器构造 B-Rep。

3. 基于拓扑连接的网格数据结构

3.1 网格表述

有限元网格是用节点和单元的集合来表示的，即使最简单的有限单元法的应用也要用到节点和单元及这两个对象的集合。节点-单元这种紧凑型表达虽然简单而且能满足某些应用，但是由于其缺少必要的拓扑连接信息，因而在诸如网格优化、自适应等程序实现如果仅使用这种紧凑型的数据结构，会大大增加实现的复杂度，因为这些程序在需要用到其它连接信息的时候不得不需要建立连接信息表，而很多连接信息的使用频率非常高，所以一般在网格生成中会拓展这种紧凑型数据结构。另外在后处理显示有限元模型的时候为了加快显示速度，需要模型的节点表、边表、面表以及各拓扑对象间的连接信息。为了使网格数据结构能够适应更多具体的应用，本文实现了一种基于拓扑连接的网格数据结构。这种数据结构包括两个方面：拓扑对象和连接关系。下面分别针对面剖分和体剖分两个应用加以介绍。

3.1.1 术语定义

(1) Ω_V：和求解模型V相关的域，$V=G,M$。这里G表示几何模型，M表示网格模型。

(2) $\overline{\Omega_V}$：封闭的求解域。

(3) T/Ω：三角形/四边形单元

(4) Tet/Hex：四面体/六面体单元

(5) V_i^d：模型V中的第 i 个维数是 d 的拓扑元素。在这里 d=0 表示点(Vertex), d=1 表示边(Edge),d=2 为面(Face),如构成四面体单元的三角面，构成六面体单元的四边形；d=3 表示一个空间域(Region),如四面体、六面体等。在拓扑上 d 维的拓扑元素由 d-1 维的元素构成。

(6) $\{V^d\}$：模型V中 d 维拓扑元素的无序集合。

(7) $[V^d]$：模型V中 d 维拓扑元素的有序集合。

(8) $\phi\{V^d\}$：模型V中 d 维拓扑元素的相邻拓扑元素的集合。集合ϕ可能代表一个元素，元素的一个集合，或者整个模型

(9) $V_i^d\{V^q\}$：和拓扑元素V_i^d相邻的维数为 q 的无序拓扑元素的集合。

3.1.2 网格数据结构的要求

(1) **空间和时间：**在设计用于网格生成和有限元分析的数据结构时必须考虑数据结构的时间效率和空间效率。设计一个占用较小内存要求的数据结构对有限单元法来说是至关重要的，因为这意味着使用同样的内存可以生成更多的单元和节点，而单元和节点的数目直接影响到问题的求解精度。而整个网格数据结构占用内存的大小直接依赖于每种拓扑元素占用的内存的大小，这样就必须减少每种拓扑元素所占用的内存。图 3.1 是网格拓扑元素之间可能的连接形式。显然采取这种连接方式，拓扑连接关系的获得是最直接的，速度是$O(1)$，但同时内存需求是很大的，同时连接关系的更新也相当复杂。

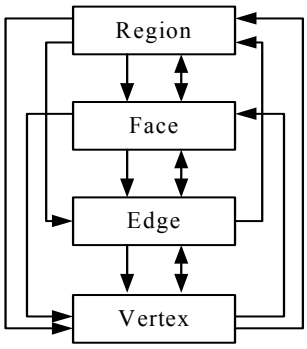


图 3.1 网格拓扑元素及之间可能的连接形式
Fig.3.1 The entities and the possible adjacences

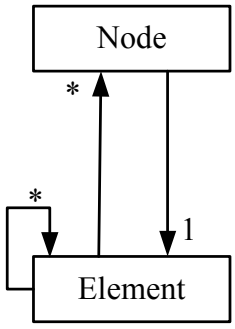


图 3.2 本文紧凑型的节点-单元关系
Fig.3.2 The compact adjacences of node and element

(2) **灵活性：**并不是所有的应用都会用到这些拓扑对象和连接关系的，即使在同一个应用中不同的阶段，所用的拓扑对象和连接关系也是不一样的。比如对于面剖分就用不到 Regin 对象，在四面体剖分中 Face 对象具体应该是三角面，而对于六面体 Face 对象就是四边形了。所以一个通用网格数据结构应该针对不同的应用动态构建相应的拓扑对象及连接关系。

(3) **易用性：**数据结构的应用程序接口(API)应该理解，从而让使用者能够很快掌握使用方法。也就是说应该把实现的复杂性封装在背后，使用者不应该被这些复杂性所迷惑。

3.2 基于拓扑连接的数据结构

构成网格模型M的两个最基本的集合是：节点集(不一定是有序的，用符号$\{N\}$表示)和单元集（用$\{E\}$表示），因而节点N和单元E是网格数据结构中两个最基本的结构。在本文的网格模型M中节点集$\{N\}$和单元集$\{E\}$是显式存在的，也就是说用户可以通过M提供的接口得到这两个集合。节点和单元的关系如图 3.2 所示。

和其它基于拓扑连接的网格数据结构类似，本文也有 0-维拓扑点 v (0-vertex),1-维拓扑边 e (1-edge，由 2 个拓扑点构成), 2-维拓扑面 f (2-face，根据不同的拓扑类型的单元由若干条边构成，三角形单元是有三条边构成，四边形单元是由 4 条边构成)。因为单元本身就代表一个 3-维域(3-regin)，所以这里就不设计这样的数据结构。另外，对于线形单元(不含中间节点的单元), N 和 v 是等价的,即 $v = N$，对于二次单元(含中间节点的单元)， N 和 v 不是是等价的，即 $v \neq N$。网格剖分过程中只是关心所生成网格的拓扑，而不关心是否是二次单元，所以在这里设计 0-维拓扑点 v，代表单元中的某个顶点，所以内存中不需要存储它，需要的时候可以由节点构建。显然对于二维问题，不存在 2-维拓扑面，为了使算法能得到复用，在二维问题中仍然含有拓扑面的概念，但此时面已经退化成边，这样可以由边来构建面。

3.2.1 单元内拓扑元素间的邻接关系

对于节点(N)，通常是由一个节点坐标和一个节点集 $\{N\}$ 内全局唯一的整形标识(ID)构成。对于单元(E)，通常是由其组成的节点的 ID 和一个单元集 $\{E\}$ 内全局唯一的 ID 构成。但针对不同的应用，用户可能需要的连接关系也是不一样的，一般来说拓扑元素之间可能存在如下关系：

(1) 和节点有关的关系：

① $N\{E\}$：和节点相关联的所有单元，无序；

② $N\{N\}$：和节点相关联的所有节点，无序

③ $N\{e\}$：和节点相关联的所有边，无序

④ $N\{f\}$：和节点相关联的所有面，无序

(2) 和单元有关的关系：

① $E[N]$：和单元相关联的所有节点，有序；

② $E[e]$：和单元相关联的所有边，有序

③ $E[f]$：和单元相关联的所有面，有序

④ $E[E]$：和单元相关联的所有单元，有序

(3) 和拓扑边有关的关系：

① $e\{E\}$：和边相关联的所有单元，无序；

② $e\{f\}$：和边相关联的所有面，无序

(5) 和拓扑面相关联的关系

①、 $f[E]$：和面相关联的所有单元，有序

显然，如果数据结构中把这些关系都存起来，主要会存在下面两个方面的问题：首先占用的内存太大，不适合大规模的网格生成算法生成及应用；其次相互之间的关系过于复杂，关系更新算法比较复杂。鉴于以上两个原因，本文的数据结构中并不直接存储拓扑元素间的这些关系，而是在通过在每种单元的内部定义拓扑元素间连接关系，运用相应的算法来推导这些关系。这样就解决了内存和连接关系的更新问题。下面分别介绍面单元和体单元两大类基于拓扑连接的数据结构以及上述关系查找和更新算法。

3.2.2 基于拓扑连接的面单元数据结构

这里的面单元主要是指三角形单元和四边形单元，图 3.3(a)是本文给出的面单元数据结构的表示，这里 Face 是 $d=2$ 的拓扑面，也就是三角形单元或四边形单元，每个 Face 由 3 条(三角形单元)或 4 条（四边形单元）$d=1$ 的 Edge 构成，也就是说每个 Face 包含了 3/4 条 Edge 的引用；每个 Edge 是由两个 $d=0$ 的 Vertex 构成，即包含两个 Vertex 的引用；这里 Vertex 和有限元节点相对应。这种高维元素包含低维元素的数据结构，只是给出了给出了由高到底的一个元素拓扑关系的查找方向。相反，当需要有低维元素查找高维元素时就不方便了。比如在很多应用中需要知道一个 Vertex 连接哪些 Edge 或 Face 等。为此，低维元素必须有一定的所连接的高维元素的信息。本文在 Edge 对象中存储两个所连接面的引用；对于一个 Vertex 可能有多个 Face 与之相连，如果把这个 Vertex 所有的连接的 Face 都存储起来，内存占用是很大的，本文只存储其中任何一个引用，其它的连接关系通过算法计算得到。这里的引用，在 C/C++中可以是一个指针，也可以是标志该元素在对应集合内的全局唯一的正整数。用正整数作为引用，向其它语言移植比较方便。

对于 Face 对象，其中包含组成的三条或四条边的引用，这些边的是按右手规则有序排列的。本文对构成的 Vertex 相同，但排列顺序不同的 Edge 认为是同一个 Edge。对于 Edge 所连接的两个 Face 也是有序的：第 1 个 Face 的中心总是在该 Edge 的左侧。这里以图 3.3(b)为例，F_i 是三角形单元，有三条 Edge 构成，按顺序分别为：E_i、E_j、E_{i+1}，F_{i-1} 是四边形单元，有四条 Edge 构成，按顺序分别为：E_{j-2}、E_{j-1}、E_i，E_{i+1}。E_i 有两个节点构成 V_i、V_j，它所连接的两个面是 F_i 和 F_{i-1}，对于位于边界上的边，只连接一个面，这样第二个面就为空(空指针，或者为-1)。该例子中 V_i 连接四个 Face，这里可以去其中任何一个作为和 Face 的连接属性。

整个网格(Mesh)是由 Vertex 集合、Edge 集合和 Face 集合构成，在这里每个集合用一个动态数组表示，为了在集合中查找的方便，每个元素都有一个在相应动态数组位置下标的属性（Index）。这样上面所说的引用实际上是一个数组的下标。数组的下标是从

0 或 1 开始连续的，但是有限元中的节点和单元编号(ID)并不是连续的，这样就需要在 Mesh 中建立节点 Index 和 ID 及单元 Index 和 ID 对应关系的映射，图 3.3(c)是整个 Mesh 的基本组成。

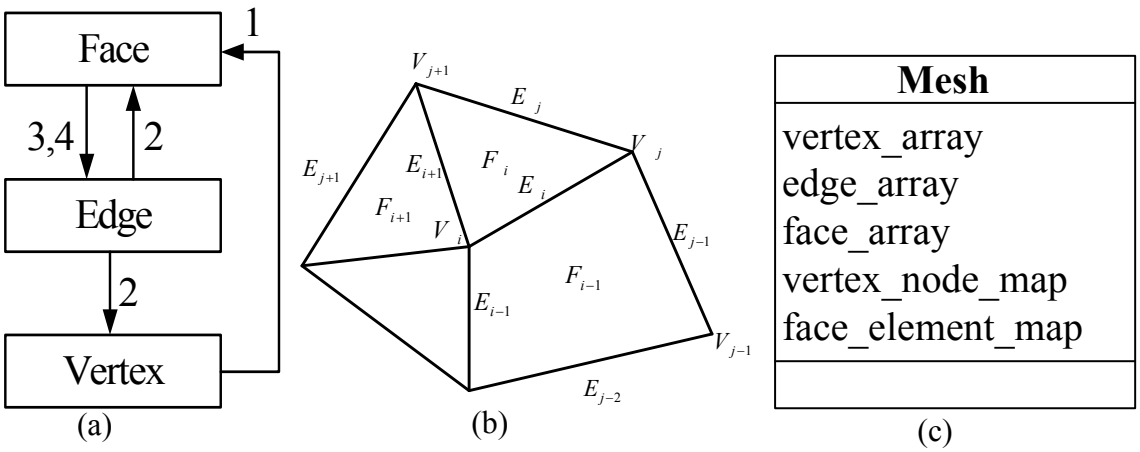


图 3.3 面网格数据结构的表示

Fig.3.3 Mesh data structure representation

3.2.3 面单元连接关系的查找算法

上述拓扑关系中不需要查找就可以得到高维元素所连接的低维元素。比如由 Face 的属性就可以得到所关联的 Edge，由 Edge 的属性也可以直接得到所关联的 Vertex。由低维元素查找高维或者同维元素间的连接关系也比较简单： 由 Edge 的属性也可以直接得到所连接的 Face；Face 和 Face 之间的连接关系也可以通过 Edge 的属性容易得到。这里主要需要解决的是怎样得到 Vertex 所连接的高维元素，比如怎么由 Vertex 知道所连接的所有 Edge 和 Face。如前所述，在每一个 Vertex 中包含有其所连接的任何一个 Face 的引用，这里以这个 Face 的引用为起点，查找该 Vertex(V_i)所连接的所有 Face，该算法是个递归过程，具体过程如下：

(1) 设已访问的 Face 的 Index 集合为 ΛI_F

(2) 取出 V_i 所连接的 Face，作为当前 F_c

① F_c 的 Index： I_{Fc} 放入 ΛI_F 中

② 取出 F_c 中包含 V_i 的两条边 E_{Ci} $(i=0,1)$

③ 循环遍历两条边 E_{Ci} $(i=0,1)$，对当前边 E_{Ci} 执行下列操作：

(a) 取出 E_{Ci} 连接的两个面 F_{Ci} $(i=0,1)$，

(b) 如果 F_{Ci} 的 Index: $I_{Fc} \neq I_{F_{C0}}$，使 $I_{F_c} = I_{F_{C0}}$ 否则 $I_{Fc} = I_{F_{C1}}$

(c) 如果 I_{Fc} 不在集合 ΛI_F 中，重复执行①，②，③

(3) 返回 ΛI_F

上述算法稍加改动就可以得到当前 Vertex(V_i)所连接的所有 Edge，具体过程如下：

(1) 设已访问的 Edge 和 Face 的 Index 集合分别为 ΛI_E， ΛI_F

(2) 取出 V_i 所连接的 Face，作为当前 F_c

① F_c 的 Index：I_{Fc} 放入 ΛI_F 中

② 取出 F_c 中包含 V_i 的两条边 E_{Ci} $(i=0,1)$

③ 循环遍历两条边 E_{Ci} $(i=0,1)$，对当前边 E_{Ci} 执行下列操作：

(a) 将 E_{Ci} 的 Index: I_{Eci} 放入 ΛI_E，并取出 E_{Ci} 连接的两个面 F_{Ci} $(i=0,1)$，

(b) 如果 F_{Ci} 的 Index: $I_{Fc} \neq I_{FC0}$，使 $I_{F_c} = I_{FC0}$ 否则 $I_{Fc} = I_{FC1}$

(c) 如果 I_{Fc} 不在集合 ΛI_F 中，重复执行①，②，③

(3) 返回 ΛI_E

因为由 Edge 得到所连接 Vertex 是直接的，所以上述算法很容易演化一个 Vertex 所连接的所有 Vertex 的算法，这里就不再赘述。

3.2.4 基于拓扑连接的体单元数据结构

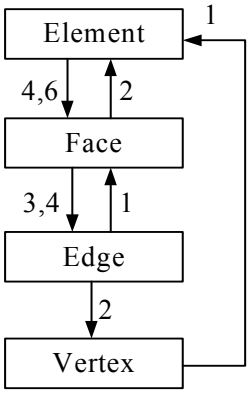


图 3.4 四面体的邻接关系

Fig.3.4 The adjacences of tethedral

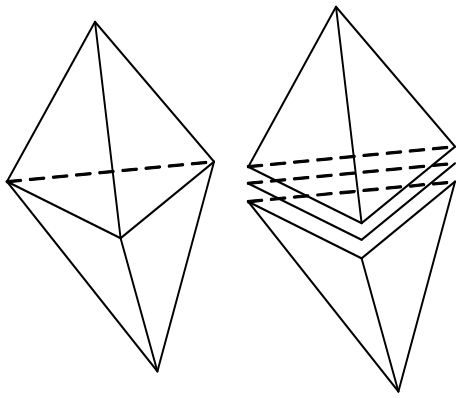


图 3.5 一个面被两个单元共享

Fig.3.5 A face is shared by two elements

在这里体单元主要是指四面体单元和六面体单元，这里以线形四面体单元($Tet4$)为例，如图 3.4 所示是本文体单元数据结构。在这一网格结构中，四面体(Element, $d=3$)是由 $d=2$ 的四个面(Face)构成(六面体由 6 个面构成)；构成面的是 $d=1$ 的边(Edge)，如果是四面体，面是三角形，有 3 条边构成，对于六面体，面是四边形，有 4 条边构成。一般来说，网格内部的面被两个单元共享(如图 3.5 所示)，这样面中除了存在其构成边的属性外，还存在连接的两个面的引用；边由 $d=0$ 的两个节点构成，在体单元中一条

边可能被多个单元和多个面共享，为了迅速查找边的相关连接信息，本文存储了边连接的任何一个面的连接信息；和面剖分数据结构一样，节点中也存储所连接的任何一个单元的连接信息。整个网格(如图 3.6)通过点数组、边数组、面数组、单元数组来表示。连接信息用元素所在数组的下标表示，这样和面剖分数据结构类似，在网格中还存在两个全局的索引与数组下标的对应关系表。

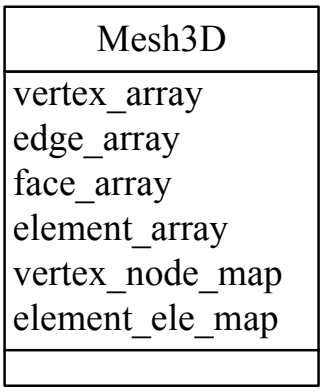


图 3.6 网格模型的组成
Fig.3.6 The buildup of mesh

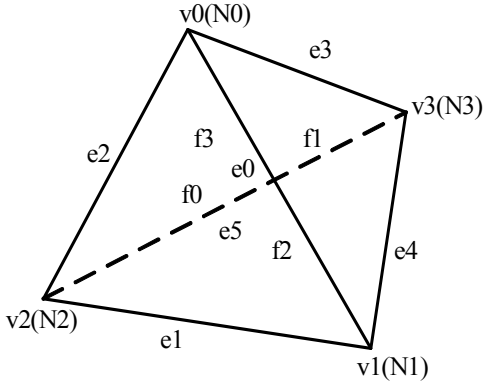


图 3.7 四面体模板
Fig.3.7 The template of tethedral

体剖分数据结构中另外一个表达连接关系的重要方法是模板。以四面体为例，它由 4 个按一定顺序排列的节点构成，在这个四面体中这些节点都有个局部的编号，这样这个四面体单元的节点构成为：$E[v_0,v_1,v_2,v_3]$。也可以说四面体有 4 个拓扑面，或者 6 条拓扑边构成，这样对这个四面体的构成可以形成 3 个模板：$E[v_0,v_1,v_2,v_3]$，$E[e_0,e_1,e_2,e_3,e_4,e_5]$，$E[f_0,f_1,f_2,f_3]$。对于这 3 个模板用 3 个向量表示为：

$V_v^E=[0,1,2,3]$，$V_e^E=[0,1,2,3,4,5]$，$V_f^E=[0,1,2,3]$。 (3.1)

在这里，用V表示n维向量，M表示$n\times m$的矩阵，上标表示所要构成的拓扑元素，下标表示构成这个拓扑的低维拓扑元素。对于构成四面体的拓扑面，根据和其它元素的相关性，同样可以形成 3 个模板，这里以$f0$为例：$f_0[v_0,v_1,v_2]$，$f_0[e_0,e_2,e_1]$，$f_0[f_1,f_2,f_3]$。对于四面体的所有拓扑面这 3 个模板用 3 个 4X3 的矩阵表示为：

$$M_v^f=\begin{bmatrix}0&2&1\\0&1&3\\1&2&3\\0&3&2\end{bmatrix},\quad M_e^f=\begin{bmatrix}0&2&1\\0&4&3\\1&5&4\\2&3&5\end{bmatrix},\quad M_f^f=\begin{bmatrix}1&2&3\\0&3&2\\0&1&3\\0&2&1\end{bmatrix} \tag{3.2}$$

对于构成四面体的拓扑边，同样根据和其它元素的相关性，同样可以形成 3 个模板，这三个模板用矩阵表示为：

$$M_v^{e\ T}=\begin{bmatrix}0&1&2&0&3&2\\1&2&0&3&1&3\end{bmatrix}$$
$$M_f^{e\ T}=\begin{bmatrix}0&0&0&1&1&2\\1&2&3&3&2&3\end{bmatrix} \quad (3.3)$$
$$M_e^{e\ T}=\begin{bmatrix}2&0&1&4&5&3\\4&5&3&2&0&1\end{bmatrix}$$

对于其它拓扑类型的单元可以用类似的方法构建单元内拓扑元素之间的关系。需要指出的是上述关系向量和关系矩阵并不存在每一单元中，而是所有同种拓扑类型的单元共享这些矩阵，因而对于大规模的有限元计算，这些拓扑关系向量和关系矩阵所占的内存是可以忽略不计的。比如对于*Tet*4单元，关系矩阵M_v^f表示四面体单元中三角面的拓扑点构成，用面向对象的C++可以在可以这样表示这个关系矩阵：

```
static int tet_face_vertex_template[4][3]=
{{0,1,2},{0,3,1},{1,3,2},{0,2,3}};//四面体面和点的关系矩阵
```

对于单元中的其它模板，可以用类似的方法实现。

3.2.5 体单元连接关系的查找算法

体单元连接关系的查找算法和面剖分数据结构类似，这里也是不需要查找就可以得到高维元素所连接的低维元素。主要需要解决的是怎样由低维元素查找高维元素以及同维元素之间连接关系的查找。比如由 Edge 查找所关联的面或单元；由 Vertex 查找所关联的 Edge、Face、Element 等。这里以 Vertex 说明这种查找过程，这个过程和面单元数据结构中的查找算法类似。如前所述，在 Vertex 中包含有所连接的任何一个 Element 的引用，这里可以以这个 Element 为起点，查找当前 Vertex(V_i)所连接的所有 Element，该算法是个递归过程，具体过程如下：

(1) 设已访问的 Element 的 Index 集合为ΛI_E

(2) 取出V_i所连接的 Element，作为当前E_c

① E_c的 Index：I_{EC}放入ΛI_E中

② 取出E_c中包含V_i的三个面$f_{ci}, i=0...2$

③ 循环遍历三个面$f_{ci}, i=0...2$，对当前面f_{ci}执行下列操作：

(a) 取出f_{ci}连接的两个单元$E_{ci}, i=0,1$，

(b) 如果E_{ci}的 Index: $I_{EC}\neq I_{E_{c0}}$，使$I_{EC}=I_{E_{c0}}$否则$I_{EC}=I_{E_{c1}}$

(c) 如果I_{EC}不在集合ΛI_E中，重复执行①，②，③

(3) 返回 ΛI_E

上述算法稍加改动就可以得到当前 Vertex(V_i)所连接的所有 Face 和 Edge。以 Face 为例，具体过程如下：

(1) 设已访问的 Face 和 Element 的 Index 集合分别为 ΛI_f， ΛI_E

(2) 取出 V_i 所连接的 Element，作为当前 E_c

① E_c 的 Index： I_{EC} 放入 ΛI_E 中

② 取出 E_c 中包含 V_i 的三个面 $f_{ci}, i = 0...2$

③ 循环遍历三个面 $f_{ci}, i = 0...2$，对当前面 f_{ci} 执行下列操作：

(a) ***将 f_{ci} 的 Index 放到 ΛI_f 中***，取出 f_{ci} 连接的两个单元 $E_{ci}, i = 0,1$，

(b) 如果 E_{ci} 的 Index: $I_{EC} \neq I_{E_{c0}}$，使 $I_{EC} = I_{E_{c0}}$ 否则 $I_{EC} = I_{E_{c1}}$

(c) 如果 I_{EC} 不在集合 ΛI_E 中，重复执行①，②，③

(3) 返回 ΛI_f

其它连接关系的查找算法于此类似，就不再赘述。在 C++中上述的集合可以直接采用标准模版库(C++ standard template library, STL)[115]中的 std::set 来实现。

4. 基于黎曼度量的二维自适应网格生成

4.1 黎曼度量的定义

目前在自适应网格生成过程中，采用黎曼度量来表示待剖分域中某点的网格尺寸和形状几乎成为一种通用的标准方法。许多文献[8, 37, 41, 49, 53, 56, 59, 78, 94, 98-101, 116-119]对此都有论述，这里对这些文献论述的内容进行一个简单总结，在实际实现过程发现其中的一些遗漏和不足，在这里也一并进行补充。

4.1.1 二维空间中的黎曼度量

在二维自适应网格剖分和三维曲面参数域的自适应剖分中，通常用一个2×2的度量矩阵来表示某点网格的尺寸和形状，这个度量矩阵在微分几何中称为黎曼度量。在二维空间Ω中点P的黎曼度量可有一个控制椭圆来表示，这个控制椭圆由 3 个量决定：长轴h_1、短轴h_2及角度θ（如图 4.1 所示），表示为：

$$\mathbf{M}(P)=\mathbf{R}\wedge\mathbf{R}^T=\begin{bmatrix}\cos\theta & -\sin\theta\\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}1/h_1^{\,2} & 0\\ 0 & 1/h_2^{\,2}\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta\\ -\sin\theta & \cos\theta\end{bmatrix} \tag{4.1}$$

$M(P)$是个2×2的实对称阵，也可表示为：

$$\mathbf{M}(P)=\begin{bmatrix}E & F\\ F & G\end{bmatrix},\ E>0,\ G>0,\ EG-F^2>0 \tag{4.2}$$

$M(P)$的两个特征值: $\lambda_1,\lambda_2>0$。设和λ_1,λ_2对应的特征向量是e_1,e_2,则有：

$$\mathbf{M}(P)=[\mathbf{e_1},\mathbf{e_2}]\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}[\mathbf{e_1},\mathbf{e_2}]^T=[\mathbf{e_1},\mathbf{e_2}]\begin{bmatrix}1/h_1^{\,2} & 0\\ 0 & 1/h_2^{\,2}\end{bmatrix}[\mathbf{e_1},\mathbf{e_2}]^T \tag{4.3}$$

Ω中所有点的黎曼度量构成一个黎曼度量场，也叫黎曼空间Σ。在各向同性的情况下，点P的黎曼度量可简化表示为：

$$\mathbf{M}(P)=\frac{1}{h^2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}=\lambda\mathbf{I} \tag{4.4}$$

此时，图 4.1 的控制椭圆退化成中心在P，半径为 h 的圆。

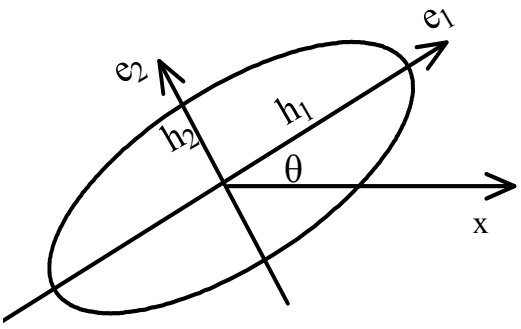


图 4.1 黎曼度量确定的控制椭圆

Fig.4.1. Ellipse defined by metric tensor

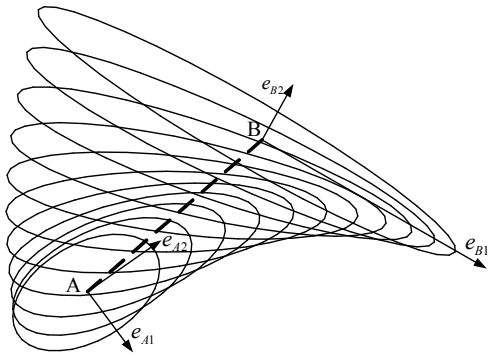


图 4.2 各向异性的黎曼度量的插值

Fig.4.2 Interpolation of anisotropic metrics

4.2 黎曼度量的插值和相交计算

4.2.1 线段上黎曼度量的插值计算

如上所述，Ω 中点 P 的黎曼度量是由一个椭圆控制，那么线段上黎曼度量的插值就要解决这样的问题：如果已知线段两个端点的黎曼度量，怎样插值计算得到线段上任意一点的黎曼度量，使在这条线段上控制椭圆的变化是单调均匀变化的？

在各向同性的情况下，因为控制椭圆退化为圆，这样插值方法就很简单。设线段 AB，在端点处的黎曼度量为：$\lambda_A\mathbf{I}$，$\lambda_B\mathbf{I}$ ，也就是说在 A、B 两点处的尺寸为：$h_A=1/\sqrt{\lambda_A}$，$h_B=1/\sqrt{\lambda_B}$，这样在线段上任意一点的黎曼度量为：

$$\mathbf{M}(A+t\overrightarrow{AB})=\frac{1}{(h_A+t(h_B-h_A))^2}\mathbf{I} \tag{4.5}$$

在各向异性的情况下，Human Borouchaki 等[37]介绍了一种称为联立插值的方法。实际上由图 4.1 可以看出，只要对椭圆的长短轴和角度 θ 分别采用线性插值，再根据公式(4.1)就可以计算得到最后的结果[98]，具体方法如下：设在线段 AB 两端点处的黎曼度量分别为 $\mathbf{M}_A$、$\mathbf{M}_B$，特征向量和特征值分别为 $(\mathbf{e}_{Ai},1/h_{Ai}^{\ 2})$，$(\mathbf{e}_{Bi},1/h_{Bi}^{\ 2})$，$i=1,\ 2$。设 x 轴向量为 $e_x=[1,0]$，那么：

$$\begin{aligned}
&\theta(t)=\theta_A+t(\theta_B-\theta_A)\\
&h_i(t)=h_{Ai}+t(h_{Bi}-h_{Ai})\\
&\theta_A=\angle\mathbf{e_{A1}e_x}\\
&\theta_B=\begin{cases}\angle\mathbf{e_{B1}e_x} & when\ \mathbf{e_{A1}}.\mathbf{e_{B1}}>0\\ \pi-\angle\mathbf{e_{B1}e_x} & when\ \mathbf{e_{A1}}.\mathbf{e_{B1}}<0\end{cases}
\end{aligned} \tag{4.6}$$

这样线段上任意一点的黎曼度量就可以同过把上述公式代入 4.1 式计算得到。图 4.2 是按上述插值方法得到的沿线段 AB 上各点的插值椭圆。

4.2.2 四边形域内点的黎曼度量插值方法

对于如图 3 所示的四边形区域，这里假设已知 4 个顶点 P1~P4 的黎曼度量，对于在四边形区域内的点 P 的黎曼度量可以用四个点的黎曼度量双线性插值得到[120]。具体步骤如图 4.3：首先用上述方法得到点 Q1 在线段 P1,P2 的插值结果,然后得到点 Q2 在线段 P3,P4 的插值结果的，最后由线段 Q1Q2 插值得到点 P 的黎曼度量 $\mathbf{M}_P$。

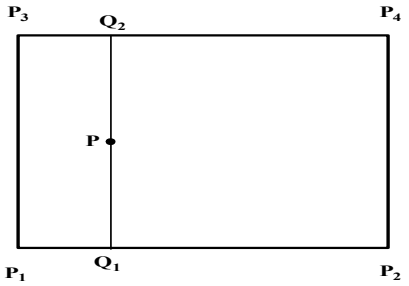


图 4.3 四边形插值方法

Fig.4.3. Interpolation of a quadrilateral

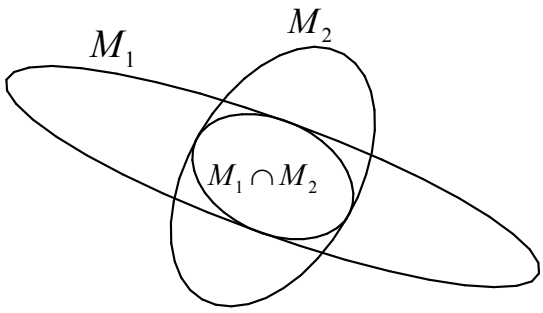


图 4.4 黎曼度量的相交运算

Fig.4.4. The intersection of metric

4.2.3 黎曼度量的相交计算

对于待剖分域的同一个点可能因为不同的自适应要求计算得到不同的黎曼度量，比如对同一个问题可能存在计算自适应、几何自适应以及用户指定的自适应等。在这里统称这些不同的自适应要求为自适应源。同一点的不同自适应源应该进行相交计算，得到一个最终的黎曼度量作为该点的黎曼度量。Human Borouchaki 等[37]给出了黎曼度量相交的计算方法，简述如下：

设空间中某点自适应源 1 和自适应源 2 的黎曼度量为 $\mathbf{M}_1$ 和 $\mathbf{M}_2$，其特征值分别为：(λ_1,λ_2)，(u_1,u_2)。首先构造矩阵：$\mathbf{N}=\mathbf{M_1}^{-1}\mathbf{M_2}$，其也是一个实对称阵，特征向量为 $(\mathbf{e}_1,\mathbf{e}_2)$，那么：

$$\mathbf{M_1}\cap\mathbf{M_2}=(\mathbf{P}^{-1})^T\begin{bmatrix}\max(\lambda_1,u_1) & 0\\ 0 & \max(\lambda_2,u_2)\end{bmatrix}\mathbf{P} \tag{4.7}$$

符号 $\cap$ 表示相交运算，矩阵 $\mathbf{P}$ 的两个列向量为 $(\mathbf{e}_1,\mathbf{e}_2)$。对于多个自适应源可以这样计算：

$$\mathbf{M_1}\cap\ldots\cap\mathbf{M}_n=(\ldots(\mathbf{M_1}\cap\mathbf{M}_2)\cap\mathbf{M}_3)\cap\ldots)\cap\mathbf{M}_n \tag{4.8}$$

图 4.4 是按上述方法得到的两个黎曼度量相交的结果。

4.3 黎曼空间中线段的长度

在黎曼空间$(\Omega,\mathbf{M}(P)_{P\in\Omega})$中，线段$AB=(A+t\overrightarrow{AB})0\le t\le 1$的长度计算公式为：

$$l(AB,\mathbf{M})=\int_0^1\sqrt{\overrightarrow{AB}^T\mathbf{M}(A+t\overrightarrow{AB})\overrightarrow{AB}}dt \tag{4.9a}$$

这里$\overrightarrow{AB}=(u,v)^T$是一个由点$A$到点$B$的向量，$\mathbf{M}(A+t\overrightarrow{AB})$是点$A+t\overrightarrow{AB}$处的黎曼度量。表示为：$\mathbf{M}(A+t\overrightarrow{AB})=\begin{bmatrix}E(t) & G(t)\\ G(t) & F(t)\end{bmatrix}$，这样线段$AB$的长度可以表示为[37]：

$$l(AB,\mathbf{M})=\int_0^1\sqrt{E(t)u^2+2F(t)uv+G(t)v^2}dt \tag{4.9b}$$

在各向同性的情况下上述公式可以简化为：

$$l(AB,\mathbf{M})=\left\|\overrightarrow{AB}\right\|\int_0^1\frac{1}{h(t)} \tag{4.9.c}$$

这里，$h(0)=h(A),h(1)=h(B)$。如果线段AB上黎曼度量处处相等，那么：

$$l(AB,\mathbf{M})=\sqrt{Eu^2+2Fuv+Gv^2} \tag{4.9d}$$

这里假如Ω已被网格化，并且已知网格中所有节点的黎曼度量：$\{\mathbf{M}(x_i),i=0,1,...,n-1\}$，$\{x_i,i=0,1,...,n-1\}$表示网格中的所有节点。假如线段 AB 上黎曼度量不是常量，可以用以下递归算法[120]来计算其长度：

算法 4.1 黎曼空间中线段长度计算

(1). 计算线段AB的长度：$l(AB)=(l(AB,\mathbf{M}_A)+l(AB,\mathbf{M}_B))/2$，$M_A$，$M_B$分别表示 A，B 两个端点的黎曼度量。

(2). 如果$l(AB)>0.5$，取线段 AB 的中点 C，递归调用步骤 1，2 计算$l(AC)$，$l(CB)$。这样$l(AB)=l(AC)+l(CB)$。

(3). 否则，返回$l(AB)$

以上算法可以比较准确地计算线段AB的长度，但是当递归的次数较大(>100)时可能会产生堆栈溢出，所以在实现过程中，当在步骤(1)中计算的长度$l(AB)$较大时，应该把线段 AB 分成若干份，然后分别按上述方法计算其长度后累加得到整个线段的长度。

自适应网格生成的目的是在黎曼空间中产生一个单位网格，也就是是对于网格中任何一个节点P周围区域的边的长度应该满足：

$$\overrightarrow{PX}^T M(P)\overrightarrow{PX}\approx 1 \tag{4.10}$$

PX表示任何一个和点P连接的边，也就是说任何和点P连接的边的边长都是 1 个单位，当网格中所有节点都满足上述条件时，这个网格就称为单位网格。产生全域的单

位网格是非常困难的，很多情况是不可能的。在实际的实现过程中，对网格中任何一条边，考虑黎曼度量情况下，其边长$l_i(M)$满足：$l_i(M)\in[1/\sqrt{2},\sqrt{2}]$即可。

4.4 黎曼空间中两向量的夹角

在黎曼空间中，两个向量$\overrightarrow{PQ}=\vec{u}$，$\overrightarrow{PV}=\vec{v}$，那么以点$P$的黎曼度量$\mathbf{M}$为参照，这两个向量$(\vec{\mathbf{u}},\vec{\mathbf{v}})$的夹角$\theta(\mathbf{M}_P,\vec{\mathbf{u}},\vec{\mathbf{v}})$定义为[120]：

$$\theta(\mathbf{M},\vec{u},\vec{v})=\begin{cases}\cos^{-1}\left(\dfrac{\vec{\mathbf{u}}^{T}\mathbf{M}\vec{\mathbf{v}}}{l(\vec{\mathbf{u}},\mathbf{M}).l(\vec{\mathbf{v}},M)}\right) & if\ (\mathbf{u}_x\mathbf{v}_y-\mathbf{u}_y\mathbf{v}_x)>0\\ 2\pi-\cos^{-1}\left(\dfrac{\vec{\mathbf{u}}^{\mathrm{T}}\mathbf{M}\vec{\mathbf{v}}}{l(\vec{\mathbf{u}},M).l(\vec{\mathbf{v}},M)}\right) & otherwise\end{cases} \tag{4.11}$$

在这里首先定义$\mathbf{M}^{1/2}$和$\mathbf{M}^{-1/2}$，在二维情况下$\mathbf{M}^{1/2}$和$\mathbf{M}^{1/2}$表示为：

$$\begin{aligned}\mathbf{M}^{1/2}&=\frac{1}{\sqrt{E+G+2D}}\begin{bmatrix}E+D & F\\ F & G+D\end{bmatrix}\\ \mathbf{M}^{-1/2}&=\frac{1}{D\sqrt{E+G+2D}}\begin{bmatrix}G+D & -F\\ -F & E+D\end{bmatrix}\end{aligned} \tag{4.12}$$

这里:$D=\sqrt{EG-F^2}$。这样，平面中在考虑点 A 的黎曼度量情况下，点 B 绕点 A 逆时针旋转角度θ，得到新点 C 的位置为：

$$\begin{pmatrix}x_C\\ y_C\end{pmatrix}=\begin{pmatrix}x_A\\ y_A\end{pmatrix}+\frac{\mathbf{M}_A^{-1/2}\mathbf{R}(\theta)\mathbf{M}_A^{1/2}}{l(AB,\mathbf{M}_A)}\begin{pmatrix}x_B-x_A\\ y_B-y_A\end{pmatrix} \tag{4.13}$$

4.5 二维空间的背景网格

背景网格是指这样的一组网格，这组网格包括已经离散完成的单元和节点，这些单元可以完全覆盖准备进行剖分的区域，在这组网格的节点上存储着单元的尺寸和形状信息。当准备计算即将生成的单元尺寸时，可以根据当前节点落在背景网格中的位置通过插值计算求得新单元在该节点处的尺寸。

在自适应有限元网格生成过程中，单元尺寸计算是关键的问题。在一些情况下，单元尺寸可以通过单元尺寸控制函数求得，但对于比较复杂的问题这样的函数很难定义，特别是在自适应网格生成中，单元尺寸是不断变化的。在这种情况下，背景网格法可以满足单元尺寸计算的要求。从几何的观点来讲，背景网格可以是任意形式的划分，下面是几种可能的划分形式。

(1) 在二维情况时可以是四叉树(quad tree)，在三维情况时可以是八叉树(octree) ；

(2) 在二维情况时可以是正方形，在三维情况时可以是立方体；

(3) 可以是用户所生成的任何形式的网格，包括待剖分域的已有的有限元网格；

背景网格的作用是存储期望单元尺寸的信息，用来辅助生成满足尺寸及方向要求的网格，因此背景网格的拓扑形状可以是任意的，只要它能够有效、灵敏地反映区域网格的尺寸及方向特征，能够快速、准确地进行新生成单元尺寸及方向特征的计算即可。按照这个标准衡量，本文采用四叉树/八叉树来作为背景网格。

4.5.1 四叉树数据结构

首先来介绍四叉树(quad tree)，四叉树在计算几何中有着非常广泛的应用，它是一种层次结构，如图 4.5 所示。在这里四叉树的根(root)代表二维空间中整个待剖分域的外包围盒，然后递归的把它分成 4 个子节点，这种分裂继续下去，直到所有的叶子节点满足剖分尺寸的要求。这样每个节点(除根节点)外都有一个父节点， 每一个非叶子节点都有 4 个子节点。这样通过四叉树既可以由上至下(top-down)遍历每一个节点，也就是从父节点来访问子节点；也可以以相反的方式遍历节点(down-top)，即由子节点访问对应的父节点。

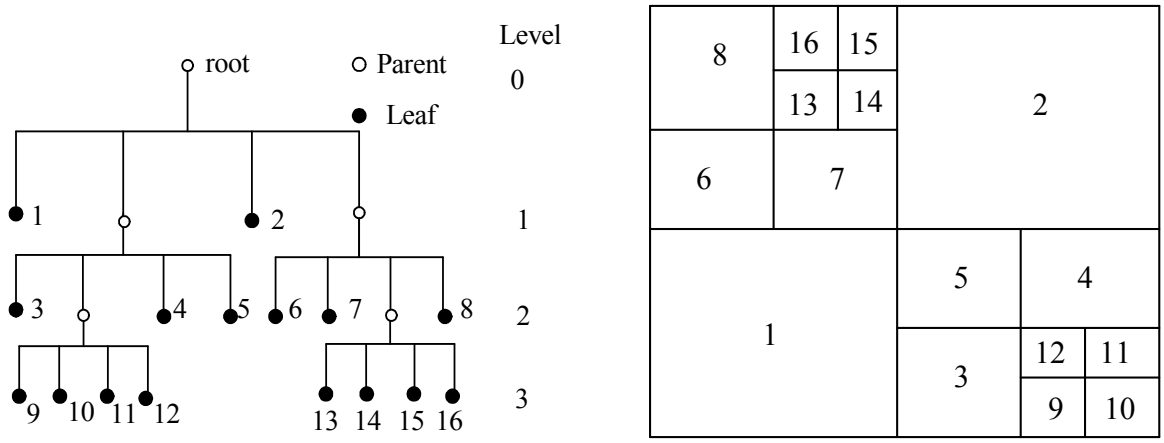


图 4.5 四叉树结构

Fig.4.5. Quadtree hierarchy

在应用四叉树作为背景网格时，首先根据自适应的要求计算出四叉树每个叶子节点的 4 个角点的黎曼度量，对于位于叶子节点内部的点的黎曼度量是根据 4 个角点的黎曼度量线性插值得到。插值公式为：

$$\mathbf{M}=\sum_{i=1}^{4}N_i\mathbf{M}_i \tag{4.15}$$

其中， $N_i(i=1...4)$ 分别为左下角、右下角、左上角、右上角的权重，计算公式为：

$$
\begin{aligned}
N_1 &= (1-\xi)(1-\eta)/4 \\
N_2 &= (1+\xi)(1-\eta)/4 \\
N_3 &= (1-\xi)(1+\eta)/4 \\
N_4 &= (1+\xi)(1+\eta)/4 \\
\xi &= 2(x-x_C)/s_x, \eta = 2(y-y_c)/s_y
\end{aligned} \tag{4.16}
$$

(x,y) 为插值点的坐标，(x_C,y_C) 为当前叶节点中心点的坐标，(s_x,s_y) 位节点的长度和宽度。

由上面的描述可以看出，一个典型的四叉树的节点应该包括 1 个父节点、4 个子节点以及节点的位置和大小信息。这样节点数据结构可以这样设计:

```
1. class SlGeBQuadTreeNode{
2. protected:
3.     SlGePointV2d          _leftdown;    // 节点左下角坐标
4.     Real                  _len[2];      // 节点的长度和宽度
5.     SlGeBQuadTreeNode* _leaf[5];        // _leaf[0] 为父节点，其它为子节点
6.     int                   _whichAmI;    // 节点位于父节点的位置。
7. };
```

本文中为了四叉树平衡算法实现的方便，另外添加了该节点在父节点位置的属性(_whichAmI：0 本身是根节点；1 位于父节点的左下角；2 位于父节点的右下角；3 位于父节点的左上角；4 位于父节点的右上角)。

4.5.2 四叉树的平衡

利用 QuadTree 结构可以高效地查询给定的坐标点落在哪个叶子节点(leaf)中，并可以高效地搜索给定叶子节点临近的叶子节点。这两项操作的时间复杂度为 $O(\log n)$。其中，n 为 Quad Tree 四叉树的最大深度。这样，叶子节点的查询时间与树的最大深度成正比。在自适应网格生成中，由于节点在区域内的分布很不均匀，节点之间的距离有很大的变化，为此需要对四叉树进行平衡。用平衡的四叉树作为背景网格的最大一个好处是因为它限制了相邻节点的变化梯度，也就是限制了相邻节点的黎曼度量变化的梯度，从而更容易产生梯度变化均匀的网格。

四叉树可以在四叉树建立以后平衡，也可以在建立四叉树的过程中平衡。后者的好处是在四叉树建立的过程中，当前的四叉树始终是一颗平衡的四叉树，从而减少整个平衡四叉树的建立过程。在本文中平衡的四叉树是指四叉树的每个叶子节点和相邻叶子节点的至多有 1 层的差别。图 4.5 的四叉树就不平衡，因为叶子节点 2 和叶子节点 14、15 有 2 层的差别。要使之平衡，必须进一步分裂叶子节点 2，平衡后的四叉树如图 4.6 所

示。四叉树的一个叶子节点至多有 8 个相邻节点，所以平衡算法的首要步骤是找出当前节点的相邻节点。找相邻节点的算法是一个递归算法，这里以找北向相邻节点(north neighbor)为例，算法如下：

算法 4.2：北向相邻叶子节点(north neighbor)的查找

```
1.  if(_whichAmI == ROOT)
2.      return NULL;
3.  if(_whichAmI == RIGHT_DOWN)
4.      return _leaf[FARTHER]->leaf(RIGHT_UP);
5.  if(_whichAmI == LEFT_DOWN)
6.      return _leaf[FARTHER]->leaf(LEFT_UP);
7.  SlGeBQuadTreeNode* node = _leaf[FARTHER]->northNeighbor(dep);
8.  if(node == NULL || node->isLeaf())
19.      return node;
10. if(_whichAmI == LEFT_UP)
11.  return node->leaf(LEFT_DOWN);
12. else
13.  return node->leaf(RIGHT_DOWN);
```

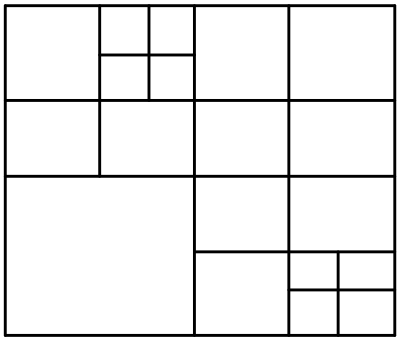


图 4.6 平衡的四叉树

Fig.4.6. The balanced quad-tree

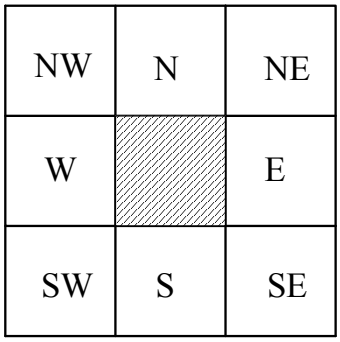


图 4.7 四叉树节点的邻居

Fig.4.7. The neighbors of quad-tree's node

找其它相邻节点的算法与此类似，这里就不列出来了。在构建四叉树的过程中，分裂当前节点之前，首先找出分裂后会产生不平衡的节点，对某一个节点来说，因为和该节点是同一父节点的相邻节点肯定是平衡的，这样不平衡的节点最多有 3 个。这个算法比较简单，如下所示：

算法 4.3 不平衡叶子节点的查找

```
1. SlGeBQuadTreeNode::getUnbalancedGrandfather(SlGeBQuadTreeNode* unBNodes [3])
2. unBNodes [0]= unBNodes [1]= unBNodes [2]=NULL;
3. if(_whichAmI==LEFT_DOWN)
```

```
4.    unBNodes [0] = southNeighbor();
5.    unBNodes [1]= westNeighbor();
6.    if(_leaf[0]!=NULL)   unBNodes[2] = unBNodes[0] ->westNeighbor() ;
7. else if(_whichAmI==RIGHT_DOWN)
8.    unBNodes [0] = southNeighbor();
9.    unBNodes [1] = eastNeighbor();
10.   if(_leaf[0]!=NULL) unBNodes [2] = unBNodes [0]->eastNeighbor() ;
11. else if(_whichAmI==RIGHT_UP)
12.    unBNodes [0] = northNeighbor();
13.    unBNodes [1] = eastNeighbor();
14.    if(_leaf[0]!=NULL) unBNodes [2] = unBNodes [0]->eastNeighbor() ;
15. else if(_whichAmI==LEFT_UP)
16.    unBNodes [0] = northNeighbor();
17.    unBNodes [1] = westNeighbor();
18.    if(_leaf[0]!=NULL) unBNodes [2] = unBNodes [0]->westNeighbor() ;
```

找出不平衡的节点后，在分裂当前节点的同时分裂这些不平衡的节点，算法如下：

算法 4.4 分裂并平衡叶子节点

```
1. SlGeBQuadTreeNodeEx::balancedSplit()
2. SlGeBQuadTreeNode* unBNodes[3];
3. getUnbalancedNodes(unBNodes);
4. split();
5. if(unBNodes [0]!=NULL) unBNodes [0]->balancedSplit();
6. if(unBNodes [1]!=NULL) unBNodes [1]->balancedSplit();
7. if(unBNodes [2]!=NULL) unBNodes [2]->balancedSplit();
```

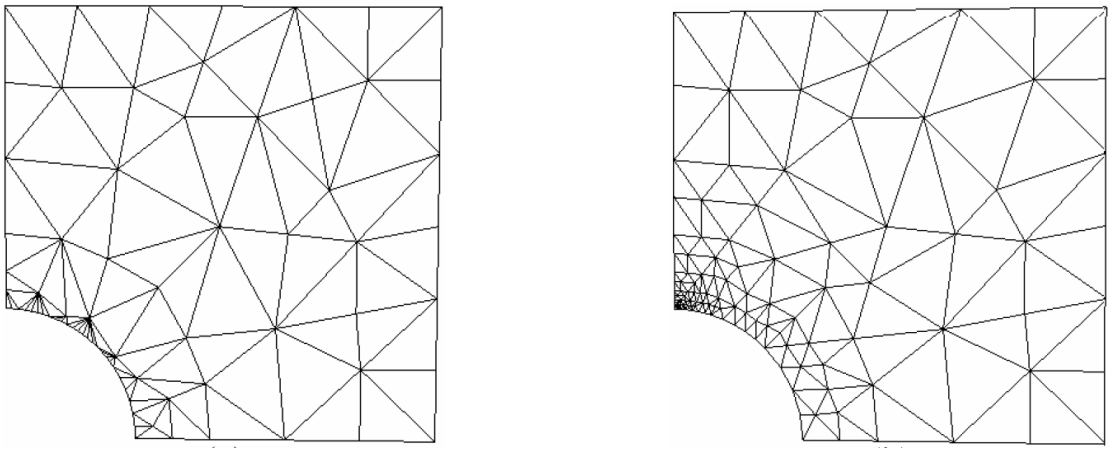
4.6 梯度黎曼度量场的构建

本文是用黎曼度量来控制网格的尺寸和形状的。一般来说网格尺寸变化越剧烈，单元的质量也就越差，假如要得到均匀梯度变化的网格，用来控制网格的黎曼度量就要梯度变化均匀。本文称梯度变化均匀的黎曼度量场为梯度黎曼度量场。

通常我们首先得到离散化的黎曼度量场，然后通过该离散化的黎曼度量场插值得到连续的黎曼度量场。离散化的黎曼度量场一般称为背景网格，背景网格是对待剖分域的一个简单剖分，比如删格、四叉树或者上一次的剖分结果。如前所述，本文是用平衡的四叉树作为背景网格的。

这里首先计算平衡四叉树背景网格中叶子节点的各角点处的黎曼度量，剖分域中其它点的黎曼度量是这样计算的：首先找到该点所在的四叉树叶子节点；然后按照公式

4.15 由该叶子节点的 4 个角点的黎曼度量插值计算得到该点的黎曼度量。通过这样的插值方法就可以得到连续的黎曼度量场。在实际应用中由于待剖分域几何和尺寸控制的复杂性，待剖分域中定义的初始黎曼度量场中相邻点的黎曼度量可能产生突变的情况，这样会导致生成的有限元网格在突变区域的尺寸和形状亦发生突变，从而不适合计算的要求，为此必须消除这些突变的黎曼度量，即黎曼度量场的梯度化。图 4.8 就是梯度黎曼度量场构建前后的生成的网格对比。



(a) 非梯度有限元网格　　(b) 梯度化有限元网格

(a) Ungraded mesh.　　(b) Graded mesh.

图 4.8 梯度黎曼度量场构建的必要性

Fig.4.8. The needing of mesh gradation control.

关于梯度黎曼度量场的构建，文献[98, 100]有较为详细的论述，这里简单总结一下：首先定义黎曼度量场中两个点 P,Q 处黎曼度量在方向 (e^p,e^Q) 上变化梯度的为：

$$\gamma(e^p,e^Q)=e^{\frac{\left|h(P)^{e^p}-h(Q)^{e^Q}\right|}{\|\overrightarrow{\mathbf{PQ}}\|}} \tag{4.17}$$

其中：

$$h(i)^{e^i}=\frac{1}{\sqrt{e^i M^i e^{iT}}}(i=p,Q) \tag{4.18}$$

在各向同性的情况下，假设单元的尺寸是按几何级数变化的，也就是说 P,Q 两点间的单元尺寸满足：

$$h(t)=h(P)(h(Q)/h(P))^t \tag{4.19}$$

这样由当 $h(Q)>h(P)$ 时，公式(4.9c)可以得到：

$$l(PQ)=\left\|\overrightarrow{PQ}\right\|\frac{h(Q)-h(P)}{\ln(h(Q)/h(P))} \tag{4.20}$$

这样(4.17a)可以简化为：

$$\gamma = (h(Q)/h(P))^{1/l(PQ)} \tag{4.21}$$

对于一个点的各向异性的黎曼度量，在两个主方向的单元尺寸是不一样的，定义一个最大主方向单元尺寸与最小主方向单元尺寸的比值：

$$R = h^{e0}/h^{e1} \tag{4.22}$$

这里 $h^{e0} > h^{e1}$，否则取其倒数。修正一个点的黎曼度量可以通过改变它的两个主方向和对应的单元尺寸两个方面着手。构建梯度黎曼度量场是通过修正背景网格中每条边上的黎曼度量来完成的，对于一条边 PQ 上两个点的黎曼度量，如果需要修正，应该遵循以下三个原则：

(1) 如果 P 点的单元尺寸大于 Q 点的单元尺寸，那么通过减小 P 点的单元尺寸来达到修正要求。

(2) 如果 P 点的 R 值大于 Q 点的 R 值，保持 P 点的两个主方向不变，修正 Q 点的两个主方。

(3) 如果 P,Q 点的 R 值接近，P,Q 两点的两个主方向都保持不变。

这样需要定义 P,Q 两点主方向修正因子，假设 $R(P) \geq R(Q)$，那么这个修正因子为：

$$\alpha = \frac{(R(Q)-1)R(P)}{(R(P)-1)R(Q)} \tag{4.23}$$

这样 Q 点的主方向修正公式为：

$$e^i(Q)\big|_{new} = \alpha e^i(Q) + (1-\alpha)e^i(P) \tag{4.24}$$

这样对于用户允许的最大黎曼度量变化梯度 γ_0，梯度黎曼度量场构建算法如下:

算法 4.5 梯度黎曼度量场的构建

(1) 遍历四叉树叶子节点中每一条边，取出当前边 PQ

(2) 判断的 P,Q 两点的黎曼度量的属性，如果都是各向同性的，那么：

假设 $h(Q) > h(P)$，按照公式(4.17e)计算 P,Q 两点的黎曼度量的变化梯度 γ，如果 $\gamma > \gamma_0$，那么：

① 求得在 $M(Q)$ 下的 P,Q 两点的黎曼长度 $l(PQ, M(Q))$；

② 计算 $\eta = (\gamma_0/\gamma)^{l(PQ)}$；

③ Q 点新的黎曼度量为：$M(Q) = M(Q)\eta^{-2}$

(3) 如果 P,Q 两点的黎曼度量有一个是各向异性的，那么：

① 按照公式(4.23)计算黎曼度量修正因子 α

② 按照公式(4.24)修正Q的主方向

③ 修正Q点黎曼度量两个主方向上的单元尺寸：$h^i(Q)=h^i(P)+l(PQ)\ln(\gamma_0)$

(4) 如果当前的叶子节点任何一条边上的点黎曼度量发生更新，则和该叶子节点在这条边相邻的叶子节点重复上述所有步骤

4.7 基于黎曼度量的二维自适应波前推进算法

本文采用流行的推进波前法(Advancing Front Technique, AFT)在二维域中生成三角形单元。和一般的 AFT 方法类似，本文的三角剖分算法也是从一个有向线段集合开始的。这个有向线段集合称为初始前沿，从前沿线段向待剖分的内推进形成三角形单元，直到整个域剖分结束。所以这里要首先生成初始前沿，也就是待剖分域的边界离散化。

4.7.1 边界离散化

一般来说待剖分域的边界是采用 B-Rep 的方式描述的，对二维剖分域来说，B-Rep 描述中主要包含平面的曲线以及连接关系。要产生如前所述的单位网格的前提是待剖分域的边界被离散化为单位长度的线段。C.K.Lee[49]给出了基于迭代的边界离散化方法，在他的方法中每产生一条边界边都要经过多次迭代才能得到，实践证明这种方法是比较慢的。本文给出速度较快的基于二分法的离散化方法。设参数曲线的表达式为：$C(t)=(x(t),y(t))$，$t\in[0,1]$,设$t_0=0$,$t_1=1$，对这条曲线的离散化也是一个递归过程，C++形式的伪代码如下：

算法 4.6 曲线的离散化

```
1. void dicreateCurve (double t0,double t1, double& residua){
2.    double length=lengthOf(t0,t1);
3     if(residua>0.0) length+= residua;
4.    if (length > √2 ){
5.        t_m = (t_0 + t_1) / 2;
6.        dicreateCurve ( t_0 , t_m ,iLevel);
7.        dicreateCurve ( t_m , t_1 ,iLevel);
8.    }else if (length ≥ 1/√2   &&    length  ≤ √2 )
9.        residua=-1.0;
10.       append( C(t_1) ); //点 C(t_1) 添加到曲线的节点序列
11.   }else if(length < 1/√2 ){
12.       residua= length;
13.   }
```

14.}

在算法 4.6 中第 2 行计算两个点的距离时(lengthOf(t0,t1))按照 4.3 描述的方法进行计算，其中任意一点的黎曼度量由背景网格中获得。参数 residua 表示残差，初始值为-1.0，表示上一次计算的结果如果小于$1/\sqrt{2}$，那么累积到下一段线段中。离散化后每条线段的黎曼长度满足：$l \in [1/\sqrt{2}, \sqrt{2}]$。这个离散化过程得到的曲线$C(t)$的节点序列，是按节点对应的$t$值的大小自然排序。有了这些有向线段以后 AFT 方法的具体实施步骤如下：

4.7.2 必要的关系和数据准备

以下数据和关系是在整个剖分过程中必须维护的：

(1) 节点 i (Node i):包含节点的坐标$x_i(u,v)$

(2) 第 n 层前沿上的节点(Node i): 包括参照高度h_{oi}、搜索半径r_{oi}、连接边的引用E_{oi}。节点的参照高度h_{oi}和搜索半径r_{oi}是这样定义的：首先令搜索半径r_{oi}等于所连接的最长边的边长，然后计算节点所连接边的边长的平均值：l_{avg}，这样参照高度$h_{oi} = \alpha l_{avg}$（这里$\alpha = 0.87$，是正三角形的高和边长之比）。为了得到变化均匀的网格，参照高度h_{oi}和搜索半径r_{oi}的计算还要考虑到其周边区域对它的影响。如图 4.9 所示，首先定义一个圆(x_i, r_{oi})（在各向异性的情况下应该是个椭圆。对应于已知圆来计算椭圆的方法在下节介绍），如果剖分域中有节点$Node(x_j)$在圆（椭圆）内，那么节点$Node(x_i)$和节点$Node(x_j)$的调整参数计算如下：

$$
\begin{aligned}
&c_{oij} = 0.05 \frac{3r_{oi}}{|x_j - x_i|} \frac{h_{oj} - h_{oi}}{h_{oj}} \\
&c_{ji} = \min(\max(c_{oij}, -0.6), 0.5) \\
&c_i = c_{ji},\ if\ |c_{ji}| > |c_i| \\
&c_j = c_{ij},\ if\ |c_{ij}| > |c_j|
\end{aligned}
\tag{4.25a}
$$

节点$Node(x_i)$和节点$Node(x_j)$的初始系数节c_i, c_j都取零。为了找到位于圆中的节点，节点的搜索过程是必须的。因为在整个剖分过程中类似的搜索过程比较频繁，所以快速的搜索算法在这里显得尤为重要。为了快速的搜索位于椭圆内的节点,首先得到椭圆的外包围盒（如图 4.10 所示），对于位于外包围盒内的所有节点，再判断这些节点是否落在椭圆内。最后，当节点的调整系数确定以后，最终的节点的参照高度h_{oi}'和搜索半径r_{oi}'按下面公式计算得到：

$$h_{oi}' = (1.0 + c_i)h_{oi}$$
$$r_{oi}' = \max(1.0 + c_i, 1.0)r_{oi} \qquad (4.25b)$$

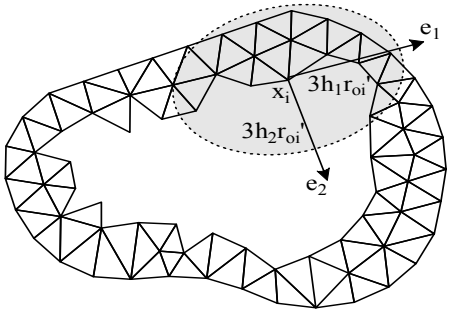


图 4.9 节点的参照高度 h_{oi} 和搜索半径 r_{oi} 的调整

Fig. 4.9. Reference height and searching radius of a node

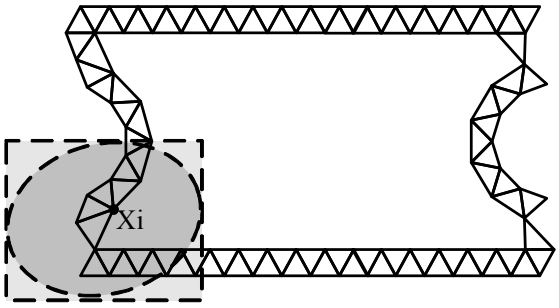


图 4.10 搜索前沿和搜索节点的形成

Fig.4.10. Forming searching fronts and nodes

4.7.3 初始化前沿队列

前沿队列Ω是一个以前沿长度作为排序准则的优先队列，所有未剖分区域的边界线段和端点的集合构成Ω。经验表明，小长度优先最后生成的三角形单元的质量系数总体较高。如果整个剖分过程都按照小长度前沿优先，其边界网格有时质量较差，所以一般采用按层推进方式：只有当当前层的前沿都推进完成后再进行下一层的剖分，在每一层的前沿的推进过程中按最小长度优先的方式进行。实践证明，按层推进也有一个弊端，就是当两个迎面的前沿层相遇时，生成的网格的质量往往不好，这种情况通常在剖分域的内部发生。为了克服上述按最短边优先的推进方式和按层推进方式的缺点，本文采用两种相结合的方式。即：为了保证边界附近网格的质量，首先按按层推进的方式推进 n 层(本文中 n=2)，当前 n 层推进完毕后再按最短边优先的推进方式的继续推进，直到整个待剖分域剖分结束。

另外，这部分的算法还应用到参数曲面参数域的网格生成中(在组合曲面剖分部分详细介绍)，如果参数曲面中存在极点，当前沿推进到极点附近的时候，求到的理想点可能在曲面参数域外，这时当前的前沿就不能生成新的单元，否则会出现错误。本文把这些前沿放到一个特殊的前沿队列Ω_h中，最后统一进行处理。

综上所述，本文的前沿队列一共有三个：当前层的前沿对队列Ω_c；下一层的前沿对队列Ω_n；以及特殊的暂时不可剖分的前沿队列Ω_h，并且这些前沿队列都是一个按前沿长度为优先序的优先队列，所以初始化前沿队列包括：

(1) 把离散化边界曲线得到按右手规则的有向前沿线段，这些前沿线段的集合形成待剖分域，放到Ω_c中。

(2) 把Ω_c中所有节点放到背景网格中，并按照公式(4.25)求得每个节点的参照高度和搜索半径。

(3) 设置剖分方式为按层推进方式，已剖分层数$L_n=0$。

4.7.4 前沿推进

对当前层的前沿队列Ω_c中的每一个前沿向前推进，其推进算法如下：

Step.1 计算单元的理想尺寸和理想点

如前所述，前沿队列是以前沿长度为优先序的一个优先对列，所以在前沿推进的时候首先从前沿队列里取出第一个前沿，记为 AB。那么新生成的单元的理想尺寸由下面公式确定：

$$h_{\Sigma}=\frac{h_{oA}+h_{oB}}{2} \tag{4.26a}$$

其中h_{oA}，h_{oB} 分别表示按公式（4.25b）计算得到的 A，B 两节点的参照高度。在这里，因为所有长度都是按公式(4.9)考虑黎曼度量后计算得到的在黎曼空间中的长度，一般来说它和欧拉空间中的长度h_{Ω}是不等的，即$h_{\Sigma}\neq h_{\Omega}$；并且在欧拉空间中，理想点也不在线段 AB 的垂直方向上，而是在和 AB 成一定夹角。可以这样计算理想点的位置：

(1). 分别以 A 点和 B 点的黎曼度量为参照，按照公式 4.13，把当前前沿绕 A 点逆时针旋转 60^{o} 计算得到点N_A，和N_B。公式中$l(AB,\mathbf{M}_A)$用h_{Σ}替换，计算理想点的位置$N=(N_A+N_B)/2$；

(2). 计算线段AN和BN的黎曼长度$l(AN)$、$l(BN)$，重新计算N_A和N_A的位置：

$$\begin{aligned} N_A &= A+\overrightarrow{AN}/l(AN) \\ N_B &= B+\overrightarrow{BN}/l(BN) \end{aligned} \tag{4.26b}$$

(3). 理想点的位置为：$N=(N_A+N_B)/2$；

h_{Ω}由如下公式计算得到：

$$h_{\Omega}=\frac{h_{\Sigma}}{\sqrt{E_{C_{AB}}u_{2D}{}^2+2F_{C_{AB}}u_{2D}v_{2D}+G_{C_{AB}}v_{2D}{}^2}} \tag{4.26c}$$

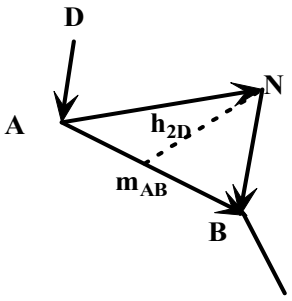


图 4.11 理想尺寸和理想点的确定

Fig.4.11. Calculation of element size and ideal nodal position

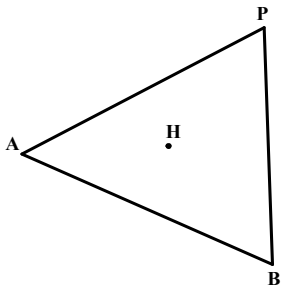


图 4.12 检查是否有节点在三角形内

Fig.4.12 Test for node inside triangle

计算得到理想点 N 后，检验该点是否落在剖分域内，如是，转入 Step2；否则，则把当前前沿 AB 放入Ω_h中，并从Ω_c中删除其前沿 AB，转入 **Step.1**, 继续剩余前沿的剖分。

Step.2 形成搜索前沿和搜索节点集

所谓搜索前沿是在当前前沿周围的一系列前沿，记为Ω_L。首先取得当前前沿 AB 的两个节点的搜索半径r_{oA}， r_{oB}，则前沿 AB 的搜索半径为：$r_\Sigma = \beta.\max(r_{oA}, r_{oB})$，$\beta$是一个经验值,本文取 1.2。再从背景网格中得到理想点 N 处的黎曼度量，根据公式(4.26c)计算得到欧拉空间的搜索半径r_Ω。

对参数域是各向异性的曲面，前沿 AB 的搜索范围应该以一个以理想点 N 为中心的椭圆。对于椭圆，有圆心在原点的椭圆的一般方程：

$$Ax^2 + By^2 + Cxy + D = 0 \tag{4.27a}$$

这里$A = E_N$， $B = G_N$， $C = 2F_N$， $D = -r_\Sigma{}^2$，然后计算：

$$\begin{aligned} R_1 &= (A\cos^2\theta + B\sin^2\theta + C\sin^2\theta/2)/D \\ R_2 &= (B\cos^2\theta + A\sin^2\theta - C\sin^2\theta/2)/D \\ \cos\theta &= F_N/\sqrt{E_N G_N} \end{aligned} \tag{4.27b}$$

这样椭圆的长短轴为:

$$\begin{aligned} a &= \sqrt{\frac{1}{\min(R_1, R_2)}} \\ a &= \sqrt{\frac{1}{\max(R_1, R_2)}} \end{aligned} \tag{4.27c}$$

所有与落在椭圆内节点相连的前沿构成搜索前沿，所有搜索前沿的端点的集合构成搜索节点集。

Step.3 形成活动（椭）圆和活动节点队列

当搜索前沿和搜索节点集形成以后，根据已有节点优先的原则，接下来的工作是形成活动椭圆和活动节点队列。所谓活动圆是以理想点 N 为圆心，当前前沿的参照高度 h_Σ 为半径的椭圆，把公式（4.13）中 r_Σ 替换成 h_Σ 计算得到椭圆长短轴，把属于搜索节点集并且在活动椭圆内的节点，按照和理想点距离最近为优先的原则进行排序，形成活动节点队列。

Step.4 在活动节点队列查找合适的点

如果节点队列为空，直接进入 Step.5，否则，依次从活动节点队列顶部弹出一个节点 P 和当前前沿 AB 组成三角形单元 ΔABP,直到通过下列检查：

(1) 面积检查：即 $\left|\overrightarrow{\mathbf{AB}}\times\overrightarrow{\mathbf{AP}}\right|>0$，即点 P 在有向线段 AB 的左侧；

(2) 不包含其它节点检查：即 ΔEFP 中不包含任何一个搜索节点。也就是说对于搜索节点 H，应该至少满足三条件之一：$\left|\overrightarrow{\mathbf{AB}}\times\overrightarrow{\mathbf{A}H}\right|>0$，$\left|\overrightarrow{\mathbf{B}P}\times\overrightarrow{BH}\right|>0$，$\left|\overrightarrow{P\mathbf{A}}\times\overrightarrow{PH}\right|>0$ 如图(4.11)

(3) 质量检查：线段 PA，PB 长度的最大值：$\max(l(PA),l(PB))<\delta h_\Sigma$，$\delta$ 是经验值，本文的实现中 $\delta=1.6$

(4) 相交检查：即线段 AP 和 PB 与任何一个搜索前沿不相交。

如果对活动节点队列中的节点 P，以上检查都通过则进入 **Step.7**,否则进入 **Step.5**。

Step.5 理想点 N 的可行性检查

如果活动圆内没有合适的节点，这时候把理想点 N 作为新生成三角形的第三个节点，进行如下检查：

(1) 不包含其它节点检查：同 Step.4 中的(2）；

(2) 和搜索前沿的距离检查：遍历所有的搜索前沿，要求理想点 N 和所有搜索前沿的距离：$d_N>\varepsilon h_\Sigma$。为了加快计算速度可以进行如下粗略判断：假如待判断的搜索前沿为 AB，中点为 C，理想点 N 到这三个点的距离分别为：d_A，d_B，d_C，取 $d_N=\min(dA,d_B,d_C)$，本文经验值 $\varepsilon=0.6$；

(3) 相交检查：同 Step.4 中的 4）。

如果理想点 N 通过以上三项检查，则进入 **Step.7**；否则进入 **Step.6**。

Step.6 在搜索节点集中查找合适的点

此时把搜索节点集中除去活动圆内的剩余节点构成新的节点集合，进行 Step.4 中的所有检查。因为搜索半径是当前前沿所连接的边的最长边，同时又使用公式（4.10）考虑了周围节点对它的贡献，并且乘以适当的系数，经验表明这个时候总能找到一个合适的节点符合上述判断条件。找到合适的节点后进入 **Step.7**。

Step.7 前沿队列的更新

前沿队列的更新主要包括：

(1) 从前沿队列Ω_c中删除当前 AB；

(2) 假如可行点是 **Step.4** 或 **Step.6** 中的节点 P，那么：

① 构造有向前沿线段$\overrightarrow{AP},\overrightarrow{PB}$，使待剖分域位于$\overrightarrow{AP},\overrightarrow{PB}$的左侧。

② 检查前沿队列Ω_c，Ω_n和Ω_h中是否有$\overrightarrow{AP},\overrightarrow{PB}$，如果有则从相应的前沿队列中删除，否则：

(a) 如果当前是按层推进方式，$\overrightarrow{AP},\overrightarrow{PB}$添加到前沿队列$\Omega_n$中；

(b) 否则，$\overrightarrow{AP},\overrightarrow{PB}$添加到前沿队列$\Omega_c$中；

③ 计算$\overrightarrow{AP},\overrightarrow{PB}$的长度，如果长度大于这三个节点所连接边的最大长度，则按公式(4.25)更新。

(3) 假如可行点是 **Step.5** 的理想点 N，那么

① 构造有向前沿线段$\overrightarrow{AN},\overrightarrow{NB}$，使待剖分域位于$\overrightarrow{AN},\overrightarrow{NB}$的左侧；

② 与(2)中② 类似地把$\overrightarrow{AN},\overrightarrow{NB}$添加到相应的前沿队列（按层推进方式添加到$\Omega_n$,否则，添加到$\Omega_c$中）；

③按公式(4.11)更新节点关系。

(4) 另外还包括：检测前沿队列Ω_c是否满足：$\Omega_c=\{\phi\}$，即Ω_c是否为空。

如果$\Omega_c\neq\{\phi\}$，重复执行 **Step1~Step7** 的所有步骤。

如果$\Omega_c=\{\phi\}$，$\Omega_n\neq\{\phi\}$并且$L_n<2$，Ω_c和Ω_n互换。

如果$\Omega_c=\{\phi\}$，$\Omega_n=\{\phi\}$ 剖分结束；否则：

如果$L_n=2$，Ω_c和Ω_n互换，并且设置剖分方式为按最短边推进，重复执行 Step1~Step7 的所有步骤。

4.8 网格的优化

在三角形单元生成的过程中，虽然每生成一个单元都力图使之质量最好，但其最终结果中还是会存在单元质量很差的单元。单元的质量系数有多种衡量方法[121]，在这里三角形ΔABC的质量系数可以用三角形的内切圆和外界圆半径之比来表示，比值越大，说明该三角形质量系数越高。这样三角形的质量系数为：

$$\beta=I\frac{2r}{R}=I\frac{8(p-l_a)(p-l_b)(p-l_c)}{l_a l_b l_c} \tag{4.28}$$

其中：r 为三角形内切圆半径，R 为三角形外接圆半径，l_a,l_b,l_c 是三角形三边的边长，用公式(4.7)进行计算，$p=(l_a+l_b+l_c)/2$ 为三角形的半周长。用上述公式计算的三角形的质量系数 $\beta\in[0,1]$ 对正三角形 $\beta=1.0$。这里:

$$I=\begin{cases}-1 & Det(ABC)<0\\ 1 & Det(ABC)>0\end{cases}$$
$$Det(ABC)=\begin{vmatrix}x_A-x_C & y_A-y_C\\ x_B-x_C & y_B-y_C\end{vmatrix} \quad (4.29)$$

在此引入符号 I，来表示单元是否倒置，以保证在进行优化算法时单元的质量系数始终为正，也就是产生合法单元。一般来说对于网格的优化分为两大类方法，第一类是通过改变网格中节点的位置来提高网格的质量；第二类是通过改变单元的连接形式来提高网格的质量。

如前所述，基于黎曼度量的自适应网格生成的目标是产生一个待剖分域的单位网格，也就是使网格中每条边在黎曼空间中长度为 1。这样可以通过统计网格中的边长来得到生成网格的总体质量。

4.8.1 黎曼空间中三角单元的基于角度的优化

在第一类方法中比较常见的有 Laplacian 优化[71]、基于角度的优化[74]等。实践证明，基于角度的优化效果要比 Laplacian 优化好。但是要把这种优化方法应用到基于黎曼度量的二维自适应剖分中需要进行一些改动，现给出改动后的基于角度的优化算法。基于角度的优化的基本思想是把当前节点移动到所连接边的相邻两个三角形的角平分线上去，具体实现过程如下：

算法 4.6 黎曼空间中三角单元的基于角度的优化

(1) 如图 4.12 所示，网格内部的每个节点 N_i，周围都有 k 个相邻的节点，每个节点都对应两个节点 N_i 相关的角度 α_1，α_2，考虑黎曼度量时，这两个角度可以按公式（4.11）进行计算，公式中的黎曼度量可以取 N_i 点处的黎曼度量。

(2) 计算向量 v_j 的转角：$\Delta\alpha=(\alpha_1-\alpha_2)/2$

(3) 向量 v_j 以 N_j 为中心转动 $\Delta\alpha$，考虑黎曼度量时，节点 N_i 的新位置 N'_i 采用公式（4.13）计算。

(4) 用上述方法遍历其它节点，这样 k 个相邻的节点就会计算得到 k 个新位置。这 k 个新位置的重心就是当前节点的新位置：$N_{new}=\frac{1}{k}\sum_{i=1}^{k}N_i'$

实践证明采用上述方法计算得到新节点的位置并不能保证不产生负质量系数的单元，所以在得到新节点的位置后还要采用公式(4.30)对连接的三角形进行验证，如果有$I=-1$的情况，当前节点就不发生移动。

4.8.1 黎曼空间中三角形单元连接关系的优化

首先剖分域内部的一个节点连接三个单元(图 4.14a)时，这个节点是多余的，可以直接删除。另外一种节点可以删除的情况是当一个节点连接 4 个单元(图 4.14b)时可以把这个节点删除，4 个单元合并成两个单元。图 4.14b 中合并后至于形成ΔABC，ΔBDC还是ΔABD，ΔADC要看形成新的三角形的质量系数的高低，设ΔABC，ΔBDC，ΔABD，ΔADC按公式(4.29)计算得到质量系数分别为β_1，β_2，β_3，β_4，那么：

$$\gamma=\frac{\beta_1\beta_2}{\beta_3\beta_4} \tag{4.31}$$

如果$\gamma>1$，删除节点 E 后形成ΔABC，ΔBDC，否则形成ΔABD，ΔADC。

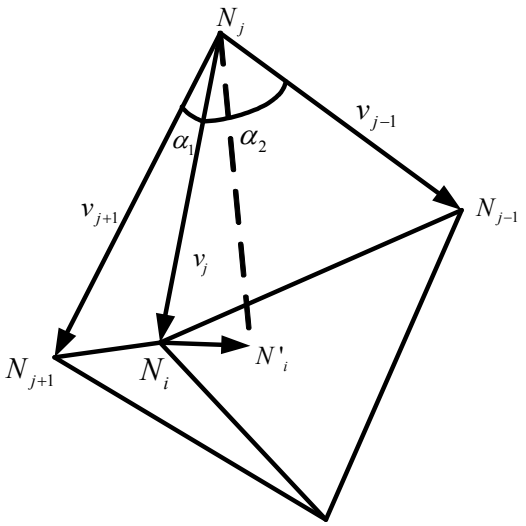


图 4.13 三角形单元的基于角度的优化

Fig.4.13. Angle-based smooth of triangular mesh

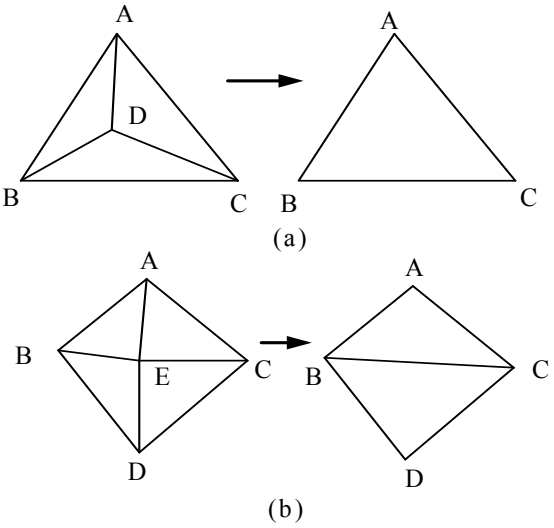
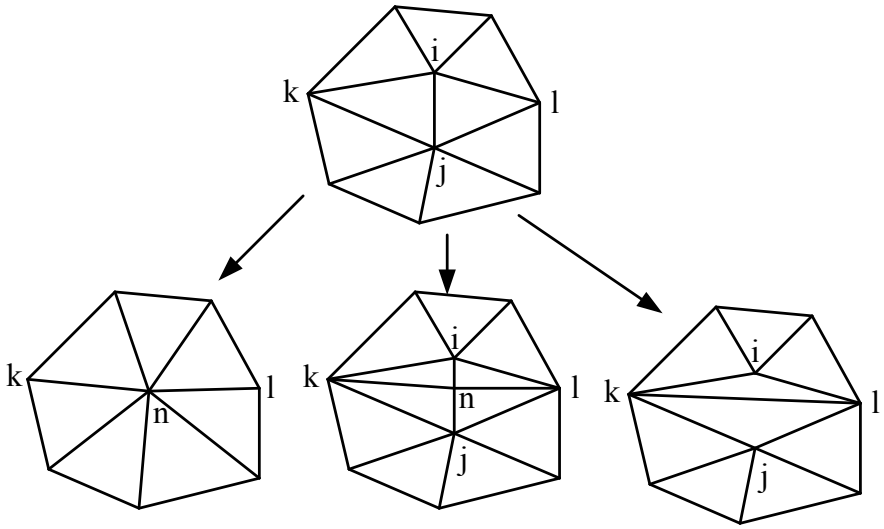


图 4.14 删除单元中多余的节点

Fig.4.14 Delete unwanted nodes in triangular mesh

另外还存在三种单元连接的优化方法，如图 4.15 所示。当剖分域内部某条边按公式 4.9 计算的长度$l_i<1/\sqrt{2}$这条边删除(图 4.14a)，当$l_i>\sqrt{2}$时这条边就分裂(图 4.15b)。在图 4.14 中，设Δikj，Δijl，Δikl，Δkjl的质量系数为为β_1，β_2，β_3，β_4，那么当按公式 4.31 计算得到的$\gamma<1$，并且$N_k>6, N_l>6$和$N_k+N_l-N_i-N_j>3$，就做图 15c 的边交换。



(a) Edge collapse (b) Edge split (c) Edge swap

图 4.15 单元连接优化

Fig.4.15 Change the topology of mesh

4.9 基于二分的二维自适应网格生成算法

这种方法是网格细化的常用方法[78, 119]。这里，首先构造待剖分域的背景网格，然后提取待剖分域的边界，按照 4.4.1 节描述的方法进行边界离散化。在文献[70]中详细论述了这种算法的详细步骤，并给出了结合 Delaunay 原理的改进方法。这里采用纯粹的基于长边分裂以及在每一个迭代步结束后进行一个优化循环的方法来生成自适应的网格，具体方法如下：

在这里假设输入的网格是一个三角剖分。一般来说当模型的边界离散化完成以后，可以用约束 Dalaunay 完成初始边界的三角剖分。这个算法认为初始剖分对边界的描述已经足够精确。接下来的算法比较简单，如图 4.16 所示，具体步骤大致如下：

(1) 统计初始剖分(4.15(a))的结果，把初始剖分中的边的长度：$l_i > \sqrt{2}$ 的所有边提取出来，形成边的集合 L

(2) 对 $L = \{l_i, i = 1...n\}$ 中的边按长度进行排序，使得 $l_1 \geq l_2 \geq ... \geq l_n$

(3) 依次遍历 L 中的所有边，对于当前边 l_i， 对分和 l_i 连接的所有四面体单元，对于新产生的边，如果边长 $l_i > \sqrt{2}$，放到集合 N 中。如果集合 N 的最长的边大于 l_i，那么这里需要首先分裂这条最长边，然后继续分裂 L 的其它边。（4.15(b),(c),(d),(e)）

(5) 清空 L，并使得 $L = N$。

(6) 调用优化程序，对网格的位置和连接进行优化:

(7) 调用 4.5.1 节描述的基于角度的优化方法，对节点的位置进行优化(14e)；

(8) 调用 4.1 节描述的方法对三角形单元的连接关系进行优化。

(9) 重复(2）~(8) 的步骤，直到 L 和 N 都为空，自适应过程结束。

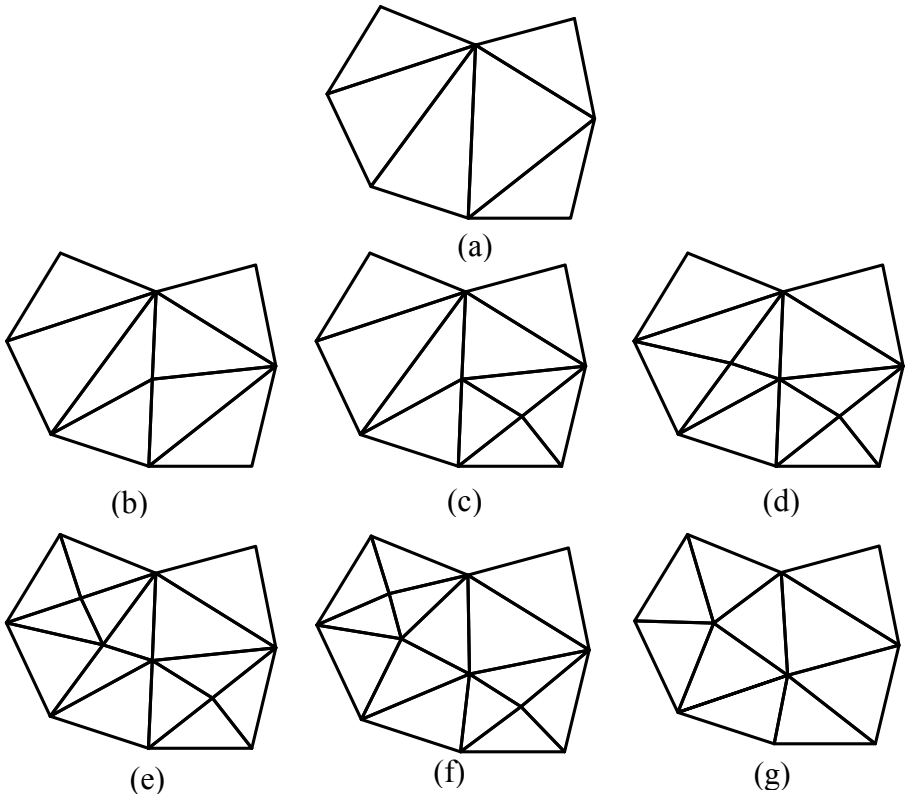


图 4.16 基于二分的网格生成算法

Fig.4.16 Adaptive mesh algorithm based on bisection

4.10 数值算例

这里分别给出采用波前推进算法和基于二分的自适应算法的数值算例。

4.10.1 波前推进算法的自适应剖分算例

算例 1 采用文献[102, 122]给出的例子，这个例子的边界是一个 7×9 矩形，矩形内部任意点的黎曼度量定义如下：

$$h(x,y)=\begin{cases}1.0-0.475y & y\in[0,2]\\ 0.05\times20^{(y-2)/2.5} & y\in(2,4.5]\\ 0.2^{(y-4.5)/2.5} & y\in(4.5,7]\\ 0.2+0.8((y-7)/2)^4 & y\in(7,9]\end{cases}$$

图 4.17(a) 是这个算例上述函数的图形显示，可以看出，这是一个分段函数，当 $y=4.5$ 和 $y=7.0$ 函数发生跳跃。这样，按照这个函数生成的黎曼度量场是个离散的黎曼度量场。图 4.17(b)是此算例的平衡四叉树的背景网格。

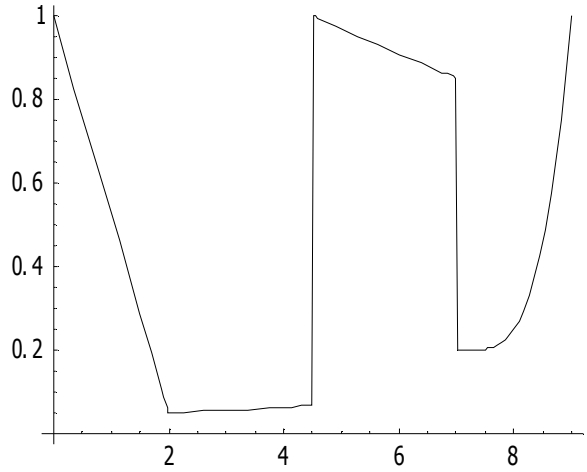


图 4.17(a) 算例 1 的单元尺寸函数

Fig.4.17(a) The size function of case 1

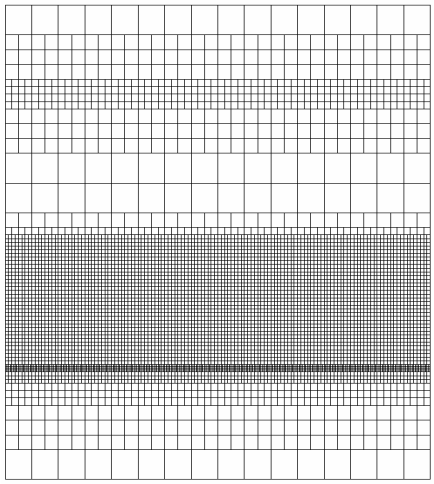


图 4.17(b) 算例 1 的背景网格

Fig.4.17(b) The background mesh of case 1

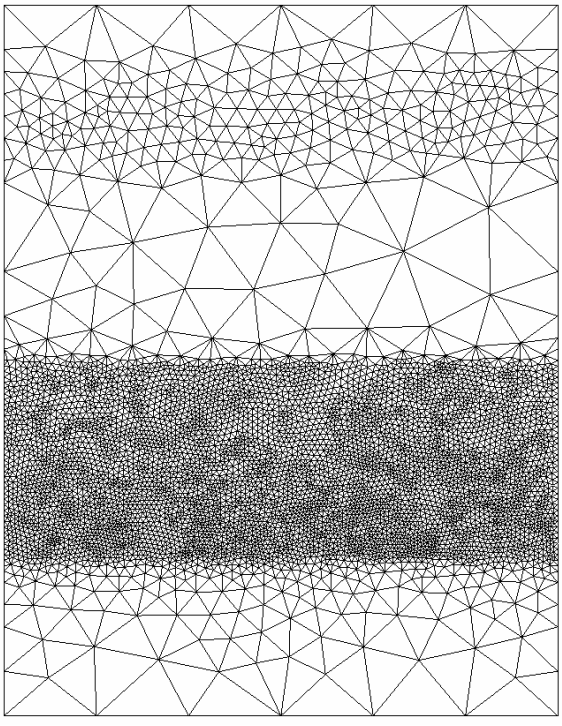


图 4.17(c) 梯度黎曼度量场构建前生成的网格

Fig.4.17 mesh without gradient metric control

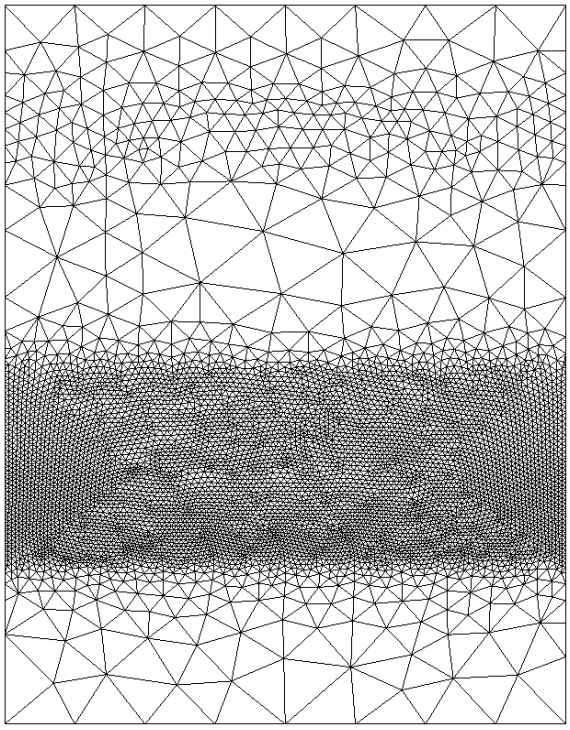


图 4.17(d) 梯度黎曼度量场构建后生成的网格

Fig. 4.17(d) mesh with gradient metric control

表 4.1 梯度黎曼度量场构建后生成的网格质量统计

Tab. 6.1 Statistics of triangle quality

单元质量系数	0.0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0	最小	最大	平均
单元个数	0	0	3	25	12555	0.42	0.99	0.98

表 4.2 梯度黎曼度量场构建后生成的网格边长统计

Tab. 6.2 Statistics of edge length

边长	0.0~0.5	0.5~0.8	0.8~1.2	1.2~1.5	1.5~2.0	>2.0	最短	最长	平均
边的个数	3	864	15386	2512	177	2	0.24	2.22	1.04

图 4.17(c) 是梯度黎曼度量场构建前生成的网格，由图中可以看出，由于单元的尺寸在 $y=4.5$ 和 $y=7.0$ 处发生突变，生成的网格在这两个地方过渡不均匀。图 4.17(d) 梯度黎曼度量场构建后生成的网格，可以看出，黎曼度量场梯度化后，整个域上网格过渡较梯度化前有明显改善。表 4.1 和 4.2 是按照公式 4.29 计算的单元质量系数以及网格中在黎曼度量下边长统计。实践证明，如果待剖分域上相邻区域的黎曼度量变化特别剧烈，由于 AFT 方法是一种启发式算法，没有严格的数学基础，这时候可能使算法失效，这样黎曼度量的梯度化就显得比较重要。

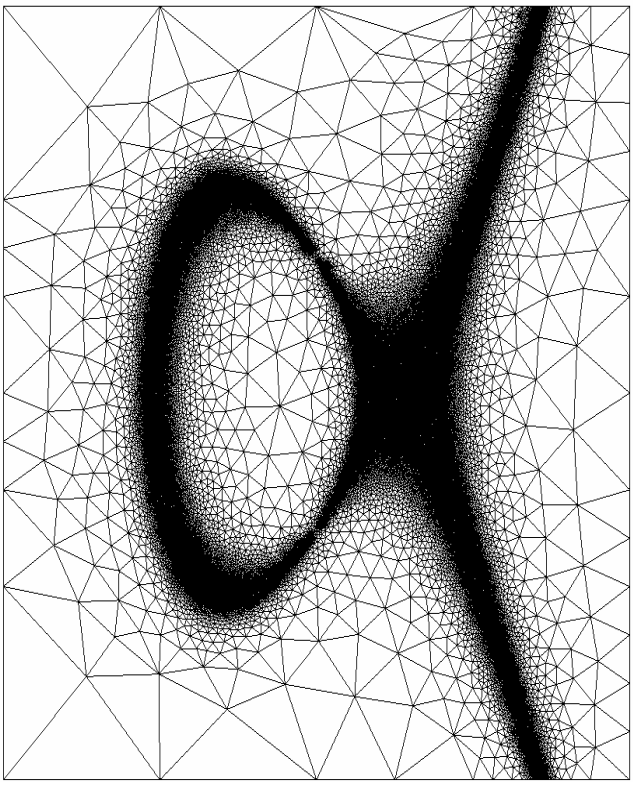


图 4.18 自适应算例 2

Fig.4.18 Adaptive meshing case 2

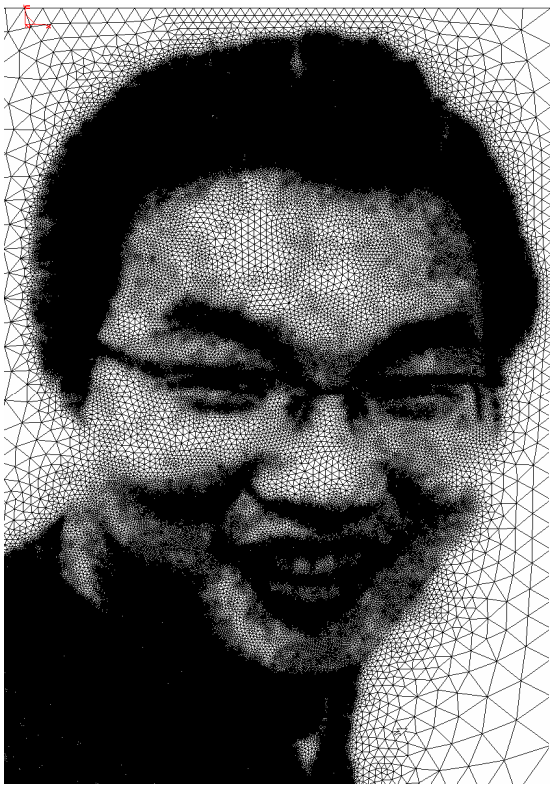


图 4.19 自适应算例 3

Fig.4.19 Adaptive meshing 3

图 4.18 是一个中心在原点，边长是 8 的正方形的自适应剖分结果，在这个算例中，单元的尺寸函数为：

$$h_1(x,y)=\begin{cases}\min(0.2(\lambda-1)^3+0.005,1.0) & \lambda\geq 1\\ \min(0.2(\lambda-1)^2+0.01,1.0) & \lambda<1\end{cases}$$

$$h_2(x,y)=\begin{cases}\min(0.2(\lambda-1)^3+0.2,1.0) & \lambda\geq 1\\ \min(0.2(\lambda-1)^2+0.2,1.0) & \lambda<1\end{cases}$$

$$\theta=\arctan\frac{-2y}{3(x^2-\lambda)}$$

$$\lambda=\frac{x^3-y^2+2}{3x}$$

图 4.19 是作者一张 2 寸照片的剖分结果，在这个算例中，首先提取图像中每个点的 RGB 的值，取原 RGB 三色的比例和作为灰度值，计算公式为：$Gr=0.3R+0.6G+0.1B$，网格的尺寸函数为：

$$h(x,y)=20\max(\frac{0.3\mathrm{R(x,y)}+0.6\mathrm{G(x,y)}+0.1\mathrm{B(x,y)}}{65535},0.001)$$

4.10.2 基于二分的自适应算例

下面给出的算例是采用基于二分的自适应剖分算法的结果。在这里给出一个 100 单位的正方形，中间有个半径 25 的圆，首先生成尺寸为 15 的均匀网格，也就是：$h(x,y)=15$。图 4.20 (a)是这个算例的 Delaunay 初始剖分。为了精确的描述模型的边界，这里可以定义相应的几何曲线，也可以采用常用的 PLC(Piecewise Linear Complex)来描述几何体的边界。图 4.21 (b)是图 4.20 (a)经过第一次迭代，在优化前的结果，图 4.20 (c)是该网格经过优化后得到的结果。中间其它步骤如图 4.20 (d)(e)所示。该算例经过 8 次迭代后，满足收敛条件，最终网格如图 4.20 (f)所示。

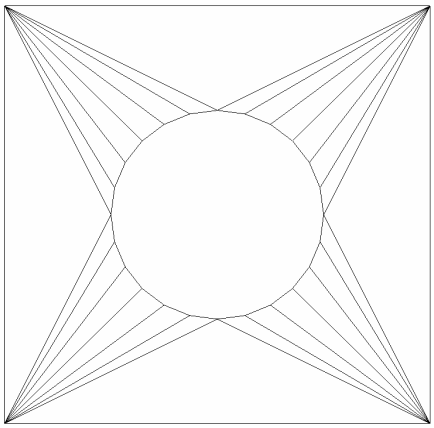


图 4.20(a) 初始剖分

Fig.4. 20 (a) Initial mesh

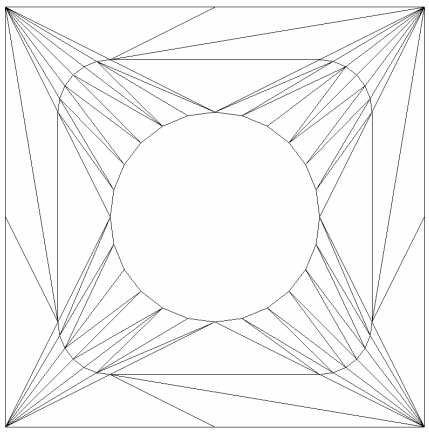


图 4. 20 (b) 第 1 次迭代(优化前)

Fig. 4. 20 (b) result of iterate 1(before optimize)

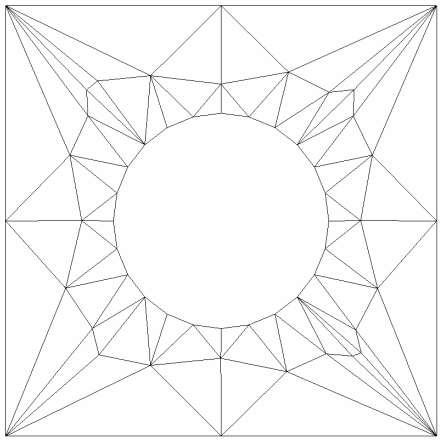


图 4. 20 (c) 第 1 次迭代(优化后)

Fig.4. 20 (c) result of iterate 1(after optimize)

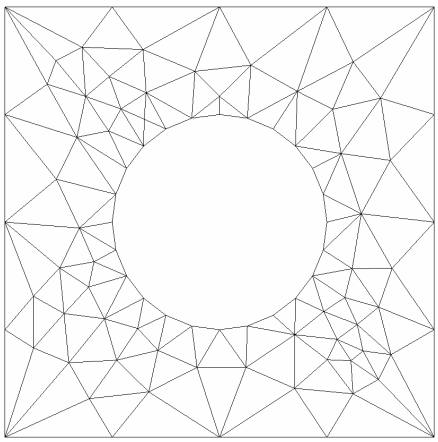


图 4. 20 (d) 第 3 次迭代

Fig.4. 20 (d) result of iterate 3

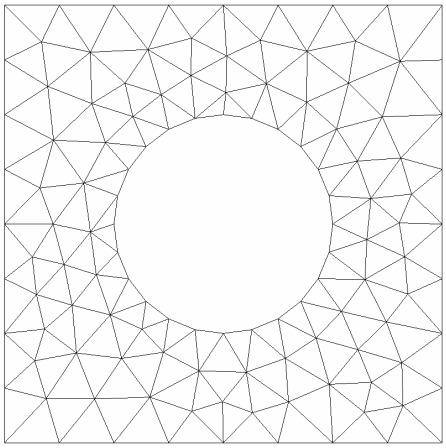


图 4. 20 (e) 第 6 次迭代

Fig.4. 20 (e) result of iterate 6

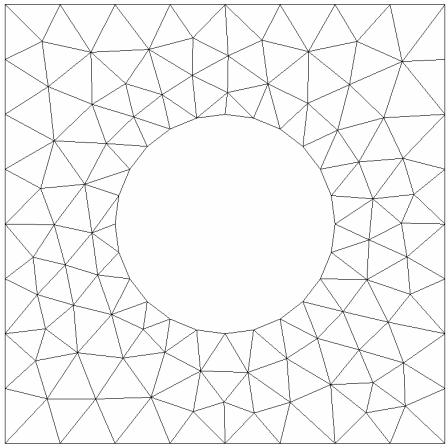


图 4. 20 (f) 最终网格(经 8 次迭代)

Fig.4. 20 (f) final mesh

下面这个算例的几何定义和上述算例相同，单元的剖分尺寸和举行两个对角线的距离成正比，初始剖分仍旧是图 4.20(a)的结果，图 21(a)到图 21(f)给出了各叠代步的结果。

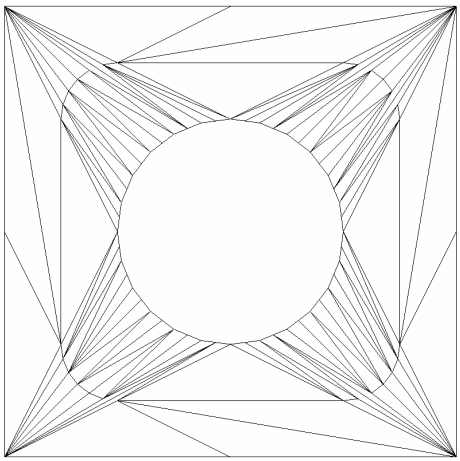


图 4.21(a) 第 1 次迭代(优化前)

Fig. 4.21 (a) result of iterate 1(before optimization)

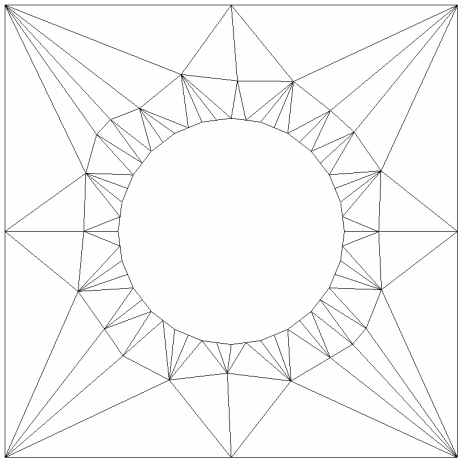


图 4.21(b) 第 1 次迭代(优化后)

Fig.4.21(b) result of iterate 1(after optimization)

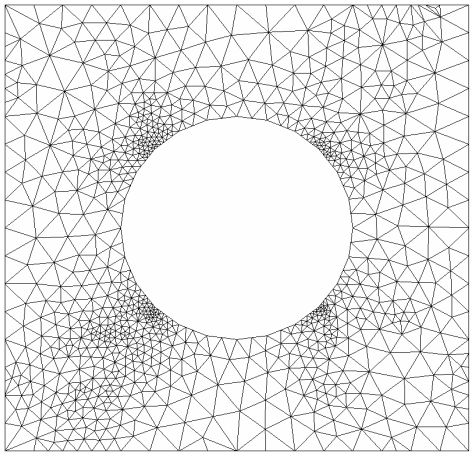


图 4.21(c) 第 3 次迭代

Fig.4.21(c) result of iterate 3

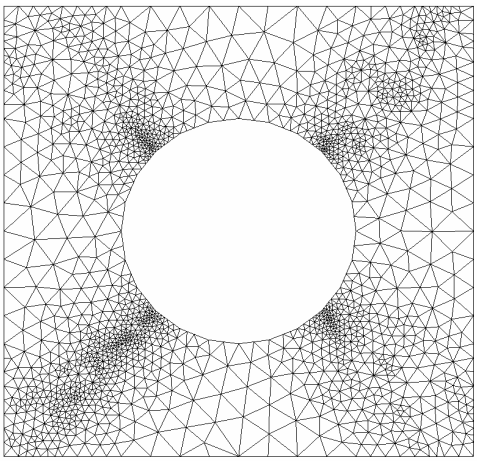


图 4.21(d) 第 6 次迭代

Fig.4.21(c) result of iterate 6

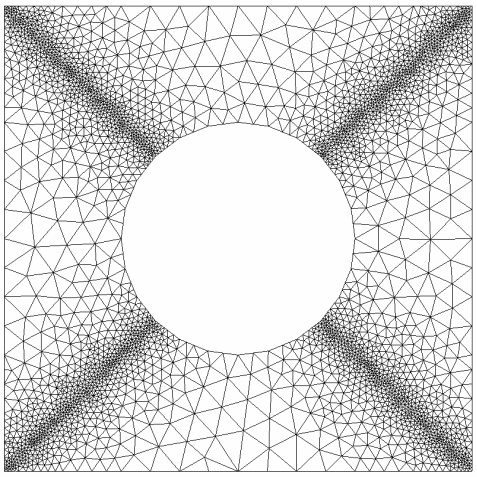


图 4.21(e) 第 10 次迭代

Fig.4.21(e) result of iterate 10

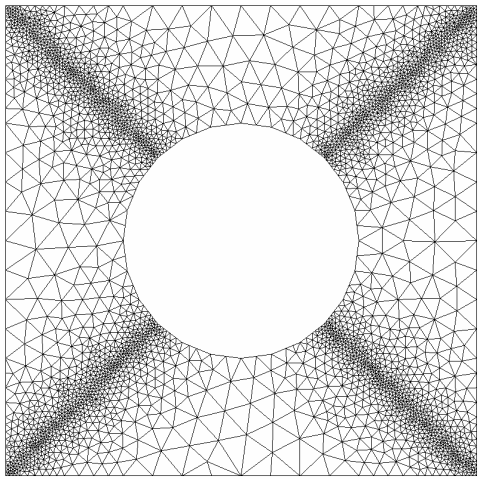


图 4.21(f) 最终网格(经 20 次迭代)

Fig.4.21(f) final mesh

5. 二维自适应网格生成在金属破坏模拟中的应用

5.1 引言

在金属破坏过程中通常存在几何非线性(如大变形)和物理非线性(如塑性变形)，对这种过程的模拟通常需要多方面的工具支持：理论基础(建立破坏的力学和数学模型)，数值方法(微分方程求解，非线性系统等)，几何工具(目标模型的表示，网格剖分、破坏过程中的自适应网格剖分)等。近年来许多学者在这个方面开展工作[96, 97, 115-117]。本章主要内容是将二维自适应有限元网格剖分算法应用于金属破坏模拟中。本文工作是与法国力学、材料与结构实验室(GMMS)合作研究的一部分，理论基础和数值方法研究是由该实验室的研究人员承担[122]。

5.2 破坏模拟的一般过程

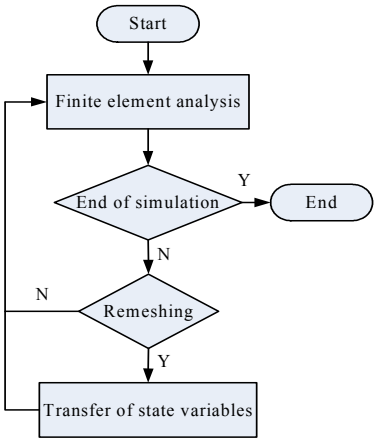


图 5.1 破坏模拟的一般过程

Fig.5.1. General Procedure of the process for simulations

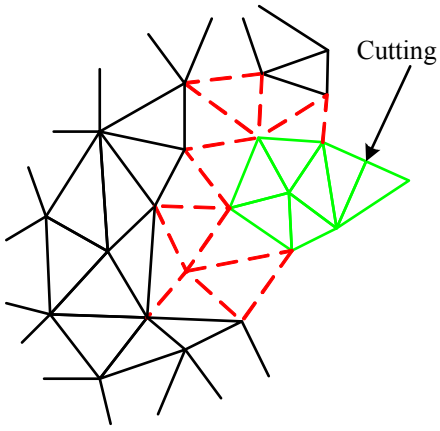


图 5.2 切屑的识别

Fig.5.2. Identification of cutting

图 5.1 是破坏模拟的一般过程，具体过程如下：

(1) 首先由有限元分析生成一个分析结果;

(2) 通过误差估计判断结果是否达到模拟的目标，如果达到目标，程序结束;

(3) 如果结果未达到目标，那么就把分析结果交给一个转换程序，把分析的结果转化为单元的尺寸和形状信息;

(4) 接下来由自适应剖分程序判断需要重新剖分的区域，对该区域进行自适应重剖分(参考上一章)。

(5) 把分析得到的结果状态变量(如应力、应变、温度等)转化到新单元上去。转到步骤(1)。

本文的主要工作是步骤(3)到步骤(5)。在这里首先把分析的结果作为一个二维表存储。表的横坐标是状态变量的类型，纵坐标是每个单元的标号，表中存储的是每个单元对应的这些状态变量的值。如果表中存储的是节点的状态变量值，首先需要把节点的状态变量值转换到单元上去，然后进行下一步的计算。转化过程将在本章后面介绍。这里还要预先定义网格中节点处的单元尺寸 $S(n(x,y))$ 和对应状态的变量的函数关系 $S(n_i(x,y))=T(n_i(x,y))$。这样自适应剖分过程又可以细分成以下几个子步：

(1) 统计表中的状态变量，一般来说状态变量有多种，比如应力、应变、速度、温度等等。用户设定状态变量表中的某一列作为参照，同时需要设定两个阀值：单元破坏阀值 α 和单元重剖分阀值 β，并且有 $\beta \le \alpha$。如果该列中某一单元的状态变量 $t_i > \alpha$，那么删除该单元，$\beta \le t_i \le \alpha$ 那么该单元需要重剖分，如果 $t_i < \beta$ 该单元保留。这样把现有的单元分成三个部分：

① 已经破坏单元的集合 E_d

② 需要重新剖分的单元的集合 E_r

③ 剩余部分的单元集合 E_m

(2) 切屑的识别。切屑是指被已经破坏的单元的集合 E_d 中所有单元或部分单元包围的单元。如图 5.2 所示，图中虚线部分的单元按照步骤(1)的统计结果已经删除，这部分单元把一部分单元和本体隔离，这些被隔离的单元就是切屑。这些切屑也需要删除，放到集合 E_d 中去。需要说明的是，为了识别本体和切屑，一个必要的步骤是设置固定节点。所谓固定节点是边界上的那些用于固定待模拟工件的节点。破坏单元的集合 E_d 把网格中的三角形的切屑部分和本体部分分开。和固定节点相连的部分称为本体，另一部分就为切屑。

(3) 建立重新剖分单元集合 E_r 占有区域的背景网格。为了得到梯度变化均匀的单元，需要把 E_r 向周边区域进行扩展，被扩展的单元也需要加入到 E_r 作为需要重新剖分的区域。

(4) 确立需要重新剖分区域的边界。为了得到相容的有限元网格，这些边界上和本体部分相接触部分的节点是固定的，接触的边界边的内部不能产生新的节点，其它非接触部分则可以根据需要插入新的节点。

(5) 调用自适应剖分程序，生成内部单元和节点；和本体部分合并形成最终的重新剖分结果。

(6) 把上一时间步的分析结果的状态变量转化到新的网格中去。转换的方法如 5.2 所述。

5.3 状态变量的转换

转换过程分为两种情况：(a) 初始提供的状态变量如果是节点的状态变量，首先把节点的状态变量转化到单元上去，这样才能判断哪些单元需要重新剖分那些单元需要删除。由节点的状态变量转化为单元的状态变量比较简单，通常把单元的三个节点状态变量求算术平均。(b) 单元的状态变量向节点的转换。这个转化过程稍有些复杂，转换的方法如下：遍历网格中的所有节点，设节点 n_j 连接的单元为 $E=\{e_i, i=0...k-1\}$，n_j 的状态变量是这 k 个单元的加权平均值：

$$t_{nj}=\sum_{i=0}^{k-1} w_i t_{ei} \tag{5.1}$$

不同权的计算方法，计算得到的计算精度是不一样的，这里采用 Max[81]的计算方法：如图 5.3 所示，对于 $\Delta n_{j-1}n_i n_j$，权的计算公式为：

$$\begin{aligned} w_i &= \frac{\left|\mathbf{V}_i \times \mathbf{V}_{\mathbf{i+1}}\right|}{\left\|\mathbf{V}_i\right\|^2 \left\|\mathbf{V}_{\mathbf{i+1}}\right\|^2} \\ \mathbf{V}_i &= \overrightarrow{n_i n_{j-1}} \\ \mathbf{V}_{\mathbf{i+1}} &= \overrightarrow{n_i n_j} \end{aligned} \tag{5.2}$$

通过公式 5.1 和 5.2 把单元的状态变量转化到节点上去。生成新的网格后，需要把上一时间步的结果转化到新的网格中。首先遍历新生成的节点，对于任意节点首先找到它在旧网格中的位置，也就是属于旧网格的哪个三角形；再计算该节点的对于该单元的面积坐标，根据面积坐标和该单元三个节点的状态变量，插值得到新节点的状态变量值。如果用户需要单元的状态变量，可以按照方法(a)的方法计算得到。

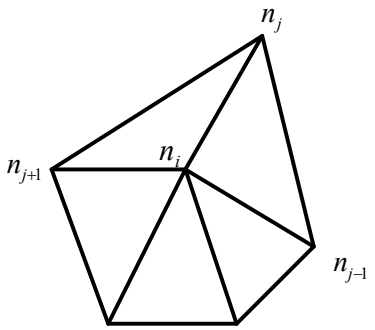


图 5.3 状态变量的转换

Fig.5.3 The transformation of state variables

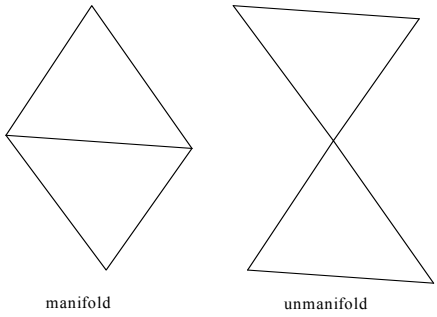


图 5.4 流网格与非流网格

Fig.5.4. Manifold and un-manifold triangle

5.4 其它应注意的问题

根据作者的经验，在进行破坏模拟的网格自适应生成中还有一些问题需要注意，这里给以总结：

5.4.1 非流网格的删除

如果网格中任何一条边只被一个三角形共享，那么这条边为边界边，这个三角形为边界三角形。如果两个相邻的边界三角形通过一个顶点相连(图 5.4 右)而不是通过边相连(图 5.4 左)，那么这样的三角形称为非流三角形。非流网格在有限元分析(Finite Element Analysis, FEA)中一般是不允许的，需要删除。删除的过程在自适应剖分的形成边界之前。

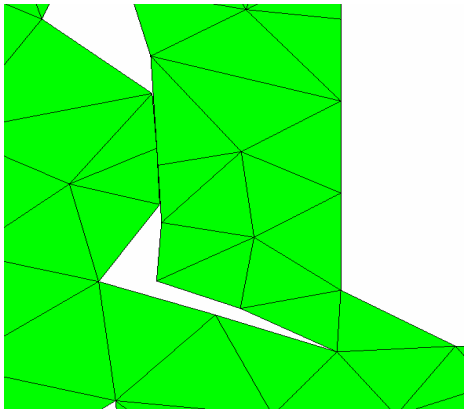


图 5.5 相交的边界
Fig.5.5 Intersected boundary edge

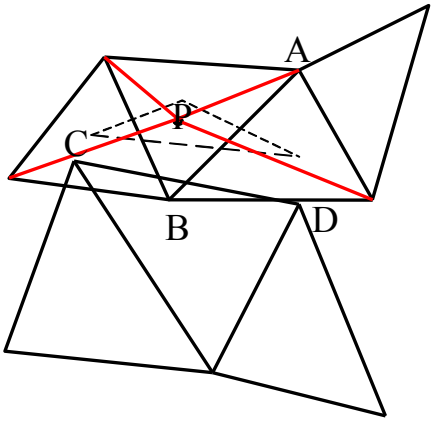


图 5.6 相交边界的修复
Fig.2-5 Revise of Intersected Edge

5.4.2 相交边界的修复

上一时间步计算得到的网格可能存在边界相交的情况(图 5.5)。这样相交的边界对网格生成程序来说是不允许的，必须进行修复。首先在形成边界以后得到边界线段的集合，对这些线段必须进行相交性检查，如果存在相交的线段，则必须进行修复。只有集合中不再存在相交的线段后才能进行下一步的自适应剖分。本文采用 Bartuschka[123]提出的快速扫描算法来找出相交的边界边。边界边的修复如图 5.6 所示，该图中线段 AB 和 CD 相交，修复的过程如下：

(1) 找到两条线段 4 个端点中和交点最近的点 B；
(2) 求出和点 B 连接的所有三角形；
(3) 分别求出这些三角形的重心；
(4) 求出这些重心坐标算术平均值作为新点 P 的坐标
(5) 移动点 B 到点 P

遍历所有相交的线段，执行上述操作，直到不再存在相交的线段为止。

5.5 自适应网格的生成

自适应网格生成的过程分成以下几个步骤：

(1) 按照上述步骤把所有需要重新剖分的单元收集起来，形成新的集合 E_r；

(2) 取出 E_r 外边界线段，构成一个封闭区域，这个区域就是需要重新剖分的区域；

(3) 根据状态变量和单元尺寸的函数，形成这个重新剖分区域的背景网格；

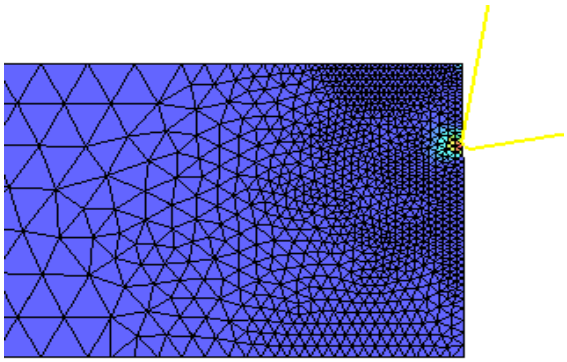
(4) 调用二维自适应剖分程序(参照第 4 章)，形成这一区域的网格；

(5) 把新的网格和剩余网格合并，形成最终的网格；

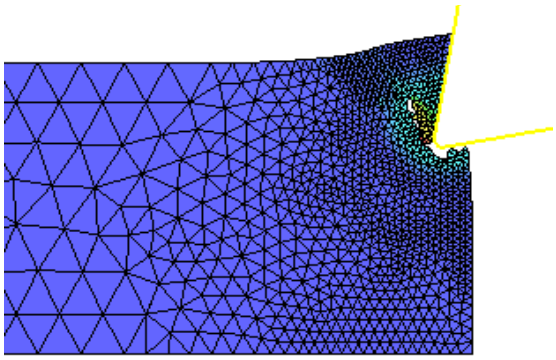
(6) 把状态变量转化到最终网格中去。

5.6 数值算例

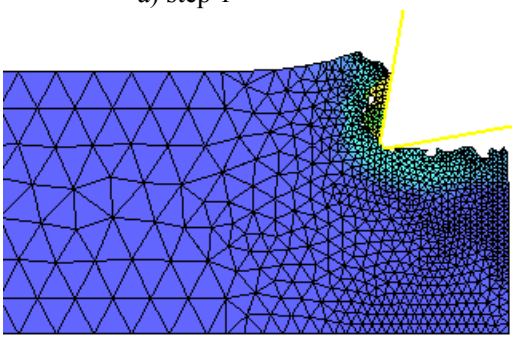
这里给出一个破坏模拟的算例，图 5.7a 是当刀具和工件接触后产生的初始网格，在这个初始剖分中，工件的剖分尺寸为 1.0，工件和刀具接触部分的单元尺寸为 0.1。这里分析程序采用 Abaqus®软件。图 5.7(b)(c)(d)(e)为分析过程中各个时间步的采用本文算法的网格(重新)剖分结果。整个模拟过程需要约 600 秒，其中网格(重新)剖分占用约 65 秒，也就是 1/10 的时间用在网格剖分上。



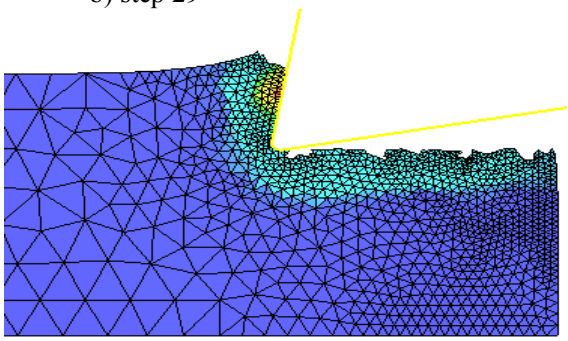
a) 初始网格
a) step 1



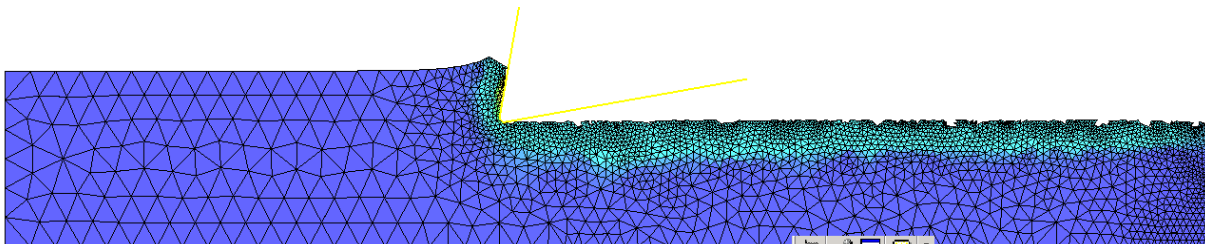
b) 第 29 步生成的结果
b) step 29



c) 第 120 步生成的结果
c) step 120



d) 第 222 步生成的结果
d) step 222



e) 第 997 步生成的结果

e) step 997

图 5.7 数值算例

Fig 5.7 Test Case

6.组合曲面的自适应网格生成

6.1 引言

复杂曲面的有限元网格生成有广泛的工程应用背景，同时又是三维实体网格生成的前提和基础。通常，曲面网格生成方法可以分为两大类：直接法和映射法。直接法是指直接在曲面上生成网格，而无需使用曲面的参数域。如 Lau 和 Lo[50, 85]采用波前推进法(AFT-Advancing Front Technique)对任意曲面进行直接三角剖分。直接法的主要问题是程序运行的效率和健壮性，它通常运用于没有参数化表达的曲面网格生成上。

映射法是目前曲面网格生成的最流行的方法。这种方法要求待剖分的曲面是一个可以在二维域上用双参数表达的参数化曲面，这样，曲面三维物理域上的点可以和曲面二维参数域上的点建立一一对应关系。这种方法首先使用通用的二维剖分程序在二维参数域生成网格，然后将剖分结果反向映射回曲面的物理空间从而形成曲面的最终网格。通常，这种直接映射方法对于简单的曲面能够生成质量较高网格，而对于曲率变化剧烈的复杂曲面，由于其映射函数在参数域的两个方向的非正交性，生成的网格的质量往往较差。为此 Lee, C. K[52]提出了一种首先将复杂曲面分解为一系列简单曲面，然后在这些简单曲面上生成网格，最后将这些简单曲面上网格合并形成复杂曲面的最终网格。然而对于一张曲率变化频繁的自由曲面，如何分解为一系列简单曲面本身也是一个十分复杂的过程。

为了使映射法生成的网格投影到曲面的物理域后保持较高质量，必须在参数域上生成网格时对网格的形状和大小进行很好的控制。实践证明，在网格的形状和大小控制方面，采用黎曼度量是一个简单有效的方法。并且，在定义待剖分域的黎曼度量后,就可以很好地控制待剖分域的网格生成，不但可以按用户指定的尺寸得到各向同性网格，而且可以根据需要生成各向异性网格。另外，黎曼度量方法是一个通用方法，它不但适用于曲面网格的剖分，对于三维实体剖分也同样适用。

曲面网格生成的另外一个难点是周期性曲面的网格生成，周期性曲面是指其参数域在 U 向或 V 向或 UV 两个方向是周期性的，也就是在有周期性的方向上，曲面的参数域是不封闭的。对这样的周期性参数曲面，为了在参数域上能够使用二维的剖分程序对参数域进行剖分，很多文献加上所谓的“虚边界”[10]。但是如果虚边界不巧靠近实边界如内孔时，该区域的网格剖分结果往往很差，不能用于有限元分析，有时甚至会造成程序崩溃。再者，虚边界的加入导致边界拓扑重构，这就增加了整个剖分程序的复杂性，降低了剖分程序的稳定性。

本章详细描述了基于黎曼度量的三维复杂组合参数曲面的自适应网格生成方法。在网格剖分的过程中同时考虑了用于生成自适应网格的三维黎曼度量和参数曲面自身的黎曼度量；同时，还考虑了模型中曲线和曲面的曲率变化以及模型中几何元素间的近亲关系。本章算法既可以生成大小一致的网格，也可以根据用户指定或计算得到的空间各点的用于控制网格生成尺寸和方向的黎曼度量来生成自适应曲面网格。

6.2 参数曲线和曲面

关于参数曲线和曲面的详细论述在很多文献[124, 125]中都有相关的论述，这里仅就组合曲面自适应网格剖分中用到的相关知识进行简单总结。三维空间中，参数曲线的一般形式为：

$$C(t)=(x(t),y(t),z(t)) \quad t\in[t_1,t_2] \tag{6.1}$$

通常曲面的边界是由这些边界曲线构成的，在网格剖分中，通常我们假设这些曲线是连续不自相交的。对于自适应剖分来说，这里还需要关心曲线的曲率，曲线的曲率可以这样计算：

$$K=\frac{\left|x'(t)y''(t)-x''(t)y'(t)\right|}{(x'^2(t)+y'^2(t))^{3/2}} \tag{6.2}$$

曲面是曲线的推广，类似地，曲面可以表示成双参数的形式：

$$S(u,v)=\begin{bmatrix}x(u,v)\\ y(u,v)\\ z(u,v)\end{bmatrix}\in\sum \quad \begin{bmatrix}u\\ v\end{bmatrix}\in\Omega \tag{6.3}$$

一般情况下参数域上的点和曲面上的点构成一一对应的关系（如图 6.1 所示）。对于这种一一对应关系不成立的点称为曲面的奇点。如锥的顶点，球的南北极点等。

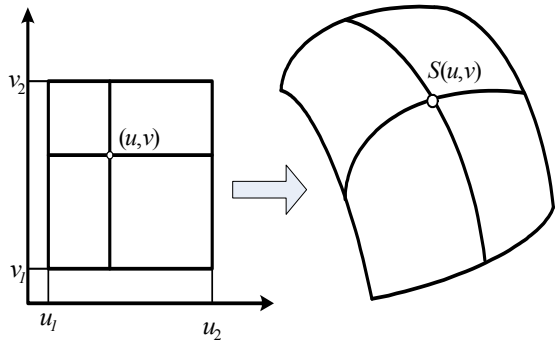


图 6.1 参数域上的点和曲面上的点的一一对应关系

Fig.6.1. Surface and its parametric space

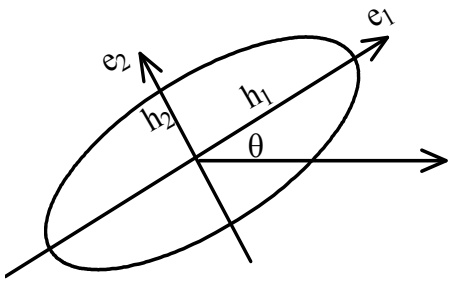


图 6.2 黎曼度量确定的控制椭圆

Fig.6.2. Ellipse defined by metric tensor

曲面在某点对 u 的偏导矢 $S_u(u,v)=\dfrac{\partial S(u,v)}{\partial u}$ 称为 u 向切矢，对 v 的偏导矢 $S_v(u,v)=\dfrac{\partial S(u,v)}{\partial v}$ 称为 v 向切矢。如果 $S_u(u,v)$ 和 $S_v(u,v)$ 不平行，既 $\left|S_u(u,v)\times S_v(u,v)\right|\neq 0$，就可以得到曲面在该点的切平面的单位法矢：

$$n(u,v)=\frac{S_u(u,v)\times S_v(u,v)}{\left|S_u(u,v)\times S_v(u,v)\right|} \tag{6.4}$$

曲面上 $\left|S_u(u,v)\times S_v(u,v)\right|\neq 0$ 的点称为曲面的正则点，而曲面上 $\left|S_u(u,v)\times S_v(u,v)\right|=0$ 的点是曲面的一种奇点。这种奇点和曲面上一阶导矢为零的奇点不同，这种奇点可能是因为两个偏导矢平行或退化引起，可以通过重新参数化消矢。

6.2.1 参数曲面的黎曼度量

假如：$r_u=\partial S/\partial u$，$r_v=\partial S/\partial v$ 那么：

$$dS=(dx,dy,dz)^T=\begin{pmatrix}\dfrac{\partial x}{\partial u} & \dfrac{\partial x}{\partial v}\\ \dfrac{\partial y}{\partial u} & \dfrac{\partial y}{\partial v}\\ \dfrac{\partial z}{\partial u} & \dfrac{\partial z}{\partial v}\end{pmatrix}\begin{pmatrix}du\\ dv\end{pmatrix}=(r_u\ r_v)dU \tag{6.5a}$$

由此可以得到曲面的第一类基本量：

$$dS^TdS=dU^T\begin{bmatrix}r_u.r_u & r_u.r_v\\ r_v.r_u & r_v.r_v\end{bmatrix}dU=dU^T\begin{bmatrix}E & F\\ F & G\end{bmatrix}dU \tag{6.5b}$$

其中：$E=r_u.r_u, F=r_u.r_v, G=r_v.r_v$ 满足 $E>0, EG-F^2>0$ 时，称为是参数域 D 的一个黎曼度量(Riemann metric)。E、F、G 称为黎曼度量系数，r_u 和 r_v 分别表示曲面上某点在 U 向和 V 向的一阶偏导矢。矩阵：$M=\begin{bmatrix}E & F\\ F & G\end{bmatrix}$ 通常称为 Riemannian 度量阵（黎曼度量）也叫度量张量(Metric Tensor)。

因为度量阵 M 是一个是对称阵，并且 $E>0, EG-F^2>0$，所以 M 有两个正特征值：$\lambda_1,\lambda_2>0$。黎曼度量的相关计算可以参看第 4 章的有关章节，这里就不再赘述。

6.2.2 曲面上点的主曲率性质

曲面上点切平面的单位法矢可以由曲面上该点在U向和V向的一阶偏导矢 r_u 和 r_v 差乘得到：

$$n(u,v)=\frac{r_u\times r_v}{|r_u\times r_v|} \tag{6.6a}$$

法线的变化即法曲率可以表示为：

$$dn=\frac{\partial n}{\partial u}du+\frac{\partial n}{\partial v}dv=n_u du+n_v dv=(n_u\ n_v)dU \tag{6.6b}$$

那么可以得到曲面的第二类基本量：

$$dn^T dS=dU^T\begin{bmatrix}n_u r_u & n_u r_v\\ n_u r_v & n_v r_v\end{bmatrix}dU=-dU^T\begin{bmatrix}L & M\\ M & N\end{bmatrix}dU \tag{6.6c}$$

其中：$L=-n_u r_u=-nr_{uu}$，$M=-n_u r_v=-nr_{uv}$，$N=-n_v r_v=-nr_{vv}$。实对称方阵 $Mc=\begin{bmatrix}L & M\\ M & N\end{bmatrix}$ 称为曲率张量(Curvature Tensor)。曲面上一点的法曲率总是沿所取的某一个方向的法曲率，曲面上一点有无数多个方向，就有无数多个法曲率。除了平面和球面外，这些法曲率一般是不同的，其中最大值和最小值称为主曲率，它们所在的方向称为主方向，可以由如下方程确定：

$$k_n M\begin{pmatrix}a\\ b\end{pmatrix}=M_c\begin{pmatrix}a\\ b\end{pmatrix} \tag{6.6e}$$

即：

$$(k_n M-M_c)\begin{pmatrix}a\\ b\end{pmatrix}=\begin{bmatrix}k_n E-L & k_n F-M\\ k_n F-M & k_n G-N\end{bmatrix}\begin{pmatrix}a\\ b\end{pmatrix}=0 \tag{6.6f}$$

主曲率可以由如下方程得到：

$$\det(k_n M-M_c)=0 \tag{6.6g}$$

解方程(6.6f)得到：$w^T=(a\ b)$，那么其中一个主方向为：

$$d_1=\frac{ar_u+br_v}{|ar_u+br_v|} \tag{6.6h}$$

因为两个主方向和法线方向相互垂直，这样另外一个主方向为：

$$d_2=\frac{d_1\times n}{|d_1\times n|} \tag{6.6i}$$

6.2.3 曲面上点的曲率的近似计算

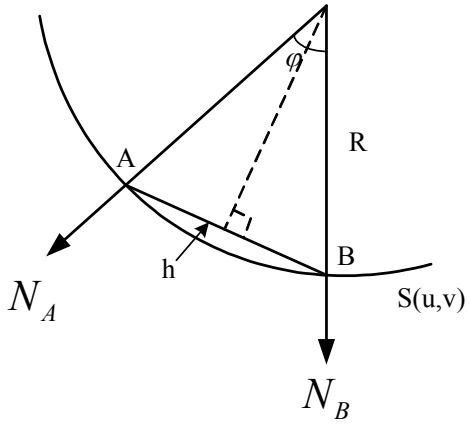


图 6.3 曲面上两点的曲率的近似求法

Fig.6.3 The approximate curvature of point on surface

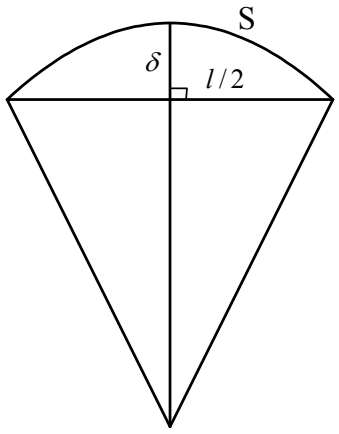


图 6.4 网格的贴近度

Fig.6.4 The approximation of mesh

用上述方法计算曲面上某点的曲率，是比较耗时的，在网格生成的过程中可以用如下方法近似曲面上某点的曲率。设φ是用户指定的网格上两点的最大张角，设在曲面上两点 A、B 的单位法向量为：N_A, N_B距离为 h，这可以用公式 6.5 计算得到。那么 A、B 两点曲率半径可以近似为：

$$R \approx \frac{h/2}{\sqrt{(1-N_A.N_B)/2}} = \frac{h}{\sqrt{2(1-N_A.N_B)}} \tag{6.7a}$$

上述方法计算得到的曲率半径对二次曲面是精确的，对于其它曲面是个近似的结果。但是对于网格生成过程，这种近似已经足够了。这样对用户给定的最大张角，曲面上 A、B 两点的单元尺寸可以这样得到：

$$\delta_\varphi = 2R\sin(\frac{\varphi}{2}) \tag{6.7b}$$

6.2.4 由曲率控制的单元尺寸

曲面网格生成的过程中，一个重要的目标是生成的网格要尽量和曲面贴近。如图 4 所示，这可以用网格和曲面的贴近程度来度量。一般来说可以用如下公式来表示网格的贴近度：

$$\eta = \frac{s-l}{s} \tag{6.8a}$$

其中s表示两点间的弧长，l表示两点的弦长，网格生成的过程中可以人为指定一个系数ε，使得$\eta \le \varepsilon$这样：$l \ge (1.0-\varepsilon)s$，可以证明[126]弧长s，弦长l，曲率k以及曲率精度系数ε满足如下方程：

$$l \geq \frac{(1.0-\varepsilon)\sqrt{40\ (1-\sqrt{(1-1.2\varepsilon)})}}{k} = \frac{g(\varepsilon)}{k} \tag{6.8b}$$

其中 $g(\varepsilon)=(1.0-\varepsilon)\sqrt{40(1-\sqrt{(1-1.2\varepsilon)})}$

在考虑曲面的曲率情况下，用户需要指定允许的最大剖分尺寸 $h_{g\max}$ 和最小剖分尺寸 $h_{g\min}$ 这样就得到考虑用户指定最大最小剖分尺寸的用户容许曲率：

$$\begin{aligned} k_{g\max} &= \frac{g(\varepsilon)}{h_{g\min}} \\ k_{g\min} &= \frac{g(\varepsilon)}{h_{g\max}} \end{aligned} \tag{6.8c}$$

曲面上某点用于计算网格尺寸的曲率为：

$$\begin{aligned} \bar{k}_{\max} &= \min[k_{g\max}, \max(|k_{\max}|, k_{g\min})] \\ \bar{k}_{\min} &= \min[k_{g\max}, \max(|k_{\min}|, k_{g\min})] \end{aligned} \tag{6.8d}$$

在任意方向上曲面上某点的网格的尺寸为：

$$h(\theta) = \frac{g(\varepsilon)}{\bar{k}_{g\max}\cos^2\theta + \bar{k}_{g\min}\sin^2\theta} \tag{6.8e}$$

6.3 三维空间的黎曼度量

三维空间中的黎曼度量是二维空间的推广，二维空间的黎曼度量在第 4 章已经有较为详细的论述，这里简单介绍一下三维空间的黎曼度量。在三维空间中可以用一个3×3的实对称阵来表示空间某点 P 的黎曼度量：

$$M_{3D} = \begin{bmatrix} a & d & e \\ d & b & f \\ e & f & c \end{bmatrix} = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_3 \end{bmatrix} \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}^T \tag{6.9a}$$

这里 $Det(M_{3D})>0, \lambda_i>0$， $(e_i,\lambda_i), i=1,2,3$ 是对应的特征向量和特征值。在 e_i 方向上单元的尺寸为： $h_i = 1/\sqrt{\lambda_i}$ 。这样 M_{3D} 就决定了空间在某点上单元的尺寸和生成方向。这个黎曼度量一般是由用户指定或由于自适应的要求计算得到。

对于参数曲面，本文首先在参数域上生成网格，然后投影到曲面的物理域上。上面介绍了曲面参数域的黎曼度量，这样在生成参数域上的网格，必须同时考虑曲面本身的黎曼度量和用户设置以及自适应要求的三维黎曼度量。把上述两个度量合成就形成曲面上某点 P 的黎曼度量 M_P：

$$M_P = \begin{bmatrix} r_u & r_v \end{bmatrix}^T M_{3DP} \begin{bmatrix} r_u & r_v \end{bmatrix} = \begin{bmatrix} E' & F' \\ F' & G' \end{bmatrix} \tag{6.9b}$$

M_P是一个实对称阵，其两个特征值: $\lambda_1,\lambda_2>0$。设和λ_1,λ_2对应的特征向量是e_1,e_2,则有：

$$M_P=[e_1,e_2]\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}[e_1,e_2]^T=[e_1,e_2]\begin{bmatrix}\frac{1}{h_1^{\,2}} & 0\\ 0 & \frac{1}{h_2^{\,2}}\end{bmatrix}[e_1,e_2]^T \tag{6.9c}$$

这样就把三维空间的黎曼度量和曲面本身的黎曼度量统一合成为曲面上点的黎曼度量。这和第 4 章所论述的 2 维空间的黎曼度量是一致的，从而二维自适应算法可以运用到曲面参数域的剖分。

6.4 三维空间的平衡八叉树背景网格

和二维空间的自适应剖分类似，在组合曲面的剖分之前，首先需要建立一个覆盖整个待剖分域的背景网格。本文在采用平衡的八叉树来作为组合曲面的背景网格。作为四叉树在三维空间的推广，八叉树的很多概念和四叉树是类似的。首先取得整个组合曲面的外包围盒作为整个八叉树的根(root)，如果取得组合曲面的外包围盒是个近似的结果，为了更好的覆盖整个剖分域，可以对这个包围盒乘以一个扩展系数，以确保这个包围盒能覆盖整个模型。然后按用户指定的剖分尺寸(或者默认的尺寸)来均分这个包围盒，形成一个均分的初始八叉树。和四叉树类似，本文把模型的尺寸信息存在八叉树叶子节点的角点上，对于模型中任意一点的尺寸的计算，可以首先找到该点所在的叶子节点，然后通过该叶子节点八个角点的尺寸插值得到：

$$h=\sum_{i=1}^{8}h_iN_{\mathrm{i}} \tag{6.10a}$$

其中：

$$\begin{aligned}
N_1&=(1-\xi)(1-\eta)(1-\zeta)/8 \qquad N_5=(1-\xi)(1-\eta)(1+\zeta)/8\\
N_2&=(1+\xi)(1-\eta)(1-\zeta)/8 \qquad N_6=(1+\xi)(1-\eta)(1+\zeta)/8\\
N_3&=(1-\xi)(1+\eta)(1-\zeta)/8 \qquad N_7=(1-\xi)(1+\eta)(1+\zeta)/8\\
N_4&=(1+\xi)(1+\eta)(1-\zeta)/8 \qquad N_8=(1+\xi)(1+\eta)(1+\zeta)/8
\end{aligned} \tag{6.10b}$$

其中：$\xi=2(x-x_c)/sz$，$\eta=2(y-y_c)/sz$，$\zeta=2(z-z_c)/sz$，sz为子节点的大小。为了得到梯度变化均匀的网格，八叉树的平衡是八叉树建立过程中一个重要的一步。八叉树的平衡分为弱平衡和强平衡两种，为了更清楚地说明问题，这里取八叉树的一个正视图(图 6.5)来解释弱平衡和强平衡。所谓弱平衡如图 6.5(a)所示，共享边(八叉树的面)的相邻叶子节点的层数差不大于 1；强平衡如图 6.5(b)所示, 共享点(八叉树的边)的相邻叶子节点的层数差不大于 1。一个八叉树的叶子节点的相邻节点最多为 26 个，其中位于同一父节点的相邻节点有 8 个，这样最多可能不平衡的节点是 26-8=18 个。平衡算法首先要

找出这些不平衡的节点，在分裂当前节点的同时也要分裂相邻的不平衡节点。这个分裂过程是个递归过程，直到整个八叉树达到强平衡为止。这样，在某点上设置剖分尺寸时，整个八叉树平衡过程是为：

算法 6.1 八叉树的平衡算法

(1) 假设计算得到空间中点 $P(x,y,z)$ 的尺寸为 s

(2) 查找 $P(x,y,z)$ 所在的叶子节点 n_k，根据公式 6.10 插值计算得到尺寸为 h

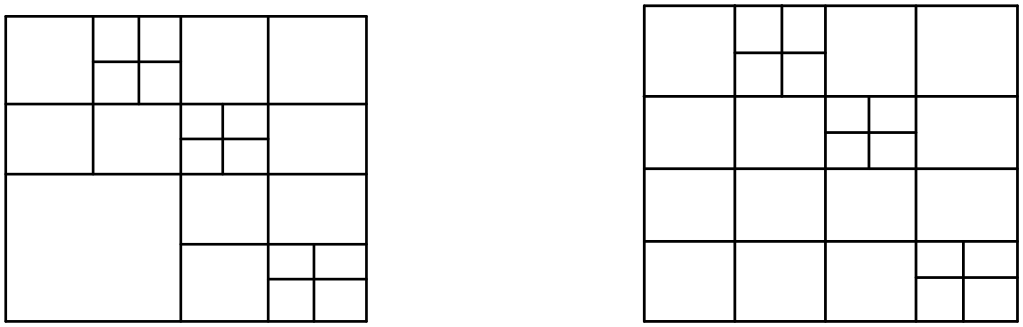
(3) 如果 $s<\lambda h$，(λ 是用户设定的网格尺寸变化的最大梯度，默认 $\lambda=2.0$)

① 找出与 n_k 不平衡的所有叶子节点 $N=\{n_0,n_1...n_{i+1}\},i\le 18$

② 分裂 n_k 与 $N=\{n_0,n_1...n_{i+1}\},i\le 18$

③ 重复执行 2），3）直到 $s\ge\lambda h$ 为止

(4) 返回



(a) (b)

图 7.5 两种形式的平衡八叉树

Fig.7.5 Two types of balanced Octree

6.5 三维空间背景网格的建立

为了生成整个模型的自适应网格，这里首先要建立三维空间的黎曼度量场。当生成某一曲面的网格时，就可以不但考虑曲面自身的几何特性，同时也考虑当前曲面作为整个组合曲面的一部分，整个模型对这一曲面施加的自适应要求。在这里我们主要考虑几何自适应和用户指定尺寸自适应。几何自适应包括曲率和近亲两个方面，为了得到组合曲面中模型的这些信息，就需要对组合曲面的特征进行扫描。

6.5.1 特征扫描算法

首先需要扫描模型的几何元素，包括点、曲线、曲面。扫描的主要目的是获得几何元素的几何特征并根据这些特征获得相应的几何尺寸，从而把这些尺寸信息添加或更新到背景网格中去。特征扫描算法如下：

算法 6.2 特征扫描算法

(1) 设模型中所有顶点的集合为 $V=\{v_i,i=1...n_v\}$，曲线的集合为 $C=\{c_i,i=1...n_c\}$，曲面的集合为 $S=\{s_i,i=1...n_s\}$。

(2) 设模型中所允许的尺寸为$[h_{\min},h]$，最大张角为$\alpha_{\max}$。这些尺寸可以是用户指定，对于最大张角，默认$\alpha_{\max}=30^o$。如果用户不指定剖分尺寸h，也可以根据模型外包围盒对角线的长度估算出一个合适的h及$h_{\min}$。

(3) 首先扫描顶点，对于当前顶点，网格在该点的尺寸为下列尺寸的最小值：

① 全局尺寸h;

② 所连接曲线长度的最小值;

(4) 扫描所有曲线，扫描的尺寸为曲线端点处(步骤 3 中已设定)尺寸的最小值。对曲线上经过的扫描点，设定其尺寸为下列尺寸的最小值：

① 全局尺寸h;

② 根据曲率计算所得到的单元尺寸；

③ 根据近亲关系计算得到的单元尺寸(参见 6.5.4)。

(5) 扫描所有曲面，对曲面上经过的扫描点，设定其尺寸为下列尺寸的最小值。

① 全局尺寸h;

② 根据曲率计算所得到的单元尺寸;

③ 根据近亲关系计算得到的单元尺寸(参见 6.5.4))。

对于顶点的扫描，因为不需要考虑曲率，只需要考虑近亲关系。这里主要论述曲线和曲面的扫描过程。

6.5.2 曲线的曲率扫描

对于参数曲线首先通过背景网格取得曲线两个端点的最小值$s_{\min}$，按照这个尺寸从$t=a$开始扫描。在已知当前曲线上点的参数t_i，扫描长度s，可以用牛顿迭代法来求出下一个点t_{i+1}的位置：

(1) 设曲线的长度为l,

(2) 定义曲线在某点的速率为：

$$v(t)=\sqrt{(\frac{\partial x}{\partial t})^2+(\frac{\partial y}{\partial t})^2+(\frac{\partial z}{\partial t})^2} \tag{6.11a}$$

(3) 那么，迭代的初始值取 $t_{i+1}=(1-\lambda)a+\lambda b$，其中 $\lambda=s/l$

(4) 当 $(\left|l(t_i,t_{i+1})-s\right|<\varepsilon)$

取 $\Delta l = l(t_i, t_{i+1}) - s$

$t_{i+1} = t_{i+1} - \Delta l / v(_{t+i})$

(5) 返回 t_{i+1}

$l(t_i, t_{i+1})$表示t_i到t_{i+1}的曲线的弧线距离，可以通过曲线对象相应的 API 或者通过龙贝格积分法计算得到。在扫描的过程中，对于曲线上的点，按照公式 6.2 计算出曲线在该处的曲率K_i，根据用户所规定(或默认)的最大张角φ，可以按照如下公式计算出在该点的处的尺寸。

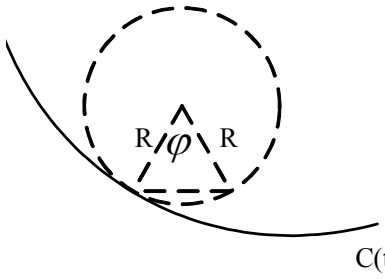
$$s_i = \min(h_{\min}, 2R\sin(\varphi/2)) = \min(h_{\min}, 2\sin(\varphi/2)/R) \qquad (6.11b)$$

曲线上相邻两点(P_i, P_{i+1})的尺寸s_i，s_{i+1}的比值$\eta = s_{i+1}/s_i$应该满足$\eta \in [1/\lambda, \lambda]$，$\lambda$是用户设定的网格变化的最大梯度（默认$\lambda = 2.0$），如果$\eta \notin [1/\lambda, \lambda]$，那么在这两个点的尺寸应该进行调整，使之满足$\eta \in [1/\lambda, \lambda]$。这里假设$\eta > \lambda$即$s_{i+1} > s_i$，那么$s_{i+1}$的尺寸这样调整：

(1) 计算P_i，P_{i+1}的黎曼长度$l(P_i, P_{i+1}) = \frac{2}{3}\frac{l_i^2 + l_i l_{i+1} + l_{i+1}^2}{l_i + l_i}$，其中$l_i = \frac{\|P_i P_{i+1}\|}{s_i}$，$l_i = \frac{\|P_i P_{i+1}\|}{s_{i+1}}$

(2) 按照 $s_{i+1} = s_i \lambda^{l(P_i, P_{i+1})}$调整$P_{i+1}$点的尺寸。

$\|P_i P_{i+1}\|$表示(P_i, P_{i+1})的欧拉距离。上述方法很容易推导出$\eta < 1/\lambda$的情况。



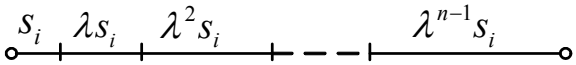


图 6.6 张角、网格尺寸和曲线曲率的关系图　　图 6.7 相邻节点的尺寸关系

Fig.7.6. Element size, curvature and span angle　　Fig.6.7 The sizes of adjacent nodes

6.5.3 曲面的曲率扫描

根据曲面的一阶导数计算得到黎曼度量，只能反映曲面的参数域和物理域之间映射函数的非线性特征，并不能反映曲面的曲率变化。在曲面自适应网格生成中，曲率是重要考察的对象。这样在自适应剖分之前必须对每个曲面进行扫描，把反映曲面曲率变化的曲面上的点存储到背景网格中，曲面的扫描算法如下：

(1) 设曲面的参数域为：$u \in [u_1, u_2], v \in [v_1, v_2]$，以这两个区间作为四叉树的根(root)

(2) 根据曲面公式计算四叉树的根节点 4 个角点的三维坐标，根据这些坐标，从模型的背景网格网格中取得四叉树的根节点的 4 个剖分尺寸：h_0,h_1,h_2,h_3，取 $h=\min(h_0,h_1,h_2,h_3)$ 作为扫描尺寸

(3) 把四叉树均分成 $n\times n$ 的栅格，这里可取 $n=10$

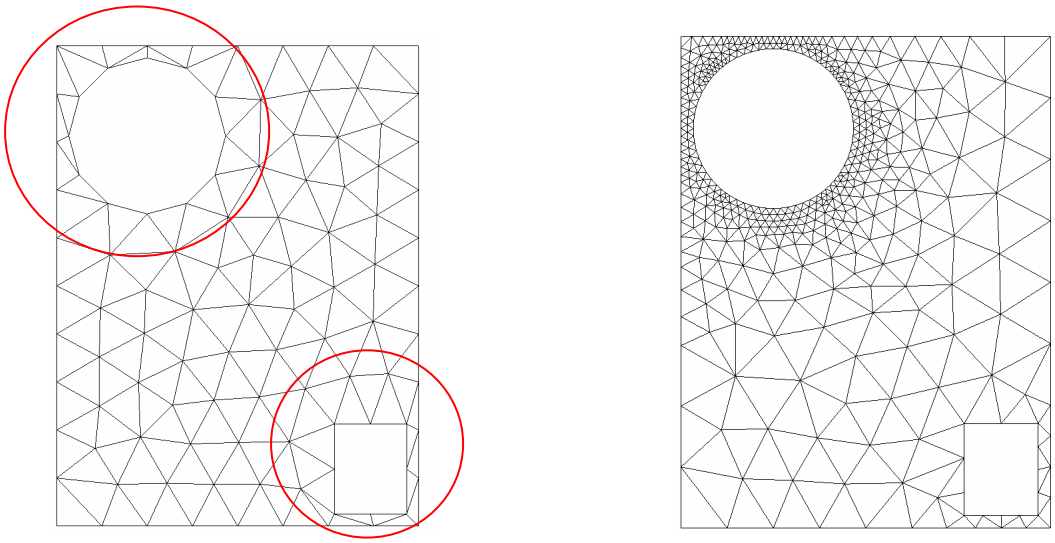
(4) 遍历四叉树的每个叶子节点，计算叶子节点的每条边对应在曲面上的曲线长度 $l_i(i=0...3)$，这可以通过公式 6.5 计算得到。

(5) 如果任一条边的长度 $l_i>\lambda h$，那么平衡分裂该子节点

(6) 遍历四叉树的每个叶子节点，根据曲面公式计算四叉树的根节点 4 个角点的三维坐标，根据这些坐标，从模型的背景网格网格中取得四叉树的根节点的 4 个剖分尺寸：$h(0),h(1),h(2),h(3)$。按 6.4 节描述的方法计算每个根据曲率要求得到的剖分尺寸 $\delta(0),\delta(1),\delta(2),\delta(4)$，由下列公式在背景网格中设置这 4 个点的尺寸：

$$h_{new}(i)=\max(h_{\min},\min(h(i),\delta(i))) \tag{6.12}$$

(7) 对当前叶子节点的任何两个相邻角点，尺寸比值 $\eta=h_{new}(i+1)/h_{new}(i)$ 应该满足 $\eta\in[1/\lambda,\lambda]$，如果不满足该条件，采用曲线扫描中类似的方法使之满足。



(a)不考虑近亲的剖分　　(b)考虑近亲的剖分

(a) Mesh without proximity　　(b) Mesh with proximity

图 6.8 近亲影响的网格尺寸

Fig.6.8 The influence of proximity

6.5.4 几何近亲扫描

曲线或曲面上的任意一点和组合曲面中的任一几何元素过近时，该曲线和过近的几何与元素是近亲。在组合曲面的自适应剖分过程中，应该考虑近亲。图 6.8(a)是不考虑

近亲的情况的模型某个表面的剖分结果，明显可以看出图中画圈的区域两个几何元素临近的地方，网格质量较差，图 6.8(b)是采用相同剖分尺寸考虑近亲生成的网格，可以看出，在这些区域网格质量明显提高。

要计算当前扫描点和模型中最近的几何元素的距离是比较费时的。一个快速的做法是：在实际扫描的过程中，把所有已扫描的点保存起来，在曲线和曲面曲率扫描的过程中，当要设置点 P 的尺寸 h_P 时，首先以点 P 为中心，用户设置的尺寸 h 为边长，建立一个正方体（如图 6.9 所示），收集所有落在这个正方体内的已扫描点，然后计算点 P 和这些点的最近距离 $d_{\min}$，最后在背景网格中设置尺寸为

$$h = \max(h_{\min}, \min(h_P, d_{\min})) \qquad (6.13)$$

按照上述方法，扫描所有的几何元素，把扫描得到的尺寸存储到平衡八叉树的背景网格中，这样在每个参数曲面的剖分过程中，考虑背景网格已经设置的网格尺寸，就可以生成组合曲面的自适应网格了。

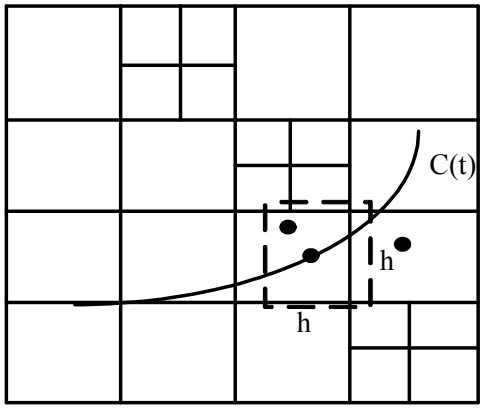


图 6.9 近亲尺寸设置

Fig. 6.9. Meshing process of Surfaces

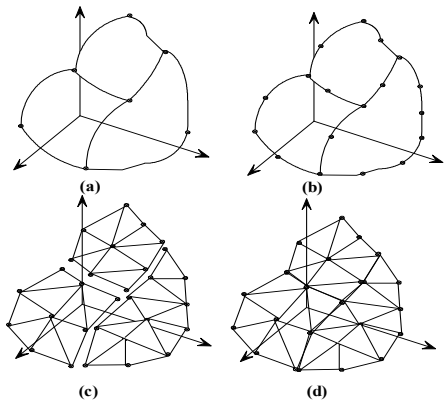


图 6.10 曲面网格的生成过程

Fig.6.10. Meshing process of Surfaces

6.6 参数曲面的自适应网格生成

与大多数曲面网格生成方法类似，本文中对实体表面的网格生成总体上分为以下四个步骤：

(1) 取得实体各曲面的边界曲线(图 6.10(a))；

(2) 根据黎曼度量对各曲面边界曲线进行离散化(图 6.10(b))，离散化的过程中本文需要满足左手侧规则，离散完后需要离散点投影到曲面的参数空间；

(3) 对每个曲面在参数空间进行剖分，并把剖分结果反向投影到曲面的物理空间，形成各个曲面的表面网格(图 6.10(c))；

(4) 把同一实体上的各个曲面的网格合并形成该实体的整个表面的网格(图 6.10(d))。

6.6.1 参数曲面背景网格的建立

由于本文的曲面网格的生成首先在参数域上完成，然后投影到曲面的物理域形成三维的曲面网格，因此在组合曲面网格生成以前首先要生成曲面参数域的背景网格。在参数域本文采用第 4 章论述的自适应剖分算法，因此背景网格也是采用平衡的四叉树。可由如下过程完成曲面黎曼度量场的建立：

(1) 设曲面的参数域为：$u \in [u_1, u_2], v \in [v_1, v_2]$，以这两个区间作为四叉树的根(root)，把根节点作为当前叶子节点。

(2) 遍历四叉树当前叶子的四个角点，对于任何一个没有计算过黎曼度量的角点：

① 根据曲面公式计算该角点的三维坐标，并从模型的背景网格中得到该点的剖分尺寸 h，这样该点的三维黎曼度量为 $M_{3D} = \mathbf{I}/h^2$，其中 $\mathbf{I}$ 表示 3×3 的单位阵

② 计算该点的 u 向和 v 向的一阶偏导矢 r_u 和 r_v

③ 按照公式 6.9(b)合成 M_{3D} 和 r_u， r_v 计算该点的黎曼度量

(3) 按照公式 6.5(d)(e)计算当前四叉树叶子节点四条边的长度 l_0, l_1, l_2, l_3，取 $l_{\min} = \min(l_0, l_1, l_2, l_3)$

(4) 如果 $l_{\min} > \sqrt{2}$，则平衡分裂当前叶子节点，并对分裂后得到的 4 叶子节点分别作为当前叶子节点，重复执行 2），3）步骤

以上述过程遍历模型中的每一个曲面，建立这些曲面的黎曼度量场，存入相应的集合中，待自适应剖分过程使用。

6.6.2 边界曲线的离散化

在进行组合曲面的剖分之前，为了得到曲面与曲面间相容的有限元网格，必须首先对模型中的所有曲线进行离散化。因为通常一个曲面的边界曲线同时也是另外一个曲面的边界曲线，为了得到曲面间相容的网格，边界曲线只能离散化一次。在这里我们假设，整个模型中的几何和连接信息都是通过 B-Rep 的数据结构来组织的。那么整个模型的边界离散化过程是这样的：

(1) 设已离散化的 B-Rep 边的集合为 E

(2) 遍历模型中所有的面，对当前面 f_i

① 取出 f_i 的黎曼度量场

② 遍历 f_i 中的所有边，对于边 e_i，如果不在集合 E 中那么：

(a) e_i 放入集合 E 中

(b) 把 e_i 对应的三维曲线投影到 f_i 的参数域中，形成参数域的二维曲线

(c) 调用第 4 章的二维域的边界离散化算法，离散化该曲线

(d) 调用曲面公式，得到该曲线的三维离散化结果

(3) 返回

对于周期性曲面，在步骤 2)-b)-III.中计算长度时，在参数域的上下边界附近，需要进行迁移操作，具体迁移算法在下一节介绍。

6.6.3 周期性曲面的特殊性

如前所述，周期性曲面是指其参数域在 U 向或 V 向或 UV 两个方向是周期性的，也就是在有周期性的方向上，曲面的参数域是不封闭的。如图 6.11 所示，对这样的周期性参数曲面，为了在参数域上能够使用二维的剖分程序对参数域进行剖分，很多文献一般需要加上所谓的“虚边界”。但是如果虚边界周围有“洞”的话，该区域剖分的单元质量往往很差，有时候甚至不能用于有限元分析。图 6.12 是 Ansys®软件对两个相交的圆柱面的剖分结果，在圆柱面上可以明显看到一根直线(程序内部添加的虚边界)，在圈出的区域内网格质量较差。再者虚边界的计算往往也是比较复杂的，这就增加了整个剖分程序的复杂性，降低了剖分程序的稳定性。

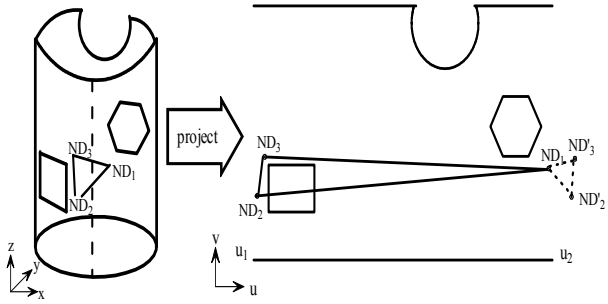


图 6.11 周期性曲面及其参数域

Fig.611.Closed surface and its parametric space

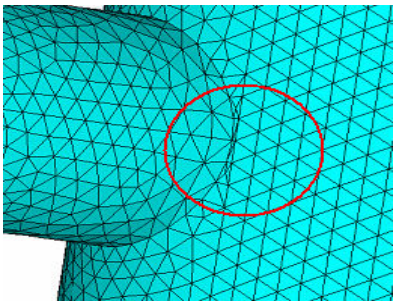


Fig.6.12 虚边界问题

Fig.6.12 Problem of virtual boundary

为了解决周期性曲面的剖分问题，本文扩展了传统的 AFT 方法，主要是添加了点和线段的迁移算子，而无需添加虚边界。如图 6.13 所示，以圆柱面为例，假设这张圆柱面在 U 向具有周期性，周期为 T。在曲面上有两个距离较近的点 ND_1, ND_2，在曲面的参数域上点 ND_1, ND_2 分别在 U 向的上下边界周围，因而相距较远。这里定义两个迁移算子，点的迁移算子和线段的迁移算子：

(1)点的迁移算子：对于周期性曲面，曲面上的一条线段的两个端点在参数域上，分别落在周期方向的上下边界周围时，那么在参数域上由黎曼度量计算距离以及进行其它几何判断时是不准确的，需要把这两点迁移到同一半周期内。以如图 6.11 中的 ND_1, ND_2 为例，如下应用迁移算子：

① 计算两个点在参数域上的 U 向距离： $d = ND_1(u) - ND_2(u)$

② 如果 $|d| < \frac{T}{2}$ 返回

③ 否则，如果 $d > \frac{T}{2}$ 那么 $ND_2(u) = ND_2(u) + T$

④ 否则，如果 $d < -\frac{T}{2}$ 那么 $ND_2(u) = ND_2(u) - T$

⑤ 返回

(2) 前沿线段的迁移算子：前沿线段的迁移算子，也叫迁移算子 F，当线段需要迁移时调用该算子，这里假设线段的两个端点是 ND_1， ND_2，在周期性曲面的参数域上，假设 ND_2 迁移后得到的点为 ND_3，（如图 11 所示），前沿线段的迁移算子如下：

① 对线段的两个端点 ND_1, ND_2, 调用点的迁移算子，迁移后 ND_2 的位置为 ND_3

② 对点 ND_1， ND_3, 调用点的迁移算子

③ 对点 ND_2， ND_3, 调用点的迁移算子

④ 返回

需要注意的是：(1) 如果曲面在 V 向是有周期性的，上述两个算子就作用于 V 向；如果曲面在 U 向和 V 都是有周期性的，上述两个算子就分别作用于 U 向和 V 向。使用上述两个算子目的是：把不在同一半周期内的点迁移到同一半周期内，以消除这些点和其它点的差别。这样，当运用 AFT 方法时，就可以把这些点和前沿作为普通的点和前沿使用了。

6.6.4 周期性曲面的边界离散化

上面章节讲述了非周期性曲面的边界曲线的离散化方法，但是对于周期性曲面（如圆柱、圆锥、球以及某些旋转曲面），由于其参数域是不封闭的，所以其离散化的过程也与一般曲面有所不同。为了避免虚边界带来的网格质量变差以及程序稳定性方面的问题，本文通过引入迁移算子来实现周期性曲面的剖分。本文把周期性曲面分为两大类

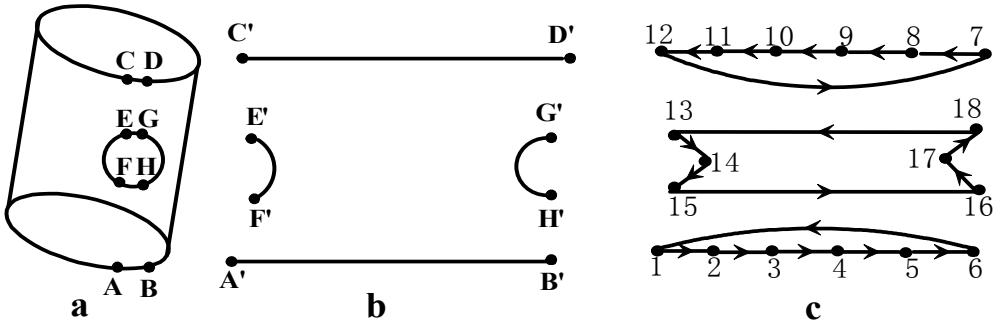
(1) 具有边界的曲线的周期性曲面(*U* 向、*V* 向或者*U V* 两个方向)；

(2) 没有边界曲线的周期性曲面。

下面分别介绍这两种曲面边界曲线的离散化方法。

(1) 具有边界的曲线的周期性曲面的边界离散化

这里以单向周期(*U* 向或者*V* 向)的曲面为例，对于这样的曲面，曲面上存在一条曲线在参数域上分别对应于参数域的上下边界。因为该曲面有一个方向是周期性的，这样在该方向上曲面是不封闭的。圆柱是该类型曲面的一个典型例子，在周期方向上，这个曲面的区间是[0,2π] (ACIS 内核)或者[-π, π] (Parasolid 内核)。图 6.13 是有一个洞的圆柱面及其参数域的离散化结果。由图可以看出，在曲面物理域两个距离较近的点，在参数域上可能相距较远。图 6.13 中点 A 和点 B 距离较近，但是对应的参数域上的点 A’、B’却相距较远。图 6.13c 是该曲面在某一剖分尺寸下的离散化结果。可以看出，其在参数域上并不是封闭的，应而不能直接使用二维的剖分程序。在本文中，因为 A-B, C-D, E-G 和 F-H 是相邻的节点，所以他们也应该作为前沿存在，这样不但诸如 1-2, 2-3 等边是前沿，而且像 12-7,6-1,15-16 以及 18-13 也是 AFT 的前沿。图中的箭头表明了这些边的方向，这一例子离散化以后，节点序列为：1-2-3-4-5-6-1, 7-8-9-10-11-12-7 和 13-14-15-16-17-18-13。



(a) 有一个洞的圆柱面 (b) 圆柱面的参数域 (c) 圆柱面的边界离散化结果

图 6.13. 有一个洞的圆柱面及其参数域的离散化结果

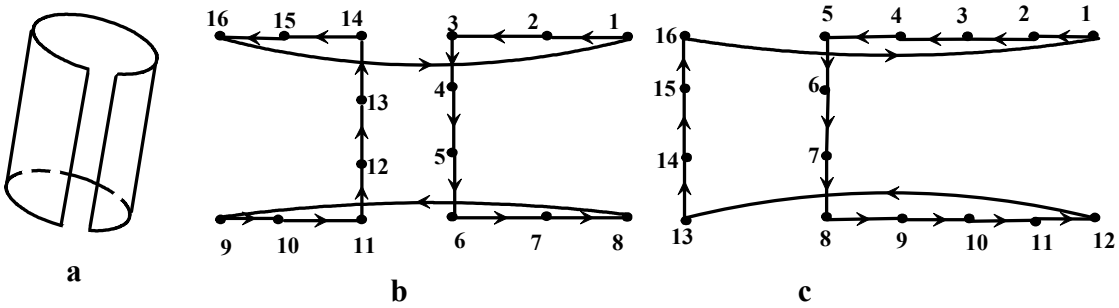
(a) Cylinder with a hole (b) The parametric space of cylinder (c) Discretization result

Fig.6.13. Cylinder with a hole & its parametric space

图 6.14 是有一个槽的单周期曲面及其参数域的离散化结果。根据槽的位置不同，其离散化结果稍有差别，图 6.14b 和图 6.14c 是两种典型的结果，两种离散化结果形成的节点序列都为：1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-1。

另外一种需要特别指出的单周期性曲面中存在退化特征曲面(如圆锥和圆球类曲面)。以圆锥为例，在离散化的时候圆锥的顶点必须考虑在内。本文中这样退化的边形成退化的前沿，在 AFT 剖分时规定，网格必须通过这样的点。图 6.15 是圆锥面及其离

散化结果，离散化后形成的节点序列为: 1-2-3-4-5-1, 6-6. 图 6.16 是带孔洞圆球面及其离散化结果，离散化后形成的节点序列为: 1-2-3-4-5-6-1, 7-7, 8-8.



(a) 有一个槽的单周期曲面 (b) 离散化结果 1 1 (c) 离散化结果 2

图 6. 14. 有一个槽的单周期曲面及其参数域的离散化结果

(a) Single closed surface with a slot (b) Discretization result 1 (c) Discretization result 2

Fig.6.14. Single closed surface with a slot and its parametric space

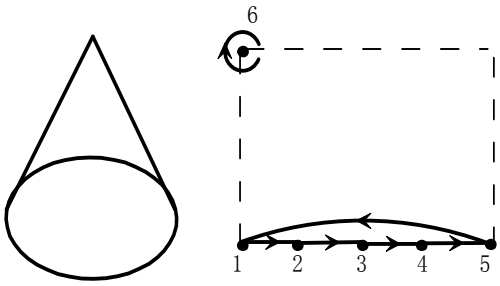


图 6.15 圆锥面及其离散化结果

Fig.6.15. Cone surface and its discretization result

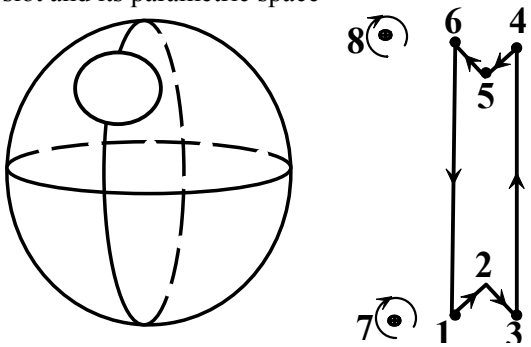


图 6.16 带孔洞圆球面及其离散化结果

Fig.6.16. Sphere surface with a hole and its discretization result

对于双向周期性曲面，其每一个方向的离散化过程和单向周期性曲面类似，也就是分别在U 向和V 向运用单向离散化算法。最常见的双向周期性曲面是圆环，对于这样的曲面，假如上面有一个孔，那么它的离散化结果为：1-2-3-4-5-6-1.

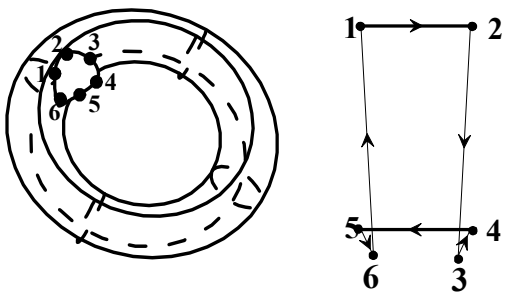


图 6.17 带有孔的圆环及其离散化结果

Fig.7.17. Torus with a hole around the boundary and its discretization result

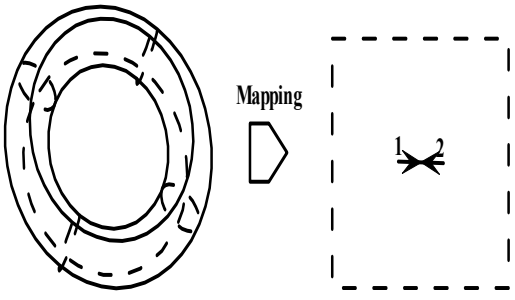


图 6.18 圆环面及其离散化结果

Fig.6.18. Torus and its discretization result

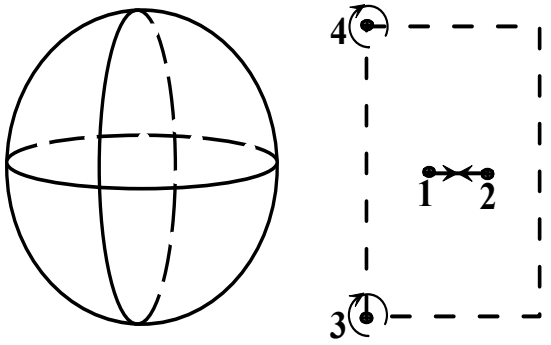


图 6.19 圆球面及其离散化结果
Fig. 6.19 Sphere and its discretization result

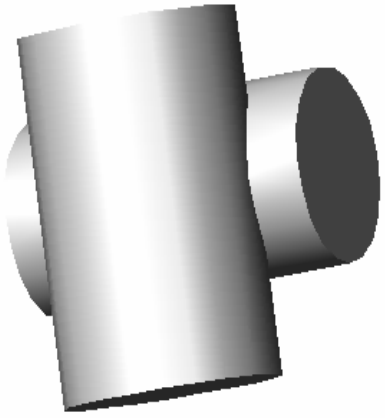


图 6.20(a) 含有周期性曲面的几何模型
Fig.20 (a). CAD model containing closed surfaces

(2) 无边界的曲线的周期性曲面的边界离散化

有些参数曲面并没有边界曲线，如圆环面和圆球面，对这样的曲面，如果其上没有孔洞，在参数域上也就无法形成初始前沿，也就不能运用二维 AFT 剖分程序形成参数域的网格。为了使 AFT 算法能够正常进行，本文对于这样的曲面，需要在曲面上添加两个前沿。图 6.18 为圆环面及其离散化结果，可以看到，在圆环面参数域的中部，添加了前沿 1-2 和 2-1，在圆环面的物理域上点 1 和点 2 之间距离等于用户在此处设置的剖分尺寸。对于圆球面，其不但没有边界曲线，而且存在两个极点，因此除了上述的处理方式以外，还要把极点考虑进去。图中的虚线框表示该曲面的参数域的范围，这样该曲面的离散化结果为：1-2、2-1、3-3、4-4。

需要指出的是，上述离散化结果并不能用于标准的二维 AFT 算法，要使二维 AFT 算法能生成周期性曲面的参数域的网格，必需扩展该算法，也就是在算法中加入点和前沿线段的迁移算子。

6.6.5 曲面参数域网格生成

当组合模型中所有的曲线的离散化完成以后，就可以对每一个曲面进行剖分，然后合成这些曲面网格，形成整个剖分域的组合曲面网格。对任何一个面的具体剖分过程如下：

(1) 取出当前面的所有曲线的离散化结果

(2) 生成参数域的三角网格，这个过程的详细算法可以参看第 4 章。如果当前曲面是一张周期性曲面，在参数域生成网格时，当前沿中线段的两个端点在周期性方向的上下边界附近时，也就是说在周期性方向上，两个端点的距离差大于 1/2 周期时，需要调用上述点的迁移算子和线段的迁移算子。

(3) 把参数域的网格投影到曲面的物理域上，更新到剖分结果中去。

6.7 数值算例与分析

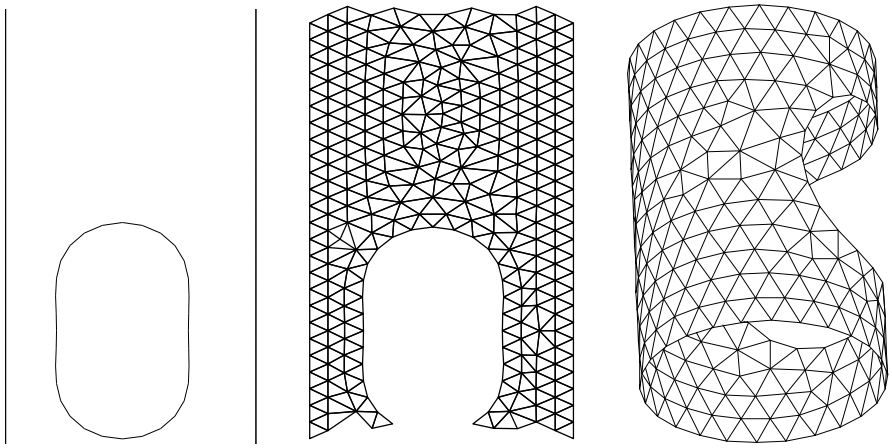


图 6.20b 曲面的参数域及其剖分结果以及物理域的投影结果

Fig.6.20 (b). One closed surface in the parametric space

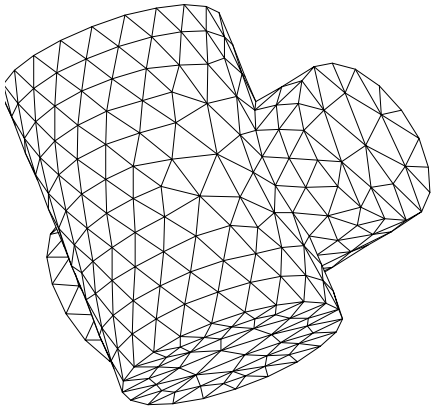


图 6.20c 整个模型的曲面剖分结果

Fig.6.20c. Surface meshing by shifting-AFT

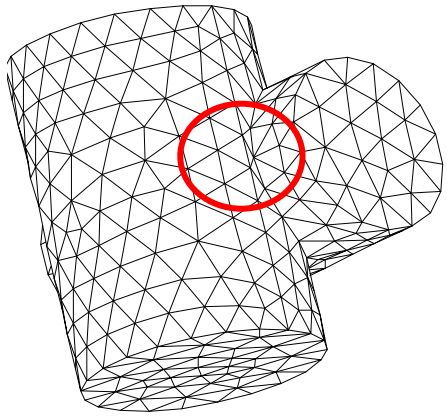


图 6.20d Ansys 软计的剖分结果

Fig.6.20d. Surface meshing by Ansys

表 6.1 周期性曲面剖分结果的质量系数对比

Tab.6.1 Distortion metrics of surface mesh

Mesher	elements	(0-0.3)	(0.3-0.6)	(0.6-0.9)	≥0.9	Min α	Avg. α
shifting-AFT	798	0	0	53	745	0.818321	0.968314
Ansys	926	0	16	75	835	0.405984	0.955256

第一个数值算例如图 6.20a 所示，这是一个含有两张圆柱曲面的几何模型。6.20b 是其中一张圆柱面的参数域以及剖分结果，最右边的图是投影到该曲面物理域后的结果。6.20c 是整个模型采用本文算法的剖分结果；图 6.20d 是采用 Ansys 软件的剖分结果；表 6.1 是本文结果和 Ansys 剖分结果的质量系数的对比。可以看出本文的结果无论在周期性曲面的内部还是在边界，网格都和理想的正三角形相接近。

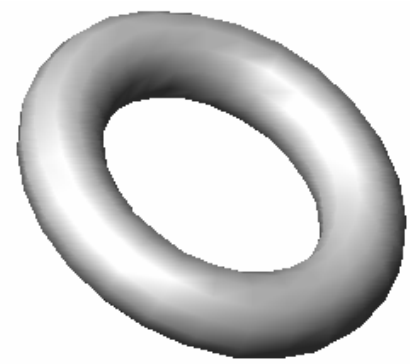


图 6.21a 圆环模型

Fig.6.21 (a) 3D solid model of torus

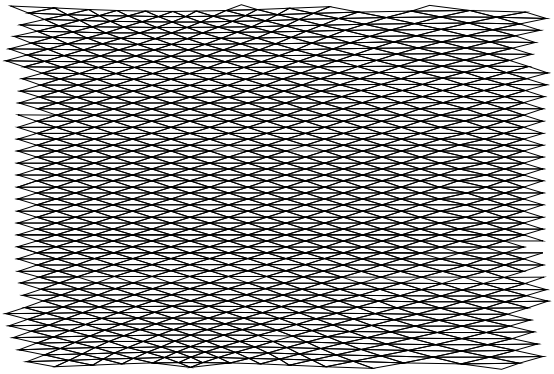


图 6.21b 圆环面参数域的剖分结果

Fig.6.21 (b) Meshing result of tours in the parametric space

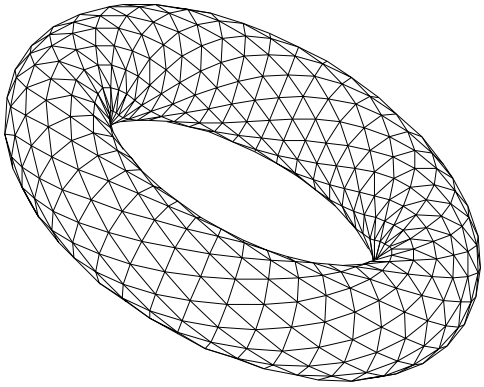


图 6.21c 本算法圆环面的剖分结果

Fig.6.21(c). Mesh of torus by shifting-AFT

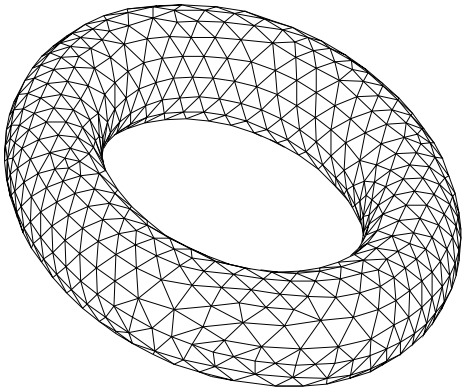


图 6.21d Ansys 圆环面的剖分结果

Fig.6.21(d). Mesh of torus by Ansys

表 6.2 圆环面剖分结果的质量系数

Tab.6.2 Torus statistics

Mesher	elements	(0-0.3)	(0.3-0.6)	(0.6-0.9)	≥0.9	Min α	Avg α
shifting-AFT	1260	0	0	294	966	0.682583	0.944629
Ansys	1704	0	2	133	1569	0.570516	0.96903

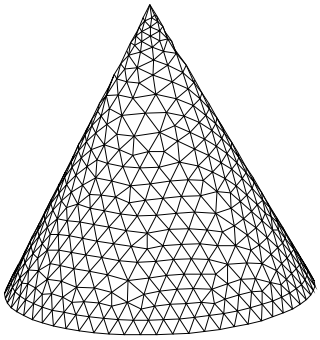


图 6.22 圆锥面的剖分结果

Fig.22 Meshing a cone

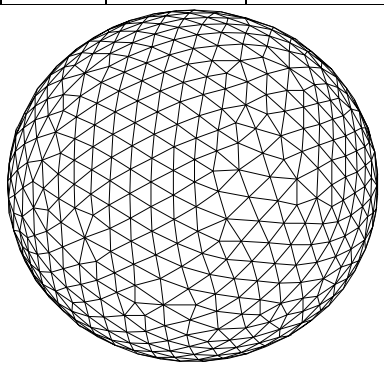


图 6.23 圆面的剖分结果

Fig.23. Meshing a sphere

第二个算例是一个双向周期性曲面的圆环的剖分。图 6.21b 是圆环面参数域的剖分剖分结果，6.21c 是本算法的对圆环面的剖分结果，6.21d 是 Ansys 软件的剖分结果。表 6.2 是本算法和 Ansys 软件剖分结果的质量系数对比情况。由图和表的对比结果可以看出，对于该算例，本算法无论单元的最小质量系数还是平均质量系数都比 Ansys 要高一些。图 6.22 和图 6.23 分别是本算法对圆锥曲面和圆面的剖分结果，这两个算例的特点是曲面不但是是周期性的，而且曲面中还存在极点(曲面在该点的法线方向不确定)。

本文的算法既可以生成曲面三维域上各向同性的规则网格，也可以生成各向异性的网格。这里首先考虑生成各向同性生成均匀网格的情况。假如用户指定的剖分尺寸是 d（d>0）,此时用户指定的黎曼度量就在空间中每点处处相等，为：

$$M_{3D}=\begin{bmatrix}\lambda & 0 & 0\\ 0 & \lambda & 0\\ 0 & 0 & \lambda\end{bmatrix}\quad \lambda=\frac{1}{d^2} \tag{6.15}$$

下面给出不考虑和考虑曲面黎曼度量情况下生成曲面网格结果的比较。图 6.24(a)是一张高纵横比的各向异性参数曲面，不考虑曲面黎曼度量在参数域上直接生成网格投影到曲面物理域的最终网格。 作为对比，用同样剖分尺寸对该曲面进行剖分，图 6.24(b)、图 6.24(c)是分别采用两个商用有限元前后处理软件的剖分结果。图 6.24(d)是本文算法在考虑黎曼度量的情况下的最终剖分网格，图 6.24(e)是其在参数域上生成的网格。 表格 1 是上述四种方法生成网格的质量系数分布情况。由表格可以看出，考虑黎曼度量后，本文算法在同样的输入条件下（曲面一样，剖分尺寸一样），产生的网格数目少，单元的最小质量系数和平均质量系数都比其它 3 种方法高。

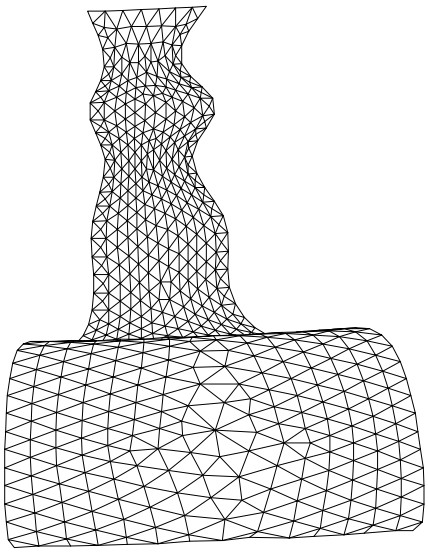


图 6.24(a)直接在参数域生成的曲面网格

Fig.6.24(a) Surface mesh without metric tensor

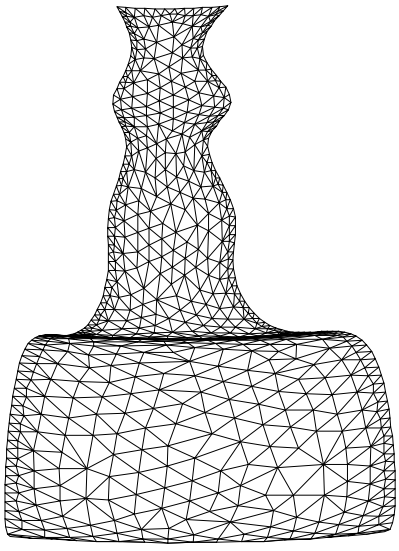


图 6.24(b) 有限元软件 1 的剖分结果

Fig.6.24(b) Surface mesh using FEM software 1

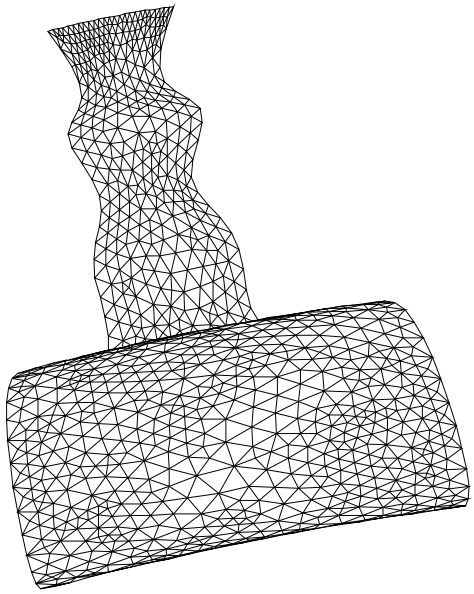


图 6.24(c) 有限元软件 2 的剖分结果

Fig.6.24(c) Surface mesh using FEM software 2

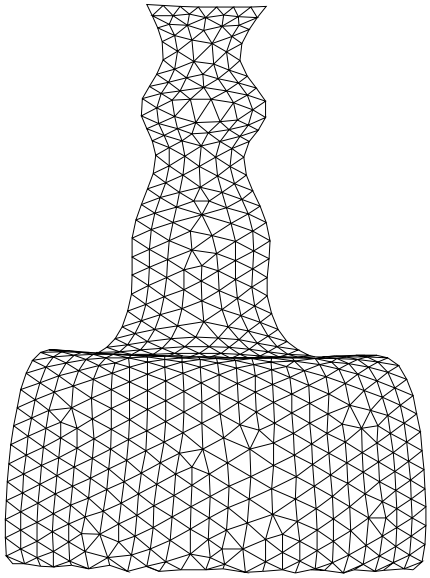


图 6.24(d) 考虑黎曼度量生成的曲面网格

Fig.6.24(d) Surface mesh considering metric tensor

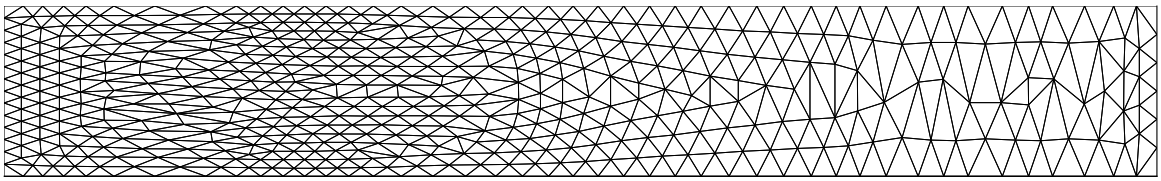


图 6.24(e) 曲面参数域上的网格

Fig 6.24(e) Mesh on Parametric space considering metric tensor

表 6.3 复杂曲面不同剖分方法生成网格质量系数分布情况

Tab. 6.3 Statistics of triangle quality of Fig 11

单元质量系数	0.0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0	最小	最大	平均
不考虑黎曼度量	89	70	79	242	966	0.041	0.999	0.784
有限元软件 1	0	28	94	474	1060	0.204	0.999	0.818
有限元软件 2	3	33	196	646	1896	0.129	0.999	0.839
本文算法	0	0	0	0	1372	0.840	0.999	0.986

在需要生成自适应的曲面网格时，三维域的黎曼度量可以由用户指定，也可以通过计算得到。下面这个算例是用户按一个公式指定三维空间每点的黎曼度量。在这里用户指定的黎曼度量和空间中两条线段(AB,CD)和两个等半径的球面(E,F)的距离有关。这里假设黎曼度量的主特征方向和坐标轴平行，即：

$e_1=[1,0,0], e_2=[0,1,0], e_3=[0,0,1]$

特征值的计算公式为

$$d_1 = \min(d_{AB}, d_{CD})$$
$$d_2 = \min(|d_E - r|, |d_F - r|)$$
$$h_1 = h_2 = h_3 = \min(2 + 0.3d_1, 1.5 + 0.2d_2)$$

这里 d_{AB} 表示空间某点到线段 AB 的距离，d_{CD} 是这点到线段 CD 的距离。d_E d_F 分别为这点到两个球心的距离，r 为两个等半径球的半径。

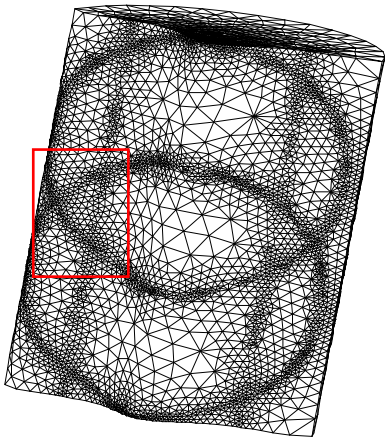


图 6.25(a) 三维组合曲面的自适应剖分

Fig.6.25(a). Example of 3D Adaptive mesh

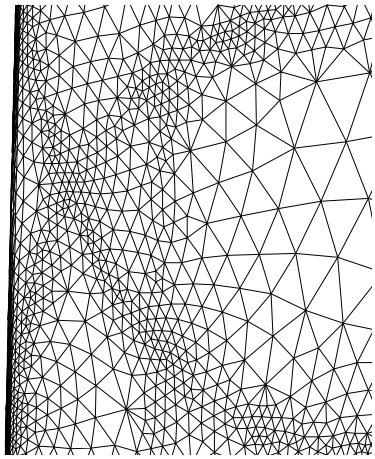


图 6.25(b) 自适应剖分的局部放大图

Fig.6.25(b). Zoom in of Figure.13a

在三维空间中计算或者设置黎曼度量本身也是一件复杂的事情，因此以下算例都假设三维空间中每点的黎曼度量都是考虑曲面本身的黎曼度量。

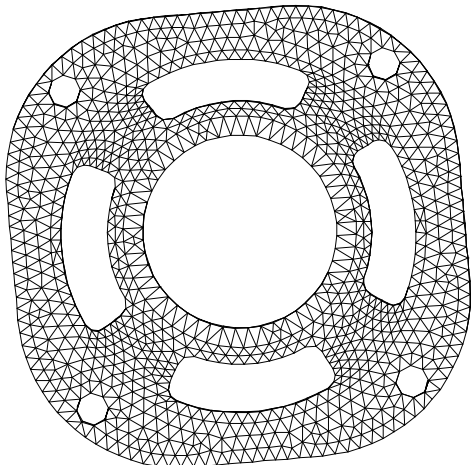


图 6.26（a） 组合曲面的剖分

Fig 6.26(a) Example of 3D combined surfaces

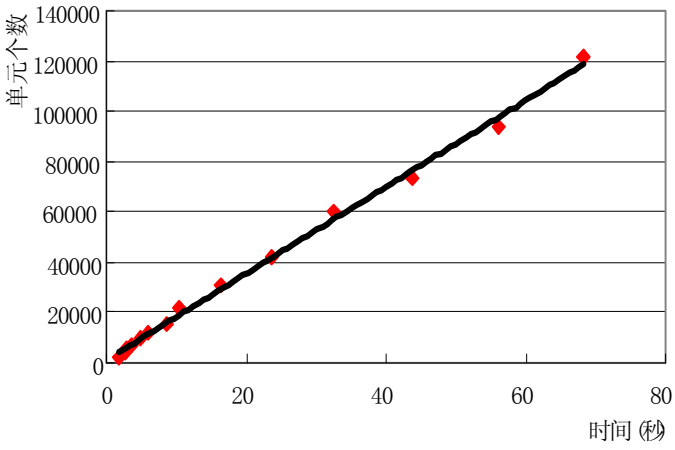


图 6.26（b） 组合曲面的剖分单元个数和时间关系曲线

Fig.6.26(b) CPU Time of 3D combined surface

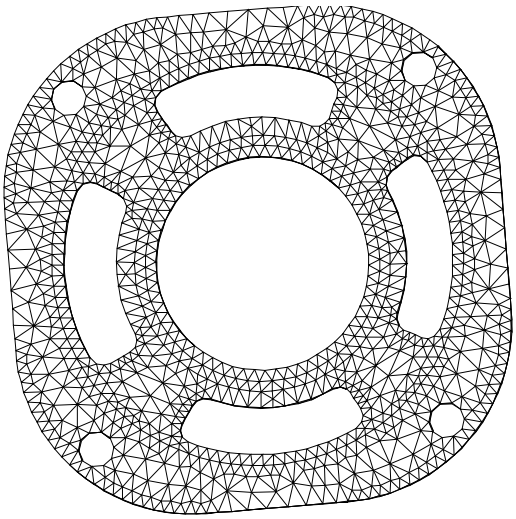


图 6.26(c) 组合曲面采用有限元软件 1 的剖分结果

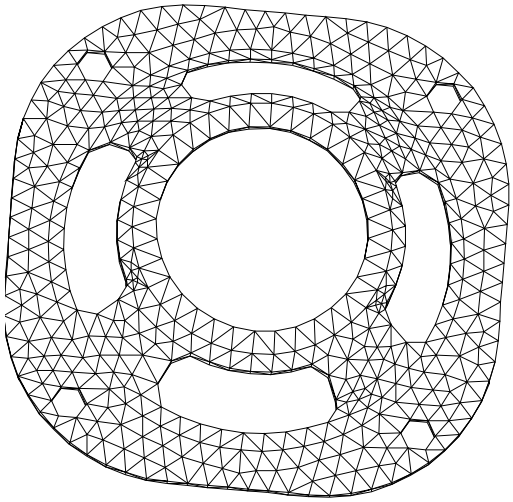


图 6.26(d) 组合曲面采用有限元软件 2 的剖分结果

Fig6.26(c) Surfaces mesh using FEM software 1　Fig.6.26(d) Surfaces mesh using FEM software 2

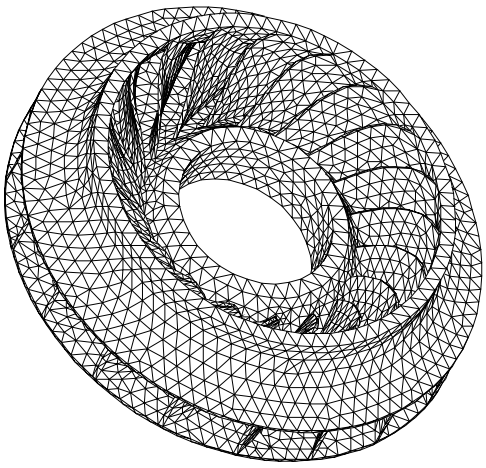
图 6.26(a)一个扬声器壳的组合曲面的剖分，图 6.26(b)是该组合曲面单元生成数目与生成时间的关系曲线，生成单元的整个过程包括曲面边界曲线的离散化、三角形单元生成、单元的优化等一系列过程。该测试在主频 1.86GHz，内存 512Mb 的 PC 机上完成。曲线表明本算法的时间复杂度基本是线性的，在该 PC 机上 1 分钟平均生成并优化 100,000 个三角形单元。图 6.26(c)、图 6.26(d)是该组合曲面采用前述两种有限元软件剖分的结果。表格 6.3 是计算得到的上述三种方法对该组合曲面生成网格的质量系数分布情况。

表 6.4 组合曲面不同剖分方法生成网格质量系数分布情况

Tab. 6.4 Statistics of triangle quality of Combined Surfaces

单元质量系数	0.0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0	最小	最大	平均
本文算法	0	0	2	501	4047	0.537	0.999	0.932
有限元软件 1	0	0	46	1041	3741	0.5075	0.999	0.895
有限元软件 2	0	18	297	209	1824	0.316	0.999	0.867

图 6.27(a)、图 6.27(b)是某厂家生产的透平机械叶轮采用本算法的剖分结果及剖分质量统计。该工程算例是一个典型的含有多张 NURBS 曲面和其它类型曲面组合而成的复杂组合曲面。



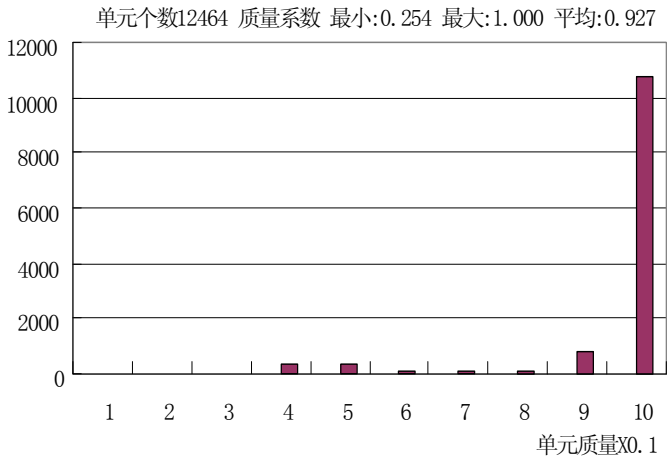


图 6.27(a) 透平机叶轮的剖分结果　　图 6.27(b)透平机叶轮的剖分结果的网格质量系数分布

Fig.6.27(a) Example of 3D combined surfaces　　Fig6.27(b) Statistics of triangle quality of Figure 27(a)

下面几个算例主要考察了曲面的曲率及近亲自适应剖分，图 6.14 时一个摩托引擎的自适应剖分结果。这个算例主要有平面和二次曲面构成，由结果可以看出在模型中倒角附近的网格明显比其它地方要密。图 15 和图 16 的两个模型是由自由曲面构成，这两个模型主要是曲率自适应，通过局部放大图可以看出，本算法的网格尺寸是梯度均匀变化的。以上几个算例和大量的工程应用表明，本文所提出方法是有效的，生成的网格质量较高。

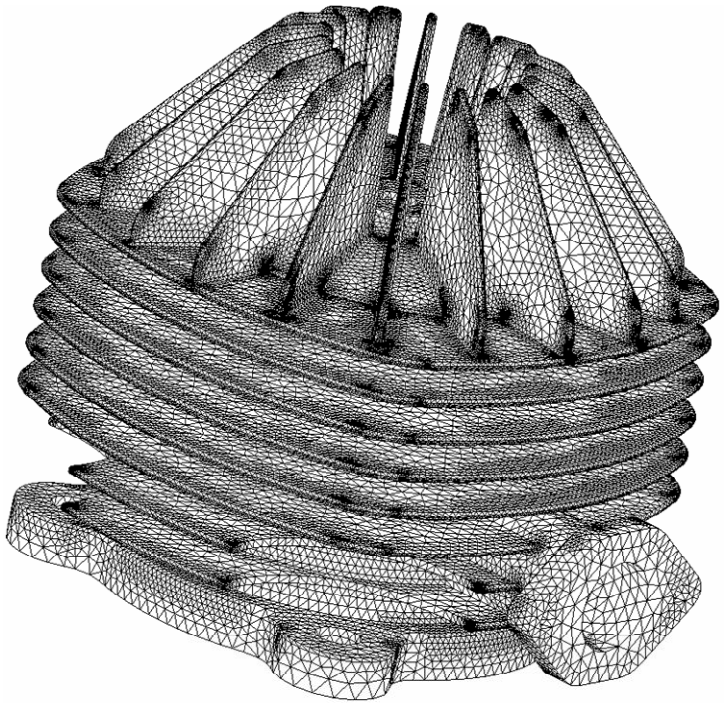


图 6.14 摩托引擎的自适应剖分结果

Fig.6.14 Adaptive mesh of engine

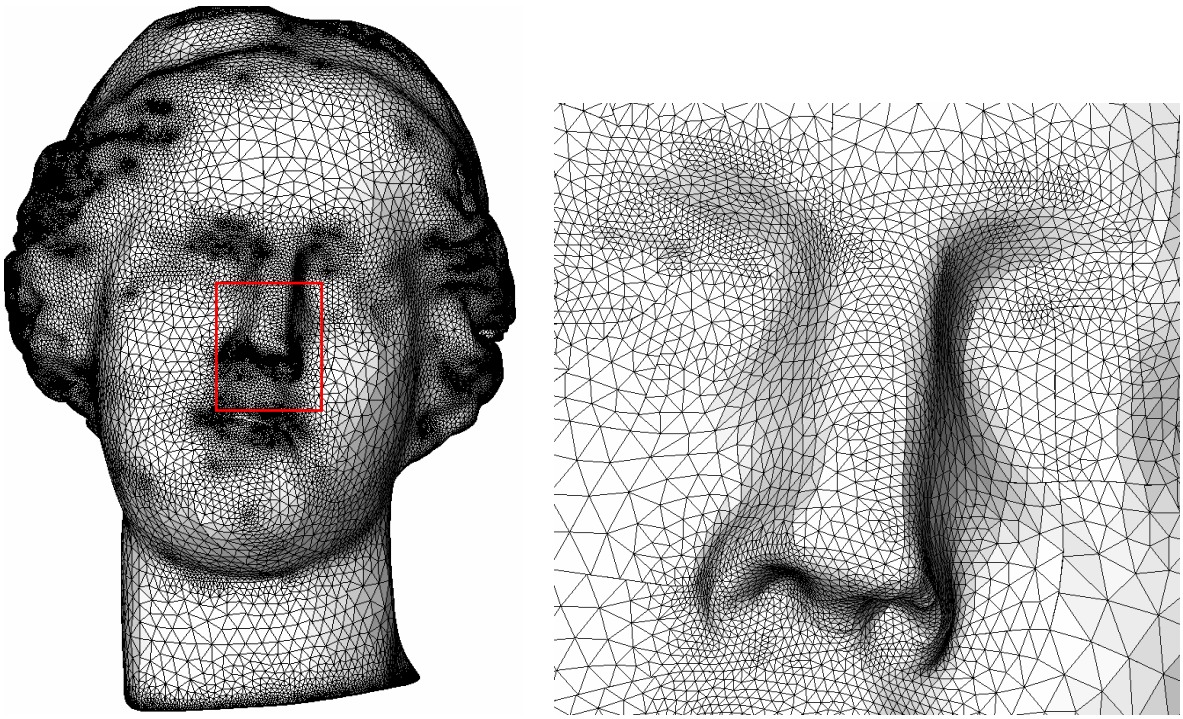


图 6.15 石膏头像及其局部放大的自适应剖分结果

Fig.6.15 Adaptive mesh of head

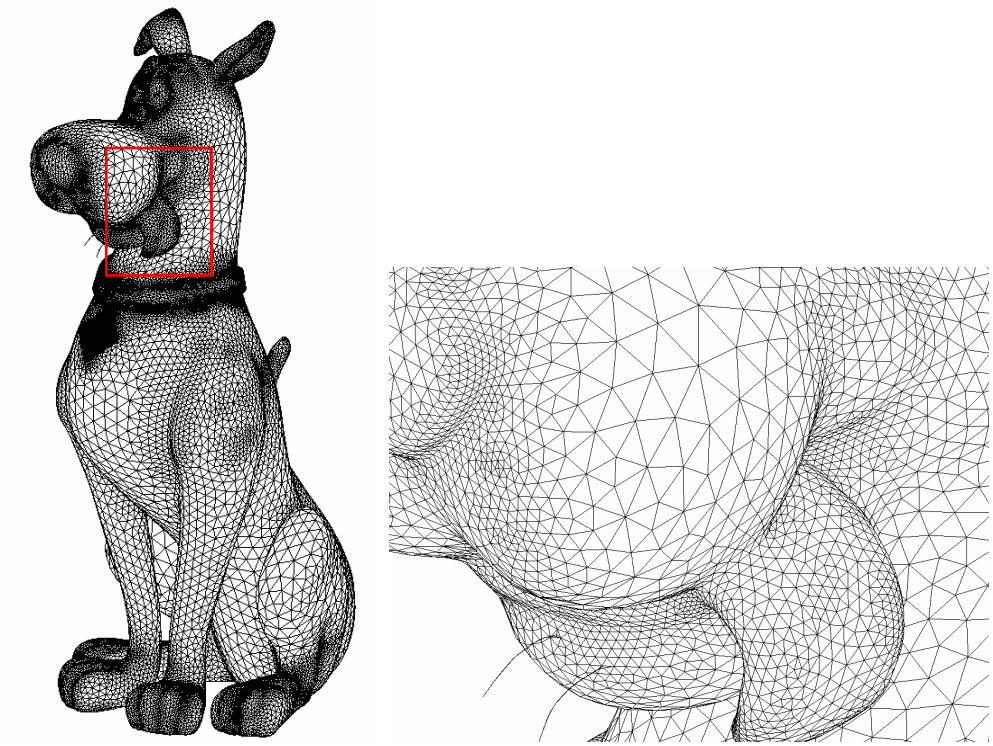


图 6.16 玩具狗及其局部放大的自适应剖分结果

Fig.6.16 Adaptive mesh of a dog doll

7. 组合曲面网格生成在金属冲压成形模拟中的应用

7.1 引言

在金属冲压成型模拟的研究中，有限元网格生成是较为关键的一步。本章主要介绍在金属冲压成型研究中附加面的构建及其有限元网格的生成，亦即通过一个实际工程算例来综合应用前面章节说叙述的内容，主要包括：附加面的生成，B-Rep 模型的构建以及由组合冲压件和附加面构成的组合曲面的有限元网格生成。由于几何造型技术并不是本文所要研究的主要内容，在这里本文主要使用开源的 Open CASCADE®所提供的接口函数(API)来生成 Nurbs 曲线和曲面。本章工作是与法国力学、材料与结构实验室(GMMS)合作研究的一部分，有关金属冲压成型方面文献可以参看郭英乔等人文章[127-129]。

一般来说工件通常是由多张 Nurbs 曲面组成的组合曲面，每张曲面的边界通常由 3 到 5 条 Nurbs 曲线构成。如图 7.1 所示，附加面一般由三个部分构成：延展面 S_1、过渡面 S_2、夹持面 S_3。为了生成附加面，需要经过以下几个步骤：

(1) 提取工件和附加面接触的边界曲线和曲面；

(2) 根据设计参数生成附加面的轮廓线(如图 7.1 的 PSTU)；

(3) 根据一系列的轮廓线生成附加面。

在生成附加面的过程中要保证附加面和工件曲面以及附加面内各曲面的一阶连续。生成附加面后，为了生成附加面及工件的有限元网格，就要构建这些曲面的 B-Rep，然后才能应用本文第 6 章的组合曲面剖分程序生成有限元网格。

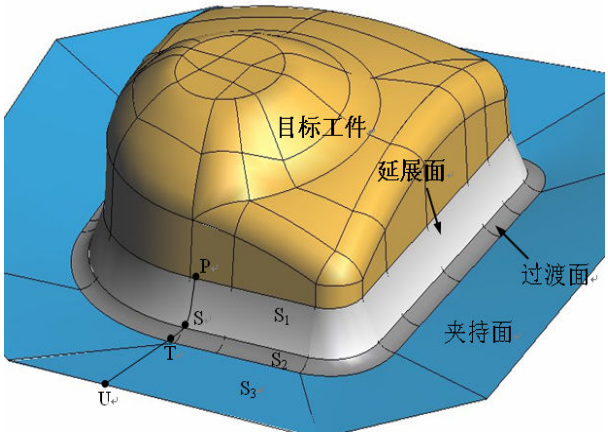


图 7.1 工件及附加面

Fig.7.1. Desired work piece and addendum surfaces

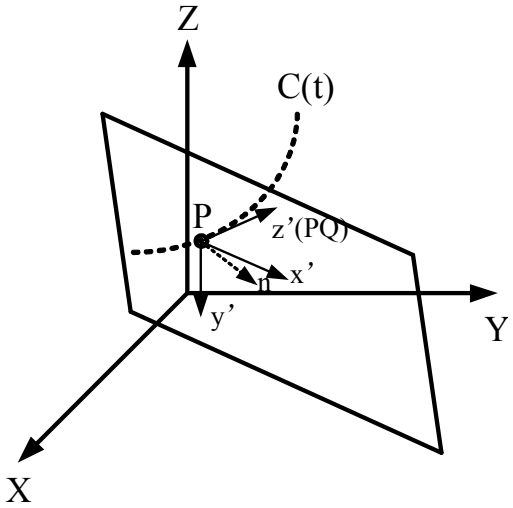


图 7.2 局部坐标系

Fig.7.2 The local reference frame

7.2 生成轮廓曲线

考察图 7.1 的算例，为了生成附加面，首先需要在工件和附加面的接触边界曲线上取得一些特征点，对每一个特征点 P 生成附加面的轮廓线 PSTU。本文首先在一个局部坐标系下生成二维的轮廓线，然后把这个二维的轮廓线转换到工件所在的坐标系下，形成三维空间的轮廓线。这个局部坐标系是这样定义的（如图 7.2）：

(1) P 点所在的工件曲面是 $S(u,v)$， $S(u,v)$ 上 P 点所在的曲线是 $C(t)$；

(2) 计算 $C(t)$ 在 P 的切向量，设为 $\overrightarrow{PQ}$

(3) 局部坐标系 $CC(x',y'z')$ 和全局坐标系 $WC(X,Y,Z)$ 的关系为：

$$\mathbf{z'}=\overrightarrow{PQ},\quad \mathbf{x'}=\mathbf{z'}\times\mathbf{Z},\quad \mathbf{y'}=\mathbf{z'}\times\mathbf{x'} \tag{7.1}$$

符号×表示两个向量叉乘。在这个应用中轮廓线(如图 7.3 所示)由两条线段(OQ,RS)，两段圆弧(QR, ST)以及一段曲线 TU 构成。

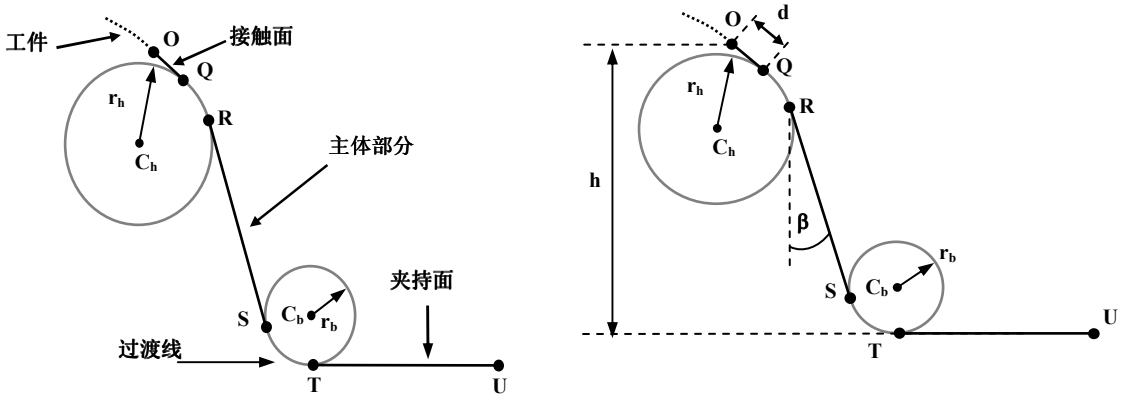


图 7.3 轮廓线及设计变量

Fig.7.3. Profile curve PQRSTU and design variables for surface generation

空间坐标系的点 $P(x_p,y_p,z_p)$ 是局部坐标系 $x'o'y'$ 下的原点，也就是 $x'o'y'$ 平面中点 P 在原点 $O(0,0)$。为了保证延展面的一阶连续，图 7.3 中向量 $\overrightarrow{O\mathbf{Q}}$ 垂直于点 P 在所在曲面 $S(u,v)$ 的外法线在 $x'o'y'$ 投影的垂直方向上。也就是说，假设曲面在点 P 的外法线为 $\vec{\mathbf{n}}$， $\vec{\mathbf{n}}$ 在 $x'o'y'$ 投影为 $\vec{\mathbf{n}}'$，那么 $\overrightarrow{O\mathbf{Q}}\perp\vec{\mathbf{n}}'$。在计算 $\vec{\mathbf{n}}'$ 时需要构造一个从全局坐标系 XOY 到局部坐标系 $x'o'y'$ 的坐标转化阵，坐标转化阵是一个 4×4 方阵，为：

$$M_{WC}=\begin{bmatrix}\vec{x}'_x & \vec{y}'_x & \vec{z}'_x & -o'.\vec{x}'\\ \vec{x}'_y & \vec{y}'_y & \vec{z}'_y & -o'.\vec{y}'\\ \vec{x}'_z & \vec{y}'_z & \vec{z}'_z & -o'.\vec{z}'\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{7.2}$$

上述$\vec{x}',\vec{y}',\vec{z}',o'$分别表示根据 7.1 计算得到的局部坐标系三个坐标轴的单位向量及局部坐标系的原点，$\vec{x}'_x$表示取该向量的 x 轴坐标，其它类似。这样全局坐标系WC中的点P转化到局部坐标系CC中点P'为：

$$P' = M_{WC}.P \tag{7.3}$$

$\overrightarrow{O\mathbf{Q}}$可以这样计算：

(1) 计算P的外法线方向$\vec{\mathbf{n}} = \dfrac{S(u,v)}{\partial u} \times \dfrac{S(u,v)}{\partial v}\Big|_{(u,v)=P(u,v)}$

(2). 全局坐标系WC点P沿着$\vec{\mathbf{n}}$平移距离 1 个单位，得到点$Q = P + \vec{\mathbf{n}}$

(3) 按照公式 7.3 计算点P和Q在局部坐标系CC的投影点O, Q',那么$\vec{\mathbf{n}}' = \overrightarrow{OQ'} / \left\| \overrightarrow{OQ'} \right\|$

(4) 局部坐标系CC， $x'o'y'$平面内$\overrightarrow{O\mathbf{Q}} = \left[-\vec{n}'_2 \quad \vec{n}'_1 \right]^T$

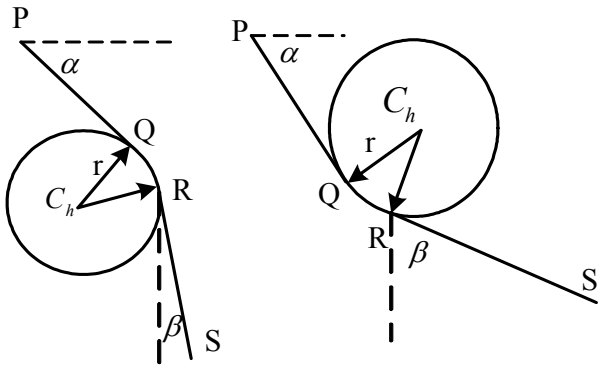


图 7.4 第一段圆弧的两种情况

Fig.7.4.Two case of the center of first circle

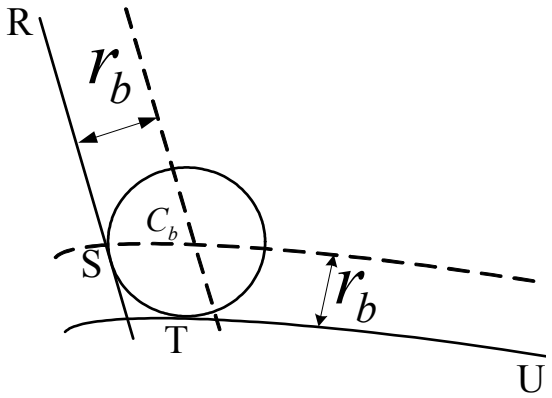


图 7.5 第二段圆弧的两种情况

Fig.7.5 Calculation of the second arc

这样线段 OQ 和x'轴的夹为：$\alpha = \arccos(\overrightarrow{O\mathbf{Q}}.\vec{x}' / \left\| \overrightarrow{O\mathbf{Q}}.\vec{x}' \right\|)$，轮廓线的其它设计变量由用户输入，图 7.3 所示。这样已知设计变量d, β, h, r_h, r_b点 Q，R 以及第一段圆弧的圆心可以用简单的向量代数的方法求得。根据输入角β和α的相对关系，有两种情况需要考虑（如图 7.4），具体可以用如下公式求得：

(1) Q 的坐标：$Q(x,y) = (d\cos(\alpha), d\sin(\alpha))$

(2) 第一个圆弧中心C_h的坐标：

$$C_h(x,y) = \begin{cases} \text{infinite} & \beta = \pi/2 - \alpha \\ C_h(Q(x) - r_h\sin(\alpha), Q(y) + r_h\cos(\alpha)) & \beta < \pi/2 - \alpha \\ C_h(Q(x) + r_h\sin(\alpha), Q(y) - r_h\cos(\alpha)) & \beta > \pi/2 - \alpha \end{cases} \tag{7.4}$$

(3) 点 R 的坐标

$$R(x,y)=\begin{cases} \mathrm{Q} & \beta=\pi/2-\alpha \\ R(C_h(x)+r_h\cos(\beta),C_h(y)-r_h\sin(\beta)) & \beta<\pi/2-\alpha \\ R(C_h(x)-r_h\cos(\alpha),C_h(y)+r_h\sin(\alpha)) & \beta>\pi/2-\alpha \end{cases} \tag{7.5}$$

点 S， T 以及第二个圆弧的中心 C_b 的计算稍有些复杂，可以这样计算(如图 7.5)：

(1) 求得 $x'o'y'$ 平面和夹持面 S_3 的交线 $C(t)$

(2) 将 $C(t)$ 投影到平面 $x'o'y'$，得到曲线 $C_1(t)$

(3) $C_1(t)$ 偏移 r_b 得到曲线 $C_2(t)$

(4) 射线 RS 偏移 r_b 得到曲线 RS_1

(5) RS_1 和 $C_2(t)$ 的交点就是 C_b

(6) C_b 分别向 RS 和 $C_1(t)$ 投影求得点 S， T。

以上所叙述的步骤可以通过 Open CASCADE®的 API 来完成，这样就生成 $x'o'y'$ 平面上的轮廓线，要得到三维工件坐标系下的轮廓线，还需要进行一次又局部坐标到全局坐标的一个坐标变换，变换阵为：

$$M_{CW}=\begin{bmatrix} \vec{x}'_x & \vec{y}'_x & \vec{z}'_x & o'_x \\ \vec{x}'_y & \vec{y}'_y & \vec{z}'_y & o'_x \\ \vec{x}'_z & \vec{y}'_z & \vec{z}'_z & o'_x \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.6}$$

7.3 生成附加面

每个附加面的边界曲线构造完成以后，可以通过三维边界曲线构造孔斯曲面作为附加面[125, 130]。在这里本文直接使用 Open CASCADE®中 GeomFill_BSplineCurves 类来构造孔斯曲面，然后使用本文第 2 章介绍的 SlGeSurfaceCasCade 类构造一张在 Open CASCADE®下特化的 SlGeSurface 类型的指针，作为拓扑面(SlBrFace)相关联的几何面，供构造组合曲面的 B-Rep 时使用。

7.4 构建模型的 B-Rep

第 2 章已经详细介绍了模型的 B-Rep 数据结构，一般来说如果模型是 CAD 软件构建的，我们可以通过相应的接口函数(API)读取模型的几何和拓扑信息，填充第 2 章所述的 B-Rep，然后就可以根据用户的剖分参数进行网格剖分了。在这里由于模型的附加面是根据模型的轮廓线分别构造的，所以 B-Rep 的构建需要通过算法来实现。在这里工件

和夹持面的模型是已知的，假如通过标准的 Step 文件存储。对这样的 Step 文件可以通过 Open CASCADE®的 Step 文件读取模块提取工件和夹持面的模型的 B-Rep 模型。这里主要是要构造附加面的 B-Rep，构建完成后和工件模型的 B-Rep 模型合并形成整个模型的 B-Rep 模型。构造附加面 B-Rep 的过程是一个由底向上的过程，在这里需要用到 2.3 节几何模型的 B-Rep 接口部分的内容，具体方法如下：

(1) **构造拓扑点 V：** 取得所有附加面曲线的端点，由这些端点构造拓扑点 V(SlBrVertex)放到 SlBrBody 结构中；

(2) **构造拓扑边 E：** 由每条三维曲线 $C(t)$ 的两个端点对应的拓扑点构造拓扑边 E(SlBrEdge)，并设置对应的 $C(t)$ 的指针及 $C(t)$ 和 E 的相对方向(调用 setCurve 方法)；

(3) **构造拓扑环 L：** 由于这里的每个附加面和过渡面都只是由一个外环构成的，也就是说对其中的任何一个拓扑面(F)来说构成其区域的环只有一个外环。对面 F，得到所有边界曲线对应的边，这些边构成曲面的外环(L)，使环的整体方向在和面 F 对应曲面 $S(u,v)$ 的参数域上满足左手规则。在往环(L)中添加边 E 的引用时，同时需要设定该边和当前环的相对方向。

(4) **构造拓扑面 F：** 如前所述，对于延展面和过度面只有一个外环，这样只要把这个环添加到拓扑面 F 就可以了，同时设定和 F 对应的参数曲面 $S(u,v)$ 的引用，在这里环和面的相对方向都是一致的。

(5) **构造夹持面的拓扑面 F：** 由于夹持面是已知的，一般来说，它往往不刚好和过度面相结合。如图 7.6 所示，这里假设 S9 是夹持面，构造这个曲面的拓扑面时，并不需要真正求得过度面和夹持面的交线，而只是需要把过度面和夹持面相接触的曲线对应的拓扑边添加到夹持面的环中，在添加的过程中只要满足左手规则就可以了。

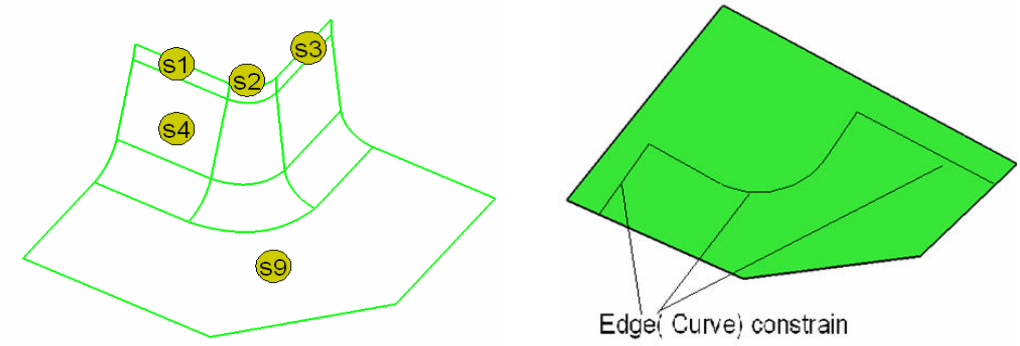


图 7.6 夹持面边界约束的添加

Fig 7.6 Adding Edge constrain to cutting surface

通过以上步骤就可以构成所有附加面的 B-Rep 模型，如果需要同时生成工件和附加面的网格，可以把工件的 B-Rep 模型和附加面的 B-Rep 合并形成一个整体。B-Rep 模型生成以后就可以根据用户设置的剖分尺寸生成有限元网格了，具体过程参照第 2 章，这里就不再赘述。

7.5 剖分结果

图 7.7 是一个冲压成形算例附加面的剖分结果，这里给出的是大小均匀规则网格。图 7.8 是把工件和附加面的 B-Rep 合成的剖分结果。图 7.9 是采用本文算法根据工件的曲率变化的自适应剖分结果，以及局部放大视图。表 7.1 是图 7.9 的算例单元质量统计，质量系数是按照第 6 章介绍的三角形曲面网格的质量系数公式计算得到，该表还统计了生成结果中的最小内角，由表中可以看出绝大多数单元的质量系数都在 0.8~1.0 之间，最小内角在 48^{o}~72^{o}，接近于正三角形。从以上结果中可以看出，无论在均匀网格还是自适应网格，本文算法都能很好地满足冲压成形的需要。

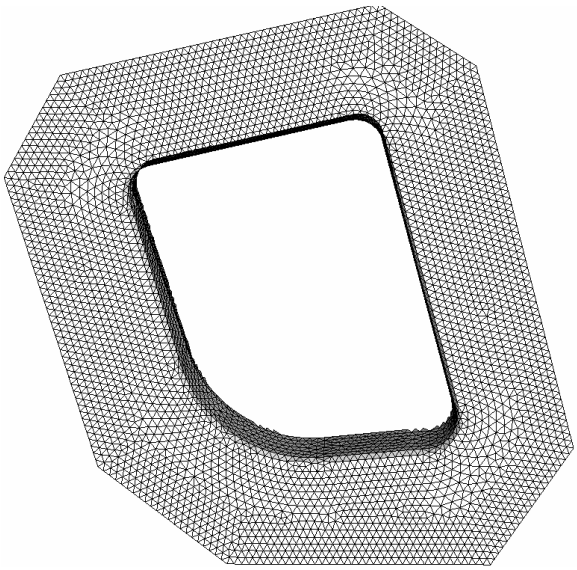


图 7.7 附加面的表面规则网格

Fig.7.7 Mesh of addendum surfaces

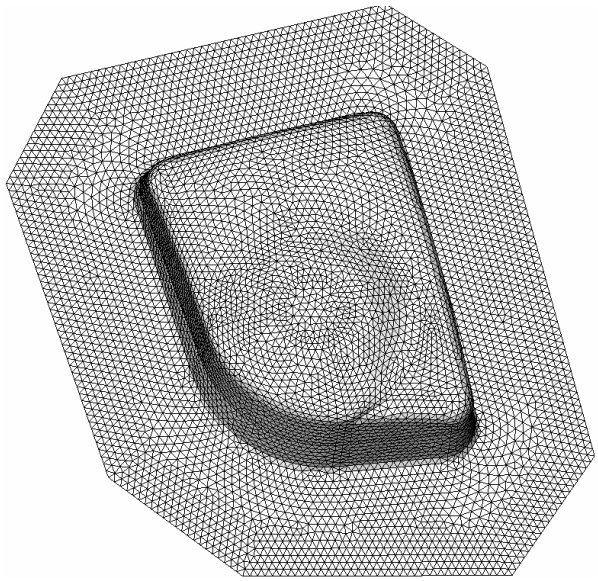


图 7.8 附加面和工件的表面规则网格

Fig. 7.8 Mesh of addendum surfaces and work piece

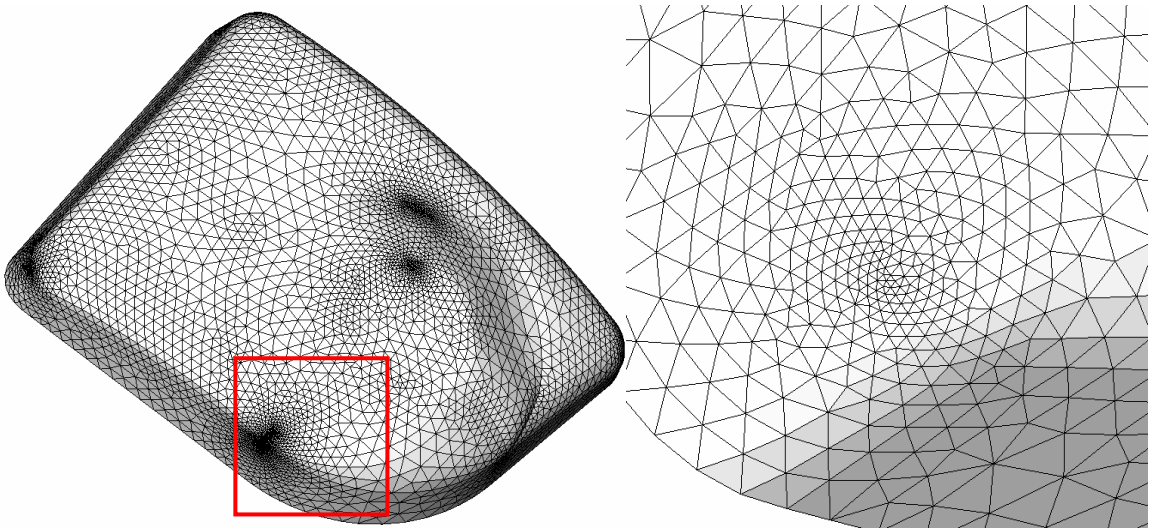


图 7.9 工件表面的自适应网格及局部放大图

Fig. 7.9 Adaptive Mesh of work piece and Zoom in

表 7.1 网格生成质量统计

Tab.7.1. The statistic of mesh quality

质量系数 $2r/R$	0.0~0.2	0.2~0.4	0.4~0.6	0.6~0.8	0.8~1.0
单元个数	0	1	1	23	8900
最小内角	$0^0 \sim 24^0$	$24^0 \sim 48^0$	$48^0 \sim 72^0$	$72^0 \sim 96^0$	$> 96^0$
单元个数	3	878	8044	0	0

8.高效可靠的三维四面体 AFT 网格生成算法

8.1 引言

由于实际应用中绝大多数问题是三维问题，因此一个全自动的健壮的快速的三维复杂几何体的网格剖分程序就显得尤为重要。在实体单元中，四面体单元由于具有许多优良特性而在实际中应用最为广泛。目前在不同的应用领域已经有很多关于四面体单元全自动生成方法，在这些方法中比较有代表性的是：Delaunay 三角化方法，改进八叉树法和波前推进方法(AFT)。Delaunay 三角化方法首先在剖分域内产生一些点，然后利用 Delaunay 空心球原理把这些点连接成四面体。Delaunay 方法速度快，产生的单元质量相对较高。其主要问题是边界恢复，因为空心球原理会破坏初始的曲面的三角剖分结果。改进八叉树法的基本思想是：通过递归地细分一个包围待剖分区域的空间来逼近该区域并生成有限元网格。这个方法的主要问题是边界网格质量较差。经过近年来的发展，推进波前法（简称 AFT）已成为目前较为流行的通用全自动网格生成方法之一。AFT 方法的基本流程是：以由三角面片构成的初始边界为前沿逐步向内推进，推进时节点与单元同时形成，直至待剖分域收缩为空。虽然 AFT 方法没有 Delaunay 三角剖分算法那样成熟的理论依据，但还是获得了非常广泛的应用。AFT 算法的时间复杂度与 Delaunay 四面体化方法和有限八叉树法相当，但生成单元的质量，特别是边界处单元质量，是三者中最好的。AFT 方法在生成节点时对节点的位置加以控制，从而控制单元形状、尺寸以达到质量控制、局部加密及网格过渡的要求。但 AFT 方法也存在一个难点问题：收敛性问题。所谓收敛性问题是指在三维网格生成中，AFT 有时会剩余一个或多个体积不为零的但又无法继续剖分的多面体。

8.2 四面体网格的生成算法

与大多数采用 AFT 方法生成实体网格的过程类似，本文中生成实体网格的过程总体上分为以下三个步骤：

(1) 取得实体各曲面的边界曲线，并按用户要求对这些边界曲线进行离散化；

(2) 生成实体的表面三角形网格；

(3) 以实体的表面网格作为输入，生成模型的实体网格。

在这里主要介绍第三个步骤:实体网格的生成。在进行实体网格生成之前，首先需要完成表面网格的初始化，使其法线方向指向实体内部，也就是满足右手定则。

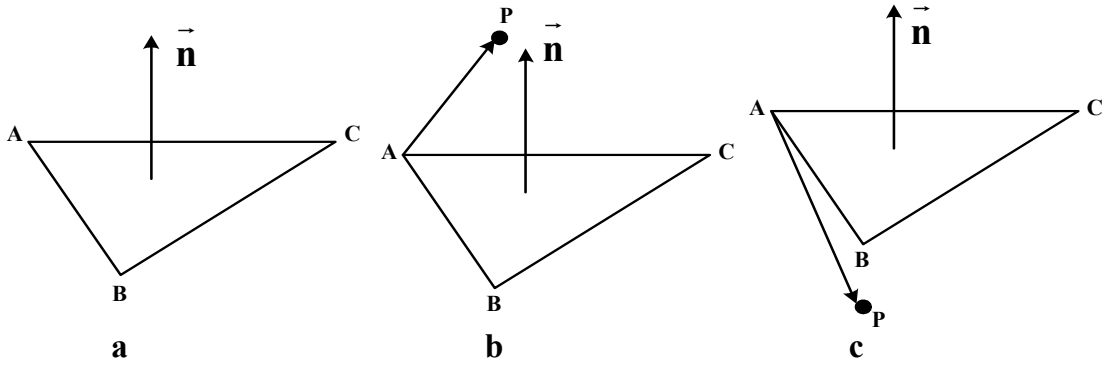


图 8.1 点与三角形之间的位置关系

Fig. 8.1. The position relationship between point and triangle

8.2.1 实体表面三角形网格初始化

用于生成四面体的网格必须满足其法线方向指向待剖分域内部。如图 8.1(a)所示，三角形 $Tri(ABC)$ 的法线方向为：$\vec{n} = \overrightarrow{AB} \times \overrightarrow{AC}$，如果 $\vec{\mathbf{n}}$ 指向待剖分域内部就可直接用来作为前沿生成四面体单元，否则调换其构成的三个节点中的任意两个。这样 $Tri(ABC)$ 把整个空间分成上下两个部分，如果点 P 在 $Tri(ABC)$ 的上方，那么 $\overrightarrow{AP} \bullet \vec{n} > 0$；如果点 P 在 $Tri(ABC)$ 的下方，那么 $\overrightarrow{AP} \bullet \vec{n} < 0$。四面体 $Tet(ABCP)$ 的体积可以按下列公式得到：

$$V_{Tet(ABCP)} = \frac{(\overrightarrow{AB} \times \overrightarrow{AC}).\overrightarrow{AP}}{6} \tag{8.1}$$

这样所有位于 $Tri(ABC)$ 上方的点和 $Tri(ABC)$ 构成的四面体体积为正，在其下方的点和 $Tri(ABC)$ 构成的四面体体积为负。

8.2.1 初始化背景网格

和二维 AFT 算法类似，为了尽可能生成局部均匀一致的四面体单元，并加快相关前沿数据的搜索速度，本文以节点为中心，在整个剖分过程中维护以下数据和关系：

(1)节点 N_i 的坐标(x_i)

(2)第n层前沿上的节点(N_i)参数: 包括参照高度 h_{oi}，搜索半径 r_{oi}，所连接的面的集合 F_{oi}。节点的参照高度可以这样计算:

$$\overline{a_i} = \left(\sum_{k=1}^{n} a_k \right) / n \tag{8.2a}$$

$$h_{oi} = 1.427\sqrt{a_i} \tag{8.2b}$$

公式中 a_k 为当前节点所连接的某个面的面积，n 为连接面的个数。搜索半径 r_{oi} 首先等于该节点所连接的最长边的边长。为了得到变化均匀的网格，参照高度和搜索半径的计算还要考虑到其周边区域对它的影响，为此定义周围节点对当前节点的影响系数 c_i，c_i 的初值为零。对于节点 N_i，首先定义一个球 $(x_i, 3r_{oi})$ (即球心在 x_i 半径为 $3r_{oi}$)，如果剖分域中有节点 N_j 在球内，那么节点 N_i 和节点 N_j 的调整参数计算如下：

$$
\begin{aligned}
&c_{oij} = 0.05 \frac{3r_{oi}}{\left|x_j - x_i\right|} \frac{h_{oj} - h_{oi}}{h_{oj}} \\
&c_{ji} = \min(\max(c_{oij}, -0.6), 0.5) \\
&c_i = c_{ji}, if \left|c_{ji}\right| > \left|c_i\right| \\
&c_j = c_{ji}, if \left|c_{ji}\right| > \left|c_j\right|
\end{aligned}
\qquad (8.2c)
$$

图 8.2 对上述讨论进行了直观显示，为清晰起见，图中以二维形式进行表述，三维情况与此类似。为了找到位于球 $S(x_i, 3r_{oi})$ 中的节点，节点搜索过程是必须的。因为在整个剖分过程中类似的搜索过程比较频繁，所以快速的搜索算法在这里显得尤为重要。最后，当节点的调整系数确定以后，最终的节点的参照高度 h'_{oi} 和搜索半径 r'_{oi} 按下面公式计算得到：

$$
\begin{aligned}
&h'_{oi} = (1.0 + c_i).h_{oi} \\
&r'_{oi} = \max(1.0 + c_i, 1.0).r_{oi}
\end{aligned}
\qquad (8.2d)
$$

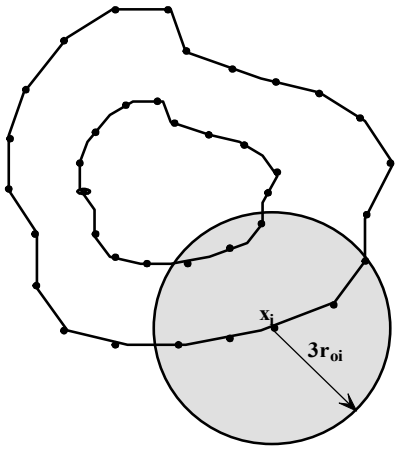


图 8.2 节点的参照高度和搜索半径的调整

Fig.8.2. Reference height and searching radius of node

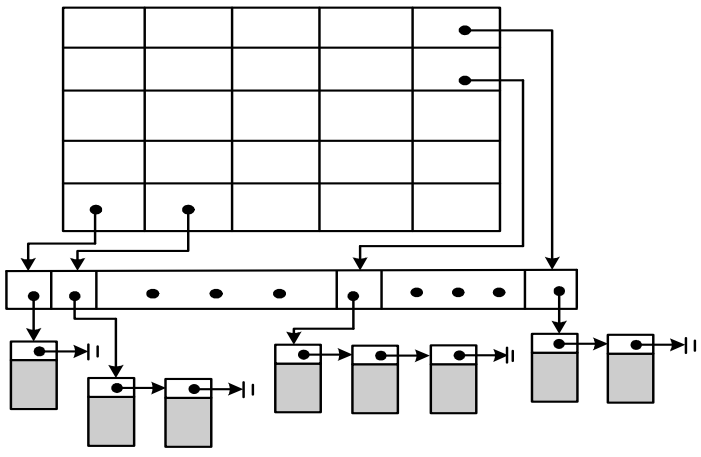


图 8.3 四面体生成的背景网格

Fig.8.3. Background for Tetrahedral mesh

当上述节点参数计算完成以后，就需要把计算结果存到背景网格中去。三维待剖分域的背景网格可以这样构造：

(1)首先取得模型的外包围盒，设其在三个坐标轴上的尺寸为：

$x \in [x_1, x_2], y \in [y_1, y_2], z \in [z_1, z_2]$

(2)计算初始前沿三角面边长的平均值，记为 s_{avg}。

(3) 细分模型的外包围盒，X,Y,Z 三个坐标轴方向的细分份数分别为 Num_x, Num_y, Num_z：

$$\begin{aligned} Num_x &= (x_2 - x_1)/s_{avg} \\ Num_y &= (y_2 - y_1)/s_{avg} \\ Num_z &= (z_2 - z_1)/s_{avg} \end{aligned} \tag{8.3a}$$

这样模型外包围盒在三个坐标轴方向上被分割成个 Num_x， Num_y， Num_z 栅格，如图 8.3 所示。这一个个小栅格作为存储上述节点及其相关信息的"桶子"。因为每个栅格可能包含一组节点，这样栅格实际存储的是包含这组节点的一个链表。这里首先把模型表面结点按公式(8.2)计算出相关信息，存储到相应的栅格中。本文的栅格在一维数组中存储顺序是 $X \to Y \to Z$，这样对于节点 N_i，设其坐标为 $x_i(x, y, z)$,那么其在一维数组中的位置 pos_{N_i} 可以这样计算：

$$pos_{N_i} = Num_x(Num_y(Num_z \frac{z - z_1}{z_2 - z_1} + \frac{y - y_1}{y_2 - y_1}) + \frac{x - x_1}{x_2 - x_1}) \tag{8.3b}$$

8.2.2 初始化前沿队列

和大多数采用 AFT 方法进行四面体剖分的算法一样，为了减小整个程序的复杂度并提高剖分的速度，本文采用按层推进的方式对模型进行四面体剖分。经验表明，对于面积较小的三角形优先推进最后生成的四面体单元的质量系数总体较高。算法的收敛性问题是在用 AFT 方法进行三维四面体剖分的一个主要问题，对于大多数三维几何模型一般会遗留多个不能按照多面体三角剖分的要求继续剖分的核。此时采用回退方法可以解决大多数三维为题，也就是删除内核附近的一层或多层四面体单元形成若干空腔，然后对这些空腔进行在剖分。然而如果回退过程中不改变前沿推进的路径，这个删除、重剖分的过程往往要经过多次，从而使整个剖分速度明显下降，有时甚至找不到一个可行的剖分结果。

因为 AFT 方法对剖分尺寸的选择非常敏感，为了减少回退的次数，提高整个算法的剖分速度，本文在进行内核回退剖分的时候，前沿三角形的优先序不再简单按照小面积优先的原则，而是循环采用多种优先策略：对于作为前沿的三角形，本文引入一个称为优先因子 γ，默认情况下 γ 等于三角形的面积。当进入第 i 次回退过程时 γ 值按如下公式计算：

$$\gamma_i=\begin{cases}\alpha & \mathrm{i}\in[0,4)\\ \alpha.\beta_{tri} & \mathrm{i}\in[4,8)\\ \beta_{tri} & \mathrm{i}\in[8,12)\\ 1.0-\beta_{tri} & \mathrm{i}\in[12,16)\\ (1.0-\beta_{tri}).\alpha & \mathrm{i}\in[16,20)\end{cases} \tag{8.4a}$$

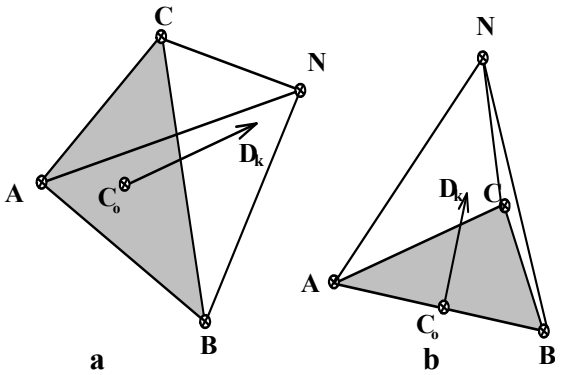
其中α是三角形的面积，$\beta_{tri}\in[0,1.0]$为三角形的质量系数，β_{tri}可以这样计算：

$$\beta_{tri}=\frac{2r_{tri}}{R_{tri}}=\frac{8(p-a)(p-b)(p-c)}{abc} \tag{8.4b}$$

其中：r_{tri}为三角形内切圆半径，R_{tri}为三角形外接圆半径，a,b,c是三角形三边的边长，$p=0.5(a+b+c)$。对正三角形$\beta_{tri}=1.0$。前沿队列Ω是一个以前沿三角形优先因子γ作为排序准则的优先队列，所有未剖分区域的边界三角形的集合构成初始前沿队列Ω_o。本文是采用按层推进的方式进行四面体的剖分，在剖分地过程中把暂时不能按照三角剖分要求进行推进的前沿单独放到前沿队列Ω_h中。这样本文的前沿队列一共有三个：当前层的前沿对队列Ω_C（初始化完成以后$\Omega_C=\Omega_o$）；下一层的前沿对队列Ω_N；以及特殊的暂时不可剖分的前沿队列Ω_H。并且这些前沿队列都是一个按前沿优先因子γ为优先序的优先队列。所以初始化前沿队列包括：

(1) 把曲面剖分程序得到的按右手规则编号的模型表面三角网格放到前沿对队列Ω_C中，这些前沿三角形的集合形成待剖分域。

(2) 按照 8.2.1 节介绍的方法，求得每个节点的参照高度和搜索半径。把Ω_C中所有节点及其参照高度和搜索半径放到背景网格中。



a)以三角形中心为基准　b) 以长边中点为基准

a) center of triangle; b) midpoint of the longest edge

图 8.3 理想点的确定

Fig 8.3. New node position

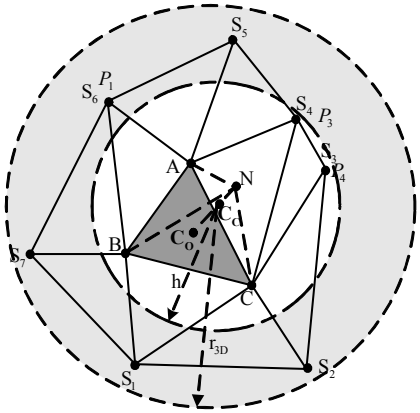


图 8.4 可行性检查

Fig.8.4 Valid Check

8.2.3. 前沿推进

对当前层的前沿队列Ω_C中的每一个前沿向前推进，其推进算法如下：

Step.1 计算单元的理想尺寸和理想点

如前所述，前沿队列是以前沿优先因子γ为优先序的一个优先对列，所以在前沿推进的时候首先从前沿队列里取出第一个前沿三角形，记为ΔABC。那么新生成的单元的理想尺寸有以下公式确定：

$$h = \max(\frac{h_{oA} + h_{oB} + h_{oC}}{3}, \frac{\sqrt{\alpha_k}}{2}) \tag{8.5}$$

$h_{oA}, h_{oB}, h_{oC}, \alpha_k$分别表示三角形三个节点的参照高度和面积。单元理想尺寸确定以后，四面体的理想点可以按下面公式确定。

$$N = C_o + gh\overrightarrow{D_k} \tag{8.6}$$

C_o是计算理想点的基准点，如图 8.3 所示，当ΔABC的质量系数$\beta > 0.5$时C_o是三角形的中心，否则C_o是三角形最长边的中心点。系数 g 是一个动态调整地参数，其初始值是 1.0。

Step.2 形成可见前沿Ω_S、可见节点集ψ_S和可见边集Φ_S

所谓可见前沿是在当前前沿周围的一系列前沿，记为Ω_S。首先取得当前前沿ΔABC三个节点的搜索半径r_{oA}, r_{oB}, r_{oc}，则前沿的搜索半径为：$r_{3D} = \delta.\max(r_{oA}, r_{oB}, r_{oc})$，$\delta$是一个经验值,本文取 1.2。当前前沿的可见节点集ψ_S是落在球$S(x_C, r_{3D})$中的所有位于当前层中的节点，其中球心C_c按如下公式计算。

$$C_c = 0.4C_o + 0.6N \tag{8.7}$$

在构成搜索节点集时，构建一个球$S(x_C, r_{3D})$的外接立方体可以大大加快节点搜索的过程，也就是说，首先取得落在球外包围盒中的所有节点，然后判断这些节点是否落在球内。这样所有落在球$S(x_C, r_{3D})$的节点就构成了搜索节点集ψ_S。和可见节点集中所有节点连接的属于当前前沿队列Ω_C中的三角形构成可见前沿Ω_S，而可见构成可见前沿的所有边形成可见边集Φ_S。

Step.3 形成活动球和活动节点队列ψ_A

当可见前沿和可见节点集形成以后，根据已有点优先的原则，接下来的工作是形成活动球和活动节点队列。所谓活动球是以理想点C_c为圆心，当前前沿的参照高度 h 为半径的球。把属于搜索节点集并且在活动球内位于ΔABC上方的节点，按照和理想点距离

最近为优先的原则进行排序，形成活动节点队列ψ_{A}，最后把当前理想点N放入ψ_{A}的最后。

Step.4 在活动节点队列ψ_{A}查找合适的点

依次从活动节点队列顶部弹出一个节点P和当前前沿ΔABC组成四面体单元T_{ABCP}。如果节点P是理想点N，则要进行下列(3-5)的节点可行性检查，否则要进行下列所有检查（如图 8.4）：

(1) 体积检查：即$(\overrightarrow{AB}\times\overrightarrow{AC}).\overrightarrow{AP}>\varepsilon$，即点$P$在$\Delta ABC$的上向，且体积大于一个最小容许值，本文$\varepsilon=0.01*h*\alpha_k$，$h$是当前前沿的参照高度，$\alpha_k$是当前三角形的面积；

(2) 距离检查：即点 P 到基准点的距离C_o的距离满足：$d_{PC}<d_{\max}$。默认$d_{\max}=2h$，如当前三角形的质量系数$\beta<0.5$，并且当前三角形的节点所连接的最长边的边长$l_{\max}>2h$时$d_{\max}=l_{\max}$。

(3) 不包含其它节点检查：即四面体T_{ABCP}中不包含任何一个搜索节点。也就是说对于ψ_{S}中的任何一个除A,B,C,P 以外的任何一个节点P_i ，应该满足下列四条件得任何一个或多个：$(\overrightarrow{AB}\times\overrightarrow{AC}).\overrightarrow{AP_i}>0,(\overrightarrow{AC}\times\overrightarrow{AP}).\overrightarrow{AP_i}>0,(\overrightarrow{AP}\times\overrightarrow{AB}).\overrightarrow{AP_i}>0,(\overrightarrow{BP}\times\overrightarrow{BC}).\overrightarrow{BP_i}>0$

(4) 夹角检查：假如新生成的四面体T_{ABCP}的第四个点P离相邻三角前沿的距离过近的话，势必在后续生成四面体的过程中生成质量较差的薄元。为此需要进行下列边-面，面-边的夹角检查。本文中把在当前前沿队列中和当前前沿ΔABC共享同一节点的三角形称为ΔABC的相邻三角形。如图 8.4 所示，节点$S_i(i=1,...7)$和ΔABC的三个边组成的三角形就是ΔABC的相邻三角形。

① 边-面夹角检查：当$P_i\neq S_i$时，边AP_i、BP_i、CP_i分别和其对应的共享同一节点的相邻三角形的夹角$\alpha_1\geq\alpha_{\min}$

② 面-边夹角检查：ΔABP_i、ΔBCP_i、ΔCAP_i和其相邻三角形中共享同一节点且除边AB、BC、CB以外的所有边的夹角$\alpha_2\geq\alpha_{\min}$。

以上检查中，首次剖分时$\alpha_{\min}=6^o$，当进入第n次回退过程时，$\alpha_{\min}=6^o\sqrt{\beta_{tri_{\min}}}$ 。$\beta_{tri_{\min}}$是当前前沿队列中质量系数最小的三角形的质量系数。

(5) 相交检查：这也包括两个方面：

① 边-面相交检查：即边AP_i、BP_i、CP_i和可见前沿Ω_{S}中的任何一个三角面不能相交。

② 面-边相交检查：即 ΔABP_i、ΔBCP_i、ΔCAP_i 和可见边集 Φ_S 中的任何一条可见边不能相交。

如果对活动节点队列中的节点 P，以上检查都通过则进入 **Step.7**,否则进入 **Step.5**。

Step.5 在可见节点集 ψ_S 中查找合适的点

此时把 ψ_A 从 ψ_S 剔除后的所有节点按照和理想点距离由小到大为优先序的原则进行排序，形成新的节点队列 ψ_L。依次从 ψ_L 取出一个节点 P，进行 step4 中的所有检查。若检查都通过则进入 **Step.7**，否则进入 **Step.6**。

Step.6 重新计算并判断理想点

把公式(8.6)中的系数 g 由 1.0 改为 $0.7, 0.7^2$ 和 0.7^3 计算得到 3 个新的理想点 N_1, N_2, N_3。对这三个点依次进行 step4 中(3-5)的节点可行性检查。如果有一个点通过以上检查则进入 **Step.8,**否则进入 **Step.7**。

Step.7 把当前前沿放到 Ω_H 中

此时认为当前前沿 ΔABC 在当前条件下不可推进，需要特别处理。在这里，把前沿 ΔABC 放到 Ω_H 中，并把它从当前前沿队列 Ω_C 中删除。

Step.8 更新前沿队列

把在 **step4，step5，step6** 中的任何一步得到的节点赋给可行点 P_F。如果 P_F 是新生成的理想点，那么就会新生成三个前沿 ΔABP_F、ΔBCP_F、ΔCAP_F，如果 P_F 是已有点，那么就会新生成三个前沿 ΔABP_F、ΔBCP_F、ΔCAP_F 中的两个前沿。把所有新生成的前沿三角形以当前优先因子为序放到下一层的前沿对队列 Ω_N 中，同时把 ΔABC 从当前前沿队列 Ω_C 中删除。

以上步骤(Step1-Step8)重复执行,当 Ω_C 为空时，互换 Ω_C 和 Ω_N，直到 Ω_C 和 Ω_N 都为空时，初次剖分结束。

8.2.4 局部网格重新生成

一般来说按以上过程完成初始剖分后，都会在剖分域内或多或少留下一些不可剖分的前沿，也就是说一般 Ω_H 是一个非空集合。经验表明，在采用 AFT 方法生成有限元网格时，生成网格质量的高低和成败很大程度上依赖于前沿的推进路径的选择。一般来说小面积优先的情况下，生成的网格整体质量较高，所以本文中初始剖分采用小面积优先的策略。对于初始剖分和回退过程中形成的内核需要删除形成新的空腔，对于这些空腔的剖分就需要尝试不同的剖分策略，虽然这种局部网格重生成的方法虽然无法保证每一次都能够在局部区域成功采用 AFT 方法进行网格生成，但在很大程度上是有效的。在

经过多次改变推进策略进行回退仍旧不能成功剖分的内核最典型的情况是 Schoenhardt 多面体，对于这些内核对面体本文采用宋超等[86]给出的基于线形规划的通过在内核多面体内插入新节点的方法来解决，简述如下：

设 Schoenhardt 多面体的三角形表面外法线方向指向多面体外部，欲保证作为 steiner 节点的 P 点在多面体内部，就要使 P 点与每一个表面 $\Delta V_iV_jV_k$ 之间满足 $\overrightarrow{V_iP}\bullet\vec{n}_i<0$，其中 $\vec{n}_i$ 为 $\triangle\Delta V_iV_jV_k$ 的外法线方向向量。设 V_i 点坐标为 $V_i(x_v,y_v,z_v)$ ，$\vec{n}_i$ 向量三个分量为 (n_x,n_y,n_z) ，P 点坐标为 (x,y,z) 则上述不等式转化为

$$(x-x_v,y-y_v,z-z_v)\bullet(n_x,n_y,n_z)<0$$

即: $xn_x+yn_y+zn_z<x_vn_x+y_vn_y+z_vn_z$ （8.8）

对于（8.8）中提出的不等式，可以归纳为形式 $AX<b$ ，其中，

$$A=\begin{bmatrix} n_{1x} & n_{1y} & n_{1z} \\ n_{2x} & n_{2y} & n_{2z} \\ & \vdots & \\ n_{8x} & n_{8y} & n_{8z} \end{bmatrix},\quad X=\begin{bmatrix} x \\ y \\ z \end{bmatrix},\quad b=\begin{bmatrix} x_{1v}n_{1x}+y_{1v}n_{1y}+z_{1v}n_{1z} \\ x_{2v}n_{2x}+y_{2v}n_{2y}+z_{2v}n_{2z} \\ \vdots \\ x_{8v}n_{8x}+y_{8v}n_{8y}+z_{8v}n_{8z} \end{bmatrix}$$

若再构造一个目标函数 f，则可将不等式求解问题转化为线性规划问题，此线性规划问题的构造思路如下。对于不等式约束 $AX<b$，通过引入一个松弛变量 S 对其进行改造，即

$$AX+S<b \qquad (8.9)$$

则线性规划问题的完整提法为，

$$\begin{cases} \min & f=S \\ s.t. & AX+S<b \\ & S\geq 0 \end{cases}$$

此线性规划问题与原来的不等式方程组是等价的。解此线性规划问题可以得出这样的结论，即如果最优值 $f=0$，那么说明 $S=0$，则原来的不等式方程组有解。如果 $f\neq 0$ ，那么说明 $S>0$，那么利用此线性规划问题求出的最优解不满足原来的某个不等式，这说明原来的问题无解。综上所述，整个回退过程主要分为以下三步：

(1). 删除所有与不可剖分前沿 Ω_H 相关的四面体单元，形成内核空腔；

(2). 内核空腔重新生成四面体单元。

(3). 采用线性规划的方法通过插入新节点完成最后若干个单元的生成。

对于不能按当前剖分要求继续推进形成四面体单元的前沿集合Ω_H，需要删除Ω_H中节点所连接的所有四面体单元，并形成包围这些四面体单元外空腔三角形面，这些三角面形成新的剖分域，在新的前沿优先推进因子下，继续采用 3.2 节的推进算法对该剖分域进行四面体剖分。这个过程称为回退，对于绝大多数模型，通过设置不同前沿优先推进因子，经过若干次回退过程就能完成整个剖分的四面体剖分。设当前进行第 i 次回退过程，按公式(4)，计算当前的前沿优先推进因子γ_i。这样在第 i 次回退算法如下：

第 i 次回退算法:

(1) 设初始剖分形成的四面体单元集合为$\Lambda(T)$。

(2) 提取Ω_H中所有节点形成节点表ψ_{H}。

(3) 计算和ψ_{H}相连的所有四面体单元$\Lambda(T_{\mathrm{H}})$。

(4) 设包围$\Lambda(T_{\mathrm{H}})$的空腔面为Ω_{ca}，则它的形成如下

(5) 对于每一个四面体单元$T_{\mathrm{Hi}} \in \Lambda(T_{\mathrm{H}})$

(6) 如果对于属于T_{Hi}的每一个法线方向指向T_{Hi}内部的三角面$\Delta_{\mathrm{Hi}} \notin \Omega_{ca}$，则$\Delta_{\mathrm{Hi}}$以$\gamma_i$为优先序插入$\Omega_{ca}$

(7) 对于Ω_{ca}中的每一个Δ_{Hi}，如果$\Delta_{\mathrm{Hi}} \in \Omega_H$并且$\Delta_{\mathrm{Hi}} \notin \Omega_O$（$\Omega_O$为初始前沿队列)，则从$\Omega_{ca}$中删除$\Delta_{\mathrm{Hi}}$。这样$\Omega_{ca}$就形成了，现在对$\Omega_{ca}$围成的区域进行重新剖分，为此：

(8) 清空Ω_H，按照公式（2）进行数据更新

(9) 设置当前层的前沿队列$\Omega_c = \Omega_{ca}$

(10) 调用 3.2 节的剖分过程进行剖分。

如果上述回退算法经过若干次(本文最多经过 20 次回退)后仍旧存在不可剖分的多面体，那么就采用上述的基于线形规划的插点算法，完成最后几个内核多面体的剖分。

8.3 自适应四面体网格的生成

采用黎曼度量(metric tensor)来控制单元的尺寸和形状是自适应网格生成的通用方法。在三维空间中某一点的黎曼度量是一个3×3实对称阵,在考虑黎曼度量的情况下线段$AB = (A + t\overrightarrow{AB})_{0\le t\le 1}$的长度计算公式为：

$$l(AB, M) = \int_0^1 \sqrt{\overrightarrow{AB}^T M(A + t\overrightarrow{AB})\overrightarrow{AB}}\,dt \qquad (8.10)$$

这里$M(A+t\overrightarrow{AB})$是点$A+t\overrightarrow{AB}$处的黎曼度量。网格自适应的目标就是按公式计算得到的每条边的长度为 1。 这里首先根据用户自适应的要求生成待剖分域的黎曼度量，可

以在四面体内部、三角面或边上插入新节点的方法来达到自适应目的。经验表明在边上插入节点的方法实现简单而且效果较好，具体实现方法简述如下：

(1) 遍历四面体网格中所有的边，如果对于某条边，按照公式 10 计算得到的长度 $l>\sqrt{2}$，则待分裂的边的集合 L 中。

(2) 对 $L=\{l_i,i=1...n\}$ 中的边按长度进行排序，使得 $l1\geq l2\geq ...\geq l_n$

(3) 依次遍历 L 中的所有边，对于当前边 l_i， 对分和 l_i 连接的所有四面体单元，对于新产生的边，如果边长 $l>\sqrt{2}$，放到集合 N 中。如果集合 N 的最长的边大于 l_i，那么这里需要首先分裂这条最长边，然后继续分裂 L 的其它边。

(4) 清空 L，并使得 $L=N$。

(5) 重复 2）~4）的步骤，直到 L 和 N 都为空，自适应过程结束。

8.4 单元优化

在四面体单元生成的过程中，虽然每生成一个单元都力图使之质量最好，但其最终结果中还是会存在单元质量很差的单元。本文对四面体单元主要以下两个方面的优化：

8.4.1 节点删除

如图 8.5a 所示，如果一个实体内部的节点 P 连接 4 个四面体($T_{ABCP},T_{ACDP},T_{ADBP},T_{BDCP}$)，那么这 4 个四面体可以合并为 1 个四面体 T_{ABCD}，从而删除节点 P。另外一种节点可以删除的情况如图 13b 所示，如果一个位于实体内部的节点 P 连接 6 个四面体（$T_{AEBP},T_{ABDP},T_{ADEP},T_{BECP},T_{BCDP},T_{CEDP}$），那么这 6 个四面体可以合并为 2 个四面体(T_{ABDE},T_{BCDA} 或者 T_{ABCE},T_{ACDE}，取这两种合并中 2 个四面体质量系数最小的最大优先)。

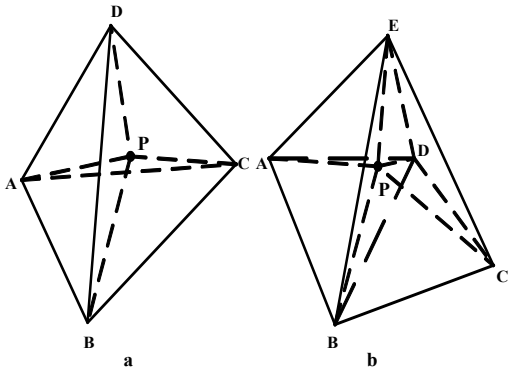


图 8.5 节点删除

Fig.8.5 Node remove

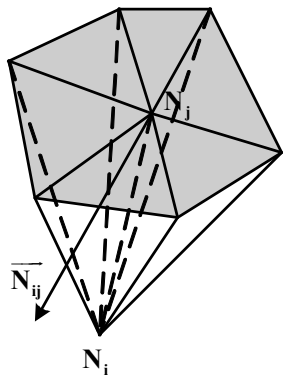


图 8.6 基于角度的优化

Fig.8.6 Angle-based smooth

8.4.2 基于角度的优化

在网格优化中，拉普拉斯优化（Laplacian smoothing）是经常使用的一种优化方法。这种方法的突出优点是速度快，然而这种优化方法并不能保证所有参加优化的单元其质量都有所提高，有时甚至产生负体积的单元。基于这个原因, S. A. Canann[77]提出了约束拉普拉斯优化方法，然而这种方法的效率有所降低。Zhou[74]等人提出了一种在不损失效率的前提下提出了一种基于角度的二维单元的优化方法，Ito, Y[131]等把这种方法扩展应用到三维四面体的优化中，现简述如下：

如图 8.6 所示，设实体内部的节点 N_i 的坐标为 x_i，和 N_i 相连的四面体和边的个数是 n_{ti}、n_{ei}。取和节点 N_i 相连的节点 N_j，则节点 N_i 和 N_j 构成节点 E_{ij}。设 $T_{ik}(k=1到n_{ij})$ 是所有包含边 E_{ij} 的四面体(n_{ij} 是包含 E_{ij} 的四面体的总数。 $f_{ik}(k=1到n_{ij})$ 是 T_{ik} 中节点 N_i 对面，也就是图中的阴影部分，那么节点 N_i 的这些对面的平均单位法向量（ $\overrightarrow{N_{ij}}$ ）和面积和 a_{ij} 为：

$$\overrightarrow{N_{ij}}=\frac{\sum_{k=1}^{n_{ij}}\overrightarrow{N}_{fik}}{\left|\sum_{k=1}^{n_{ij}}\overrightarrow{N}_{fik}\right|},\quad a_{ij}=\sum_{k=1}^{n_{ij}}a_{fik} \tag{8.11a}$$

如果单独考虑节点 N_j，新节点的坐标为：

$$x_i'=x_j+\left|x_j-x_i\right|\overrightarrow{N_{ij}} \tag{8.11b}$$

$\left|x_j-x_i\right|$ 表示 E_{ij} 的长度。因为一共有 n_{ei} 边和 N_i 相连，以面积和为权求取优化后新节点的位置：

$$x_i'=\frac{\sum_{j=1}^{n_{ei}}a_{ij}(x_j+\left|x_j-x_i\right|\overrightarrow{N_{ij}})}{\sum_{j=1}^{n_{ei}}a_{ij}} \tag{8.11c}$$

在程序实现的过程中首先计算节点 N_i 连接的四面体的质量系数的最小值 $\beta_{tet}^{\min}$。和三角形质量系数的定义类似，四面体的质量系数 β_{tet} 可以用四面体的内切圆半径 $\mathbf{r_{tet}}$ 和外接定义圆半径 $\mathbf{R_{tet}}$ 的比值确定:

$$
\begin{aligned}
r_{tet} &= \frac{3V}{\sum_{i=0}^{3} s_i} \\
R_{tet} &= \frac{\sqrt{16a^2b^2 - (a^2+b^2-c^2)^2}}{24V} \\
\beta_{tet} &= I\frac{3r_{tet}}{R_{tet}} = I\frac{216V^2}{\sum_{i=0}^{3} s_i\sqrt{4a^2b^2 - (a^2+b^2-c^2)^2}} \\
I &= \begin{cases} 1 & (V>0) \\ -1 & (V<0) \end{cases}
\end{aligned}
\tag{8.12}
$$

在这里V是四面体的体积，在这里如果四面体的节点序列右手定则体积为正（$I=1$），否则为负($I=-1$),通过引入符号I，可以防止优化过程中产生负体积的单元。a,b,c是四面体两两相对的边的边长积，S_i是四面体某个面的面积。这样按公式（8.11）得到新节点的位置，然后计算在位置下这些单元的质量系数的最小值$\beta_{tet}^{\min}{}'$。如果$\beta_{tet}^{\min}{}' > \beta_{tet}^{\min}$则接受新的位置，否则继续节点集合下一个内部节点的优化，直到所有内部节点优化完毕。

8.5 数值算例

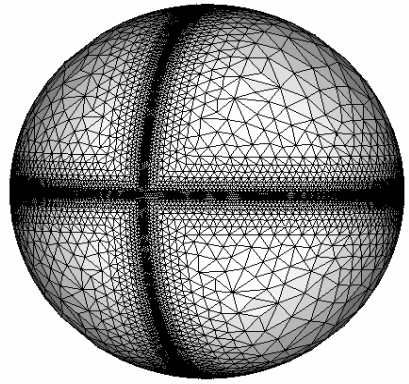


图 8.7（a）自适应算例 1
Fig.8.7 (a) Example 1

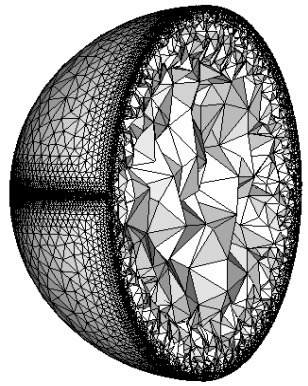


图 8.7（b）算例 1 的剖视图
Fig8.7 (b) Cutaway view of example 1

图 8.7 是一个球的自适应剖分结果。图 8.8(a)是一个城市雕塑的三维四面体剖分算例，图 8.8 (b)是该模型的一个剖视图。图 8.9(a)是另一个组合曲面模型的三维四面体剖分算例，图 8.9(b)是该模型的一个剖视图。图 8.9(c)是在 PC1.86GB,512M 内存，windowns 系统环境下单元生成个数和所花费时间的关系曲线。经过对多个实际使用的机械零件剖分统计，本文提出的算法时间复杂度基本为 O(n),平均一分钟生成 6 万个单元，显然这已经能满足实际工程的需要了。图 8.9(d)是图 11 中的质量系数计算公式得到的该算例生成 36459 个单元时优化前后单元质量系数分布图。由图中可以看出优化后单元 36432 个，

通过节点删除和合并单元，单元总数减少 27 个。本算法最终生成的四面体单元质量系数小于 0.1 很少（本算例为 5 个，最小 0.068）。

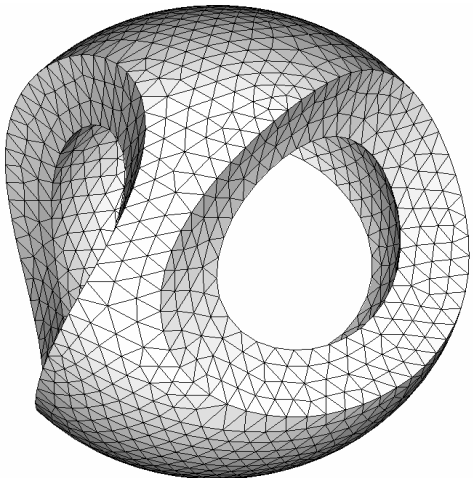


图 8.8（a）数值算例 1

Fig.8.8(a) Example 1

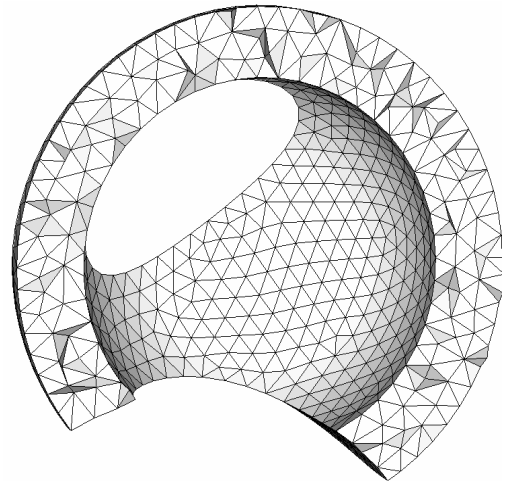


图 8.8（b）算例 1 的剖视图

Fig 8.8(b) Cutaway view of exapmle 1

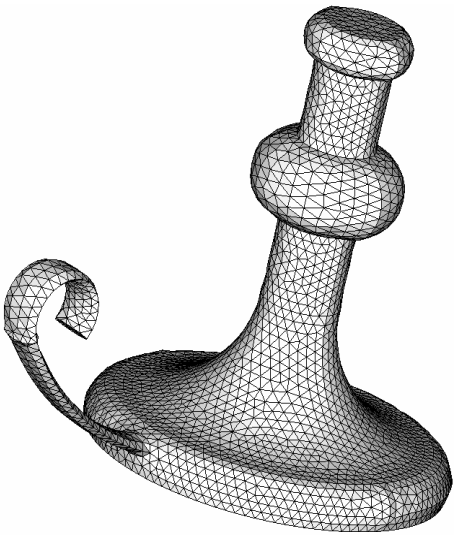


图 8.9（a）数值算例 2

Fig.8.9(a) Example 2

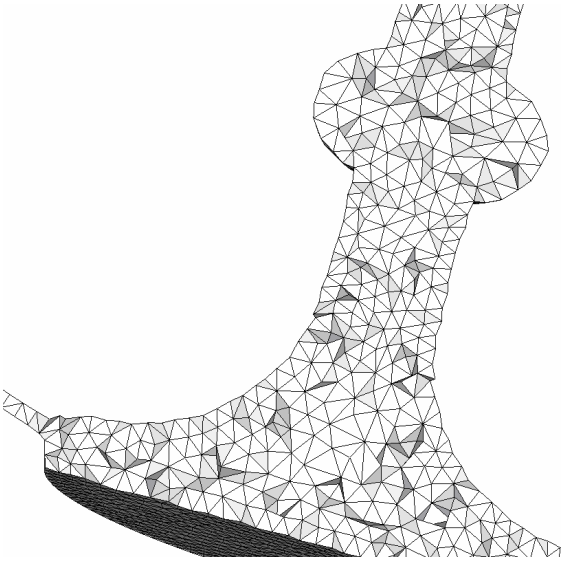


图 8.9（b）算例 2 的剖视图

Fig8.9(b) Cutaway view of exapmle 2

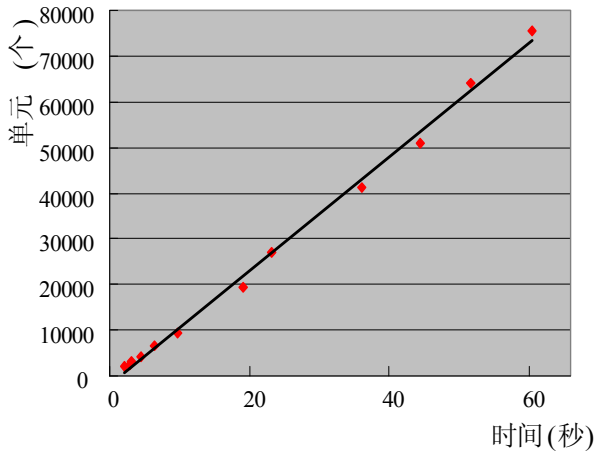


图 8.9（c）算例 2 的单元个数和时间的关系曲线

Fig. 8.9(c) CPU Time of 3D exapmle 2

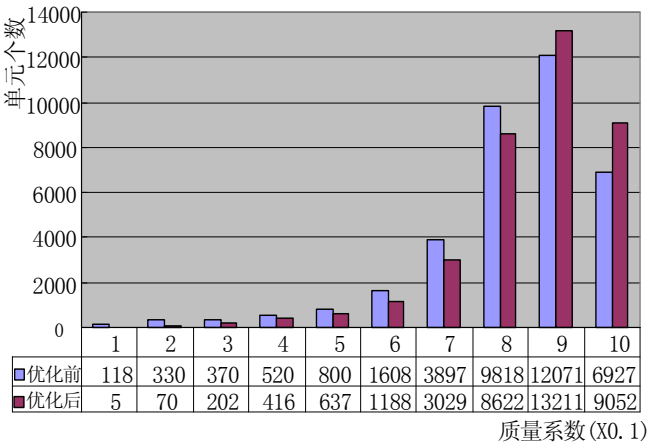


图 8.9（d）算例 1 的单元质量系数分布图

Fig8.9 (d) Statistics of triangle quality of exapmle 2

9. 基于转换模板的三维实体全六面体网格生成方法

9.1 引言

六面体单元因其求解精度高的特点在很多有限元分析领域中都是一种十分重要的单元。在某些情况下，如当用有限体积法与边界适应坐标系统求解复杂形体的控制守恒方程时，只能采用六面体单元进行有限元分析。对任意复杂三维实体实现可靠、高质量的全六面体网格自动生成一直是 CAD/CAE 领域内的一个难点。本文以四面体-六面体基本转换模板为基础，提出了一系列具有伸缩性的扩展转换模板，可将四面体分解为不同数量、不同密度过渡形式的的六面体单元；提出了基于几何造型的边界节点坐标修正方法，使边界网格能够更好地拟合几何模型边界。

9.2 四面体-六面体扩展转换模板

四面体-六面体基本转换模板的形式如图 9.2 所示。该模板的在工程应用中是有效的，但也存在如前所述的两个主要缺点：网格疏密过渡不灵活和生成单元质量不高，因而该单元的应用并不广泛。实际上基于该模板可以扩展出一系列更为有效的转换模板。

(1) 1—4 转换模板

它是其它转换模板的基本形式，也称为基本转换模板。为了叙述方便，本文称之为核。

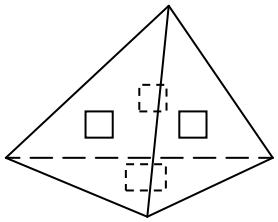


图 9.1、待剖分的目标四面体

Fig.9.1. Tetrahedron to be subdivided

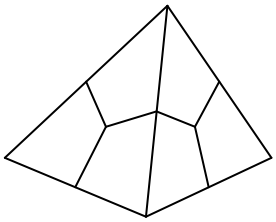


图 9.2、基本转换模板

Fig.9.2. Basic template

(2) 1—7 转换模板

如图 9.3 所示。相当于基本转换模板与目标四面体（见图 9.1）有三个面共面的情况。

(3) 1—10 转换模板

如图 9.4 所示。相当于基本转换模板与目标四面体（见图 9.1）有二个面共面的情况。

(4) 1—12 转换模板

如图 9.5 所示。相当于基本转换模板与目标四面体（见图 9.1）有二个面共面的情况，但连接方式与 1—10 转换模板不同。

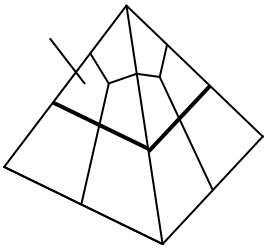


图 9.3、1—7 转换模板
Fig.9.3. 1 to 7 template

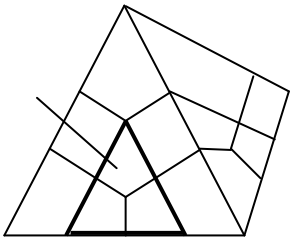


图 9.4、1—10 转换模板
Fig.9.4. 1 to 10 template

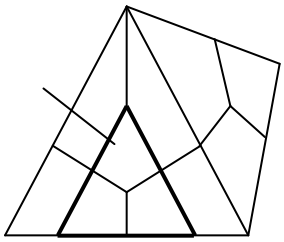


图 9.5、1—12 转换模板
Fig.9.5. 1 to 12 template

(5) 1—13 转换模板

如图 9.6 所示。相当于基本转换模板与目标四面体（见图 9.1）有一个面共面的情况。

(6) 1—20 转换模板

如图 9.7 所示。相当于基本转换模板与目标四面体（见图 9.1）有一个面共面的情况，但连接方式与 1—13 转换模板不同。

(7) 1—28 转换模板

如图 9.8 所示。相当于基本转换模板完全在目标四面体（见图 9.1）内部的情况。

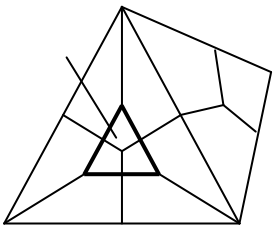


图 5.6、1—13 转换模板
Fig.5.6. 1 to 13 template

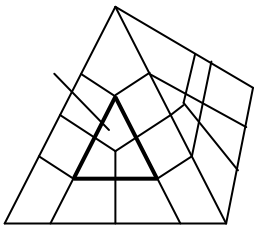


图 5.7、1—20 转换模板
Fig.5.7. 1 to 20 template

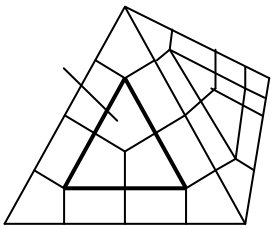


图 5.8、1—28 转换模板
Fig.5.8. 1 to 28 template

以上转换模板可以组合使用，以完成网格的不同形式的疏密过渡。转换后的网格还可以进行再细化，例如将一个六面体网格细化为八个小六面体。

9.3 算法总体流程与边界节点坐标修正方法

为了实现后期的六面体网格表面节点的坐标修正，需对现有算法[17]进行扩展。具体地，在生成实体表面三角形网格时，对网格的节点赋予额外的三个属性：节点的定位类型、所在的曲面（Surface）或边(Edge)，以及节点参数坐标。节点的定位类型有 4 种：0-在实体内部；1-在曲面内；2-在参数边界上；3-在两个曲面的交线上。对于定位类型

为 1、2 的节点，节点参数坐标是所在曲面的参数坐标（u-v 值）；对于定位类型为 3 的节点，节点参数坐标则是所在边的参数坐标(t 值)。在生成实体表面三角形网格时不会出现定位类型为 0 的节点，只有在生成实体网格时才会出现定位类型为 0 的节点，该类型节点不需要作节点坐标修正处理。

9.3.1 边界节点坐标修正方法

四面体网格在转换为六面体网格时会产生新节点。内部新节点不会带来特殊问题，但对于在表面上产生的新节点就需要进行特殊处理以保证它们落在几何模型的表面上。文献[27]采用十节点曲边四面体转换六面体网格，四面体中位于几何模型表面上的面是由三个曲边描述的曲面。但是当初始网格尺寸较大时，该类曲面并不能准确描述几何模型的边界表面。这样，生成的新节点与真实边界会有一定的误差。对此，本文提出一种基于几何造型的边界节点映射处理方法。

当一个四面体单元通过转换模板分解为多个六面体时，会产生三类新节点：四面体的内部节点，四面体的表面节点和四面体的边节点。对新节点，需按 5.5 提供的信息确定节点的定位类型。四面体的内部节点不必进行特殊处理。对于四面体的表面节点，按以下方法确定定位类型：

a. 当所在面的三个顶部节点的定位类型均为大于 0 时，则新生成的节点的定位类型为 1；

b. 当所在面的三个顶部节点中有一个节点的定位类型为 0 时，则新生成的节点的定位类型为 0。

对于四面体的边节点，按以下方法确定定位类型：

a. 若所在边的一个端部节点的定位类型为 1，另一个端部节点的定位类型为 1 或 2 或 3，则新生成的节点的定位类型为 1；

b. 若所在边的两个端部节点的定位类型均为 2，则新生成的节点的定位类型为 2；

c. 若所在边的一个端部节点的定位类型为 2，另一个端部节点的定位类型为 3，则新生成的节点的定位类型为 1；

d. 若所在边的两个端部节点的定位类型均为 3，则新生成的节点的定位类型为 3；

e. 对于其它情况，新生成的节点的定位类型为 0。

在确定节点的定位类型之后，就可以对不同定位类型的边界节点采用不同的映射方式进行坐标修正。对于定位类型为 1、2 的节点，用曲面的参数坐标(u-v 值)到物理坐标的转换；对于定位类型为 3 的节点，用曲线的参数坐标(t 值) 到物理坐标的转换。对于定位类型为 0 的节点不需要进行处理。通过上述映射处理方法，边界节点可以精确地落在几何模型的表面上。

9.3.2 算法总体流程

下面给出基于转换模板的三维实体全六面体网格生成算法的总体流程:

Step1: 对给定的三维实体模型的表面进行网格划分，生成三角形网格；

Step2: 对网格的节点赋予额外的三个属性：节点的定位类型、所在的曲面（Surface）或边(Edge)，以及节点参数坐标；

Step3: 根据三维实体的表面网格，采用 AFT 方法生成实体内部的四面体网格；

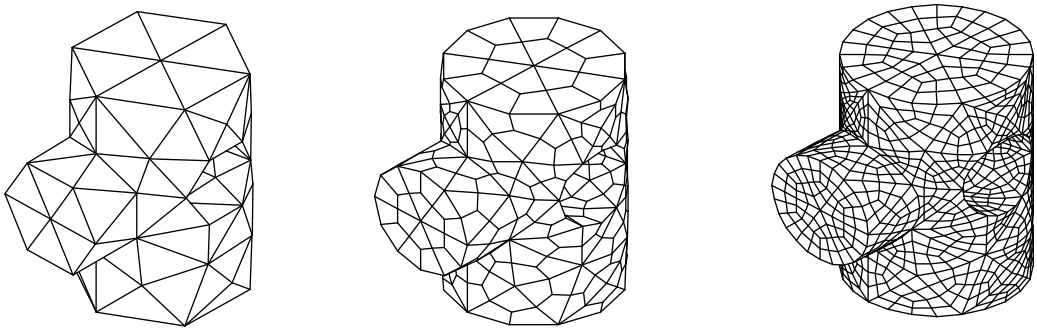
Step4: 利用转换模板将四面体网格分解为六面体网格；

Step5: 根据 step2 提供的信息，对 Step4 产生的新节点进行归类，并计算定位类型为 1、2、3 的新节点的参数坐标；

Step6: 对于定位类型为 1、2 的节点，用参数曲面方程把节点参数坐标(u-v 值)转换为物理坐标；对于定位类型为 3 的节点，利用参数曲线方程将节点参数坐标(t 值) 转换为物理坐标。

9.4 数值算例

算例一为一具有贯穿孔的组合圆柱体，图 9.9(a)为初始四面体网格，图 9.9(b)为采用 1—4 转换模板转换生成的全六面体网格，图 9.9(c)为二次细化后的结果。可以看出，无论是第一次转换后网格还是细化后网格的边界节点均与实体表面符合得很好。算例二、三是复杂三维实体的全六面体网格生成。



(a) 初始四面体网格　(b)一次转换后全六面体网格　(c) 二次细化后全六面体网格

(a). Initial tetrahedral mesh　(b) Translate to all hexahedra mesh　(c) Using feat template to refine

图 9.9、算例一

Fig.9.9. Example of all hexahedra mesh 1

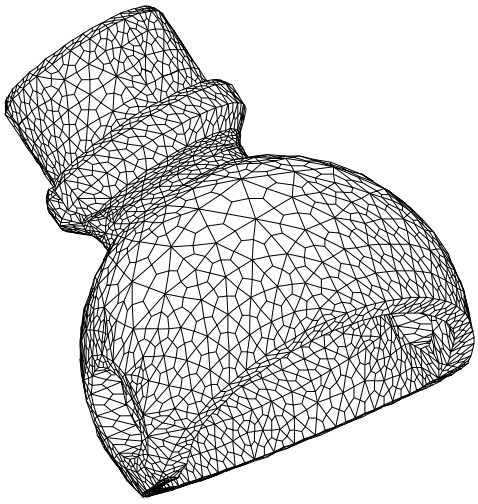


图 9.10　全六面体网格生成算例二

Fig. 9.10.　Example of all hexahedra mesh 2

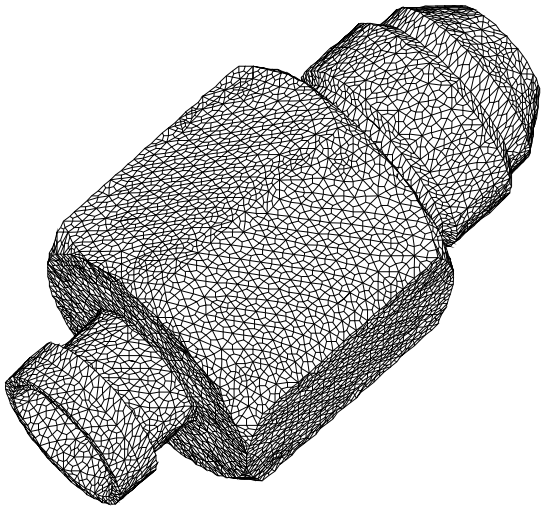


图 9.11 全六面体网格生成算例三

Fig. 9.11.　Example of all hexahedra mesh

10. 三维空间中点定位的回溯算法

10.1 前言

点的定位查找在网格生成及其它数值算法中应用广泛。例如，在本文第 5 章金属破坏模拟的中，将旧网格分析结果的状态变量转化到新的网格中时，需要找到新的节点在旧网格中的位置；在基于 STL 文件的网格生成算法中，生成的新节点也需要找到最近的三角形，然后才能投影到这个三角形或由这个三角形所构建的曲面上；在 AFT 的算法进行有效性判断中，需要首先找到落在当前前沿影响区域内的节点和前沿；在用 Q-Morph 方法实现四边形网格生成算法[132]中需要找到两个三角形之间最近或者接近最近的一条路径等等。归结起来，这些算法通常要解决两个问题：(1) 在 n-维空间中的点集中查找和当前点最近或者接近最近的一个或一些点；(2) 两个单元间的一条最短路径。

关于从一个点集中找到和当前点最近的一个点的算法，很多文献给出了详细的论述，其实现的时间效率基本在 O(NlogN)到 $O(N^2)$之间。ANN (Approximate Nearest Neighbors)[133], 是一个关于临近元素查找的 C++库，这个库实现了基于 k-d tree 和栅格的查找算法。Murphy 和 Skiena [134] 也是用 k-d tree 来实现这一算法，其效率也在 O(NlogN)到 $O(N^2)$之间。W. Randolph Franklin[135]给出了基于均匀栅格的三维空间最近点的查找算法，该方法的特点是：对于均匀分布的点集中最近点的查找速度快，理想状况下时间效率为 O(1)，内存占用少。

在很多应用中需要在一个三角剖分中找出某个点所在三角形，对这个问题通常采用分治算法[136]、无向环图[137]、有向查找法(Walk-through strategy)[138, 139]等。在这些方法中有向查找法是一种快速简单的方法，这种方法并不需要复杂的数据结构，只需要按照一定条件穿过一个三角形(或凸多边形)某条边走到临近的另外一个三角形，直到找到目标点为止。

走向计算方法的不同就有不同的有向查找法的实现，如线段相交测试法、逆时针法(Counter Clock Wise, CCW)[138]、重心坐标法(Barycentric Coordinates Search, BCS)[139]等，在这里我们重点研究后两种方法。CCW 法通过计算凸多边形的边和待查点的面积的符号来计算行走方向：如果计算的面积为负，下一个凸多边形就是和这条边共享的当前多边形的相邻凸多边形。一个凸多边形可能有不止一条边和待查点的面积大于零，这样就取其中任何一条边作为行走方向。这样用这种方法得出的行走路径就可能不是一条最短的路径。在 BCS 方法中首先计算待查点在当前凸多边形中的重心坐标，从而以“最快速度”找到目标凸多边形。但是上述两种方法并不能确定在任何情况下一定能够找到

目标凸多边形。实际上在特定的条件下，查找的路径可能是一个环，这样在不引入其它条件的情况下，可能永远找不到目标凸多边形。

在这里首先对 CCW 法和 BCS 进行简单的回顾，以此为基础介绍本文提出的可回退的 CCW 法来解决环的问题。另外文献中通常只是针对二维单连通凸域的方法，运用本章的算法不但可以对二维单连通凸域的剖分进行点的定位搜索，而且对于三维任意复杂域的凸多边形剖分仍然适用。

10.2 点的定位查找算法的一般思想

为了简化对问题的描述，这里首先以二维空间的平面三角剖分为例，来介绍点的定位查找算法的一般思想，然后把这个算法推广到三维空间的复杂域剖分的点的定位算法中。假设三角剖分中三角形的三个节点按逆时针排列。

10.2.1 定位算子和重心坐标

三角形 ΔABC 的有向边 AB 和点 C 的定位算子定义如下：

$$Det(ABC)=\begin{vmatrix} x_A-x_C & y_A-y_C \\ x_B-x_C & y_B-y_C \end{vmatrix} \tag{10.1}$$

这个算子是 ΔABC 面积的 2 倍，如果 ΔABC 的三个节点按逆时针排列，这个值为正，否则这个值为负。点 P 在 ΔABC 中的重心坐标可以这样计算：

$$r=\frac{Det(PBC)}{Det(ABC)},\quad s=\frac{Det(PCA)}{Det(ABC)},\quad t=\frac{Det(PAB)}{Det(ABC)} \tag{10.2}$$

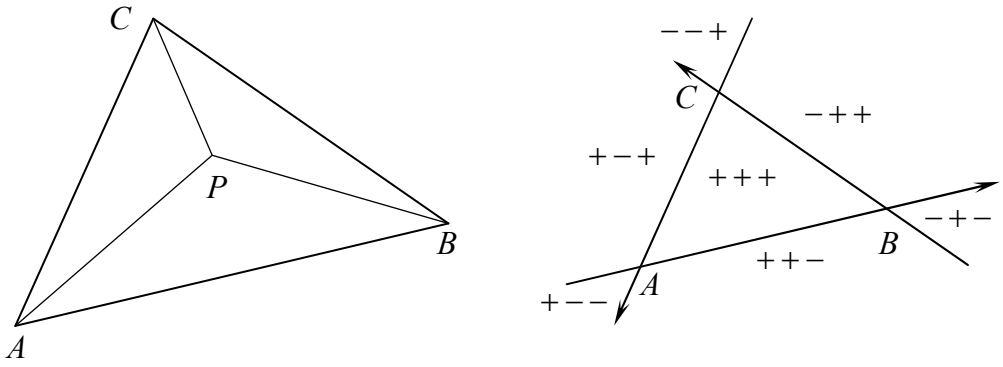


图10.1 重心坐标 r,s,t 和符号

Fig. 10.1 Barycentric coordinates r,s,t and their signs

重心坐标的 3 个值和三角形的三个点或者三条边相对应，如果点 在有向线段的左侧，定位算子为负(用“-”表示)，否则为正（用“+”表示）。从图 10.1 可以看出点 P 在三角形内的充要条件是重心坐标的三个值为正(“+++”)。一个点对一个三角形的重心坐标不可能存在三个都为负(“– – –”)的情况。

10.2.2 逆时针法(Counter Clockwise Wise Search, CCW)

如图 10.2 所示，假如搜索从 ΔABC 开始，CCW 方法计算点 P 和三角形各边的定位算子，如果定位算子为负，那么搜索路径从这条边进入临近的三角形，这种搜索过程一直进行下去，直到找到目标三角形为止：即待查点对三角形的三个定位算子都为正。由图 10.1 可以看出这种方法对当前三角形来说可能存在两条路径。对三角形的一条有向边来说，这种方法总是按照右手方向搜索，所以这种方法我们称为右手搜索法。

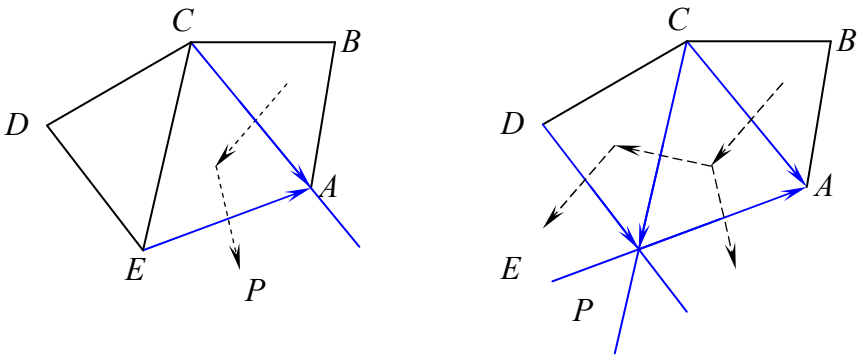


图 10.2 逆时针法的搜索路径

a) 一条搜索路径　　b) 搜索路径分叉

a) Single searching path　　b) Searching path bifurcation

Fig. 10.2 CCW searching path

10.2.3 重心坐标法(Barycentric Coordinates Search, BCS)

对当前三角形来说，BCS 算法这样计算下一个相邻的三角形：

(1) 计算点 P 对 ΔABC 的重心坐标 (r,s,t) 。

(2) 计算重心坐标的两个最大值，如图 10.2 中点 P 的三个重心坐标： $r(PBC)>0$，$r(PCA)<0$， $r(PAB)>0$ 得到线段 AC，找到和 ΔABC 共享边 AC 的 ΔACE

(3) 上述过程重复进行，直到三个重心坐标都为正。

从几何上来说这也是一个右手搜索法，和 CCW 方法不同的是，这里多了重心坐标的信息，因而搜索的路径可能比上一种方法要短。但是需要计算所有边的定位算子，才能得到两个最大的定位算子，从而得到相邻的下一个三角形。

10.3 逆时针法和重心坐标法的应用条件

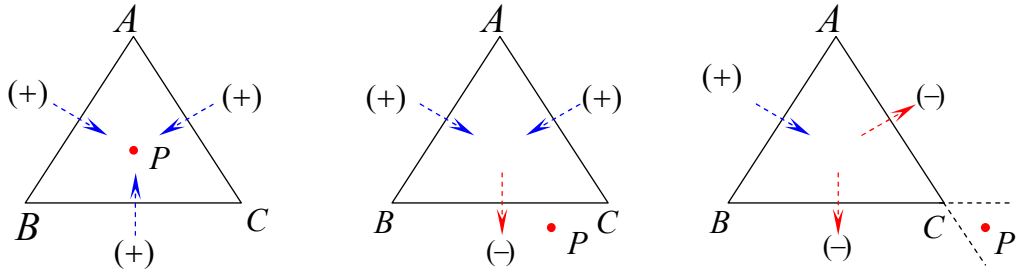
10.3.1 搜索路径分析和算法应用条件

如前所述，点 P 对 ΔABC 不同位置的重心坐标 (r,s,t) 的正负只有如下三种情况：

(1). “+++” 点 P 在三角形内，图 10.3(a)；

(2). “−++” 点 P 在三角形外，图 10.3(b)；

(3). “−−+” 点 P 在三角形外，图 10.3(c)。



(a) (b) (c)

图 10.3 由重心坐标决定的三角形的“进”“出”路径

Fig. 10.3 “Go in” or “go out” determined by signs of barycentric coordinates

如图 10.3 所示，P 对 ΔABC 的三个重心坐标，如其中对任条边的值为正，表示为“+”，说明搜索路径由这条边进入这个三角形，为负（表示为“-”），说明由这条边出去而进入和这条边相邻的三角形。应用 CCW 或者 BCS 的条件是：对于凸域，搜索路径不能形成环；对于凹域搜索路径除了不能形成环外，还不能使用三角剖分的边界边。但是，实际上，上述方法是有可能形成环，或到达边界边，下面一个例子就是二维凹域形成环的情况。

10.3.2 搜索路径为环的例子

以图 10.4 的例子为例，这个例子中是一个圆形域的一个三角剖分，搜索点 P 在圆心。点 $P_1,P_2,P_3\ldots$ 和点 $Q_1,Q_2,Q_3\ldots$ 分别在两个同心圆的圆环上，圆心为点 P。$\Delta P_2P_3Q_2$ 和 $\Delta P_3P_3Q_2$ 是 $\Delta P_1P_2Q_1$ 和 $\Delta P_2P_2Q_1$ 绕圆心 P 旋转($\theta=2\pi/n$)的结果，这样在圆 C_1 和圆 C_2 之间的三角形是由两个三角形集合构成的。在这里，显然点 P 在边 P_2Q_1 和边 P_2Q_2 的右侧，对于这个三角剖分待查点用 CCW 和 BCS 方法都会形成环，这样点 P 用这两种方法就不能找到。也就是这两种方法对整个凸域的三角剖分是失效的。在这个例子中，因为圆 C_1 和 C_2 的半径以及边 P_nQ_n ($n=1,2,\cdots$)是可以变化的，这样搜索路径就很容易满足 CCW 和 BCS 的条件。这里假设点 P_2 到线段 PQ_2 的距离大于 Q_1 到线段 PQ_2 的距离，对 $\Delta P_2Q_2Q_1$ 点 P 的重心坐标 $(r_{P_2},s_{Q_2},t_{Q_1})$ 为：

$$r_{P_2}=\frac{Det(PQ_2Q_1)}{Det(P_2Q_2Q_1)},\quad s_{Q_2}=\frac{Det(PQ_1P_2)}{Det(P_2Q_2Q_1)},\quad t_{Q_1}=\frac{Det(PP_2Q_2)}{Det(P_2Q_2Q_1)} \tag{10.3}$$

因为 $Det(PQ_1P_2)>0>Det(PQ_2Q_1)>Det(PP_2Q_2)$ 也就是 $s_{Q_2}>r_{P_2}>t_{Q_1}$，这样搜索路径经过 P_2Q_2 而不是 Q_2Q_1，这种过程对其它在 C_1 和 C_2 之间的三角形是类似的，从而形成环。

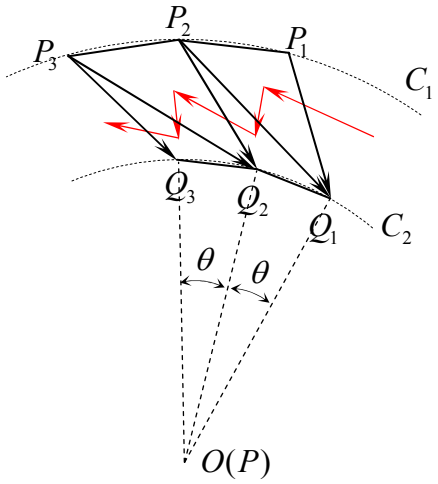


图 10.4 圆形域的三角化

Fig. 10.4 Triangle discretion in a circular field

10.4 点定位的回溯算法

10.4.1 术语定义

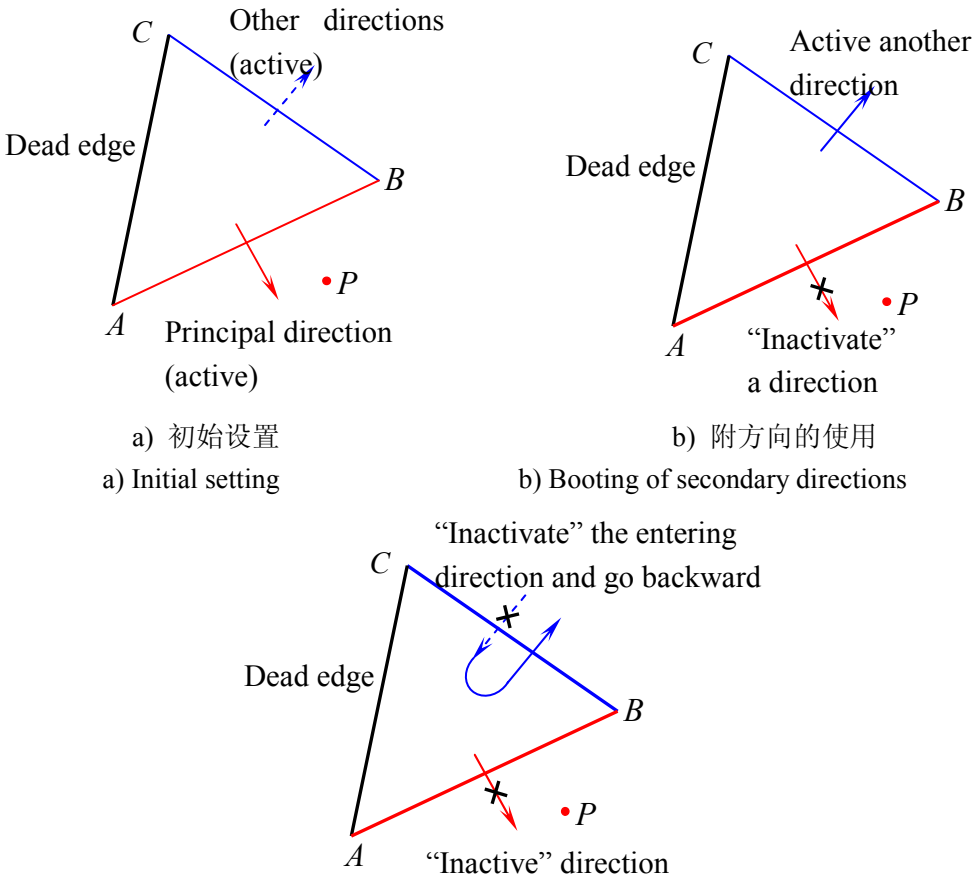
(1) 主方向：如图 10.3 所示，当前三角形对待查点的重心坐标中为负的边定义为主方向。也就是待查点在当前三角形该有向边的右侧。一个三角形对待查点可能有一个或两个主方向。

(2) 附方向：当前三角形对待查点的重心坐标中为正的边定义为负方向。也就是待查点在当前三角形该有向边的左侧。一个三角形对待查点可能有一个或两个附方向。

(3) 死方向：在三角剖分中，位于边界上的三角形，在边界边上没有相邻三角形，这样的边对应的搜索方向为死方向。

上述定义中，每个方向都和三角形的一条边相对应。

10.4.2 可回溯的搜索算法



a) 初始设置
a) Initial setting

b) 附方向的使用
b) Booting of secondary directions

c) 回退
c) Inactivate and go backward

图10.5 可回退搜索算法的关键步骤
Fig. 10.5 Key steps of the backward search algorithm

如上所述，在 CCW 和 BSC 算法中可能会形成环，并且对凹域可能失效。为了克服这种情况，本文的策略是：首先使用主方向作为搜索路径，在主方向失效时使用附方向作为可选搜索路径继续搜索。具体步骤如下（如图 10.5）：

(1) 激活三角形所有的主方向和附方向，边界边设为死方向，如图 10.5a。

(2) 从一个三角形开始进行搜索，这个初始三角形可以是用户输入的，默认情况下可以三角剖分中任意一个三角形。

(3) 对当前三角形，按下列顺序搜索相邻三角形。①第一主方向；②如果存在第二主方向，使用第二主方向；③第一附方向；④如果存在第二附方向，使用第二附方向。上述顺序是按优先序有高到低排列的。当优先序高的方向可以使用时，就暂时不使用优先序

低的方向。如果三角形中某一活动方向已被使用，那么这个方向由活动状态变为非活动状态。

(4) 如果当前三角形中所有的方向状态为非活动或死方向，那么就回退到最近一次进入该三角形的上一三角形，图 10.5c。

(5)当待查点已经找到，或者三角剖分中所有三角形已经被遍历，就结束上述过程。

上述过程可以推广到 n-边多边形的剖分，在这个 n-凸多边形剖分中每个多边形的节点按逆时针排列。和三角剖分类似，对多边形的任一条有向边 AB 和待查点 P，定位算子 $Det(ABP)>0$ 表示搜索路径由这条边进入该多边形，否则表示搜索路径由这条边出，进入和这条边相邻的多边形。

10.4.3 扩展到三维曲面网格

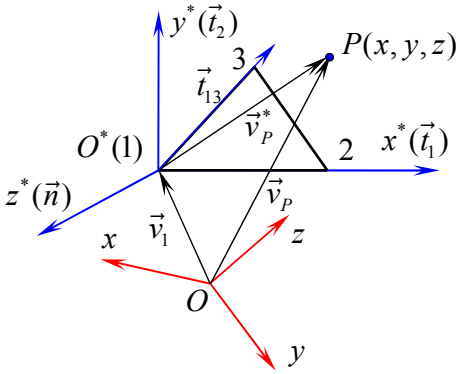


图 10.6 把待查点 P 投影到当前三角形所在的平面上

Fig. 10.6 Projection of the query point P onto current triangle plane

上述方法也可以扩展到三维曲面网格，整个搜索过程不变，需要改动的是主附方向的定义和终止条件。曲面中每个单元的法线方向的定义按照右手规则定义，指向模型外侧。在三维空间中，为了定义搜索方向首先把待查点投影到当前三角形所在的平面上(如图 10.6)，对当前 $\Delta 123$，局部坐标系定义如下：节点 1 位坐标新的原点；$x^*\left(\vec{t}_1\right)$ 轴为 $\vec{t}_1=\overrightarrow{12}$；$z^*(\vec{n})$ 是三角形法线方向，$\vec{n}=\vec{t}_1\times\vec{t}_{13}$；这样 $y^*(\vec{t}_2)$ 定义为 $\vec{t}_2=\vec{n}\times\vec{t}_1$。把点 P 转化到这个局部坐标系下新的坐标为：

$$vp=[T](\overrightarrow{v_p}-\overrightarrow{v_1})$$

$$[T]=\begin{bmatrix} t_{1x} & t_{1y} & t_{1z} \\ t_{2x} & t_{2y} & t_{2z} \\ n_x & n_y & n_z \end{bmatrix} \tag{10.4}$$

10.4.4 收敛性分析

在上述算法中二维空间和三维空间的差异主要是收敛路径定义的不同。在二维情况下，当按照第一主方向的路径不能达到搜索目标时会激活其他主方向和附方向，直到达到搜索目标为止。也就是说本文算法在最坏的情况下是遍历模型中的所有单元，这样就避免了环的形成，也就是说本文算法在 2 维空间中是收敛的。

在三维情况下，算法的收敛性这里用图 10.7 说明。如果曲面的表面比较平坦，搜索路径就比较短；如果曲面中在起始点和目标点之间有凹凸不平的曲率变化，搜索路径就会回溯，形成一个回溯区域。因为上述算法中主方向根据单元和目标点的相对位置，可能会有多个，而第一主方向的选择是随机的，所以回溯区域的大小也是随机的，但是因为算法是可回溯的，所以搜索路径路径会沿着目标点越来越近的方向，直到搜索到目标点为止。

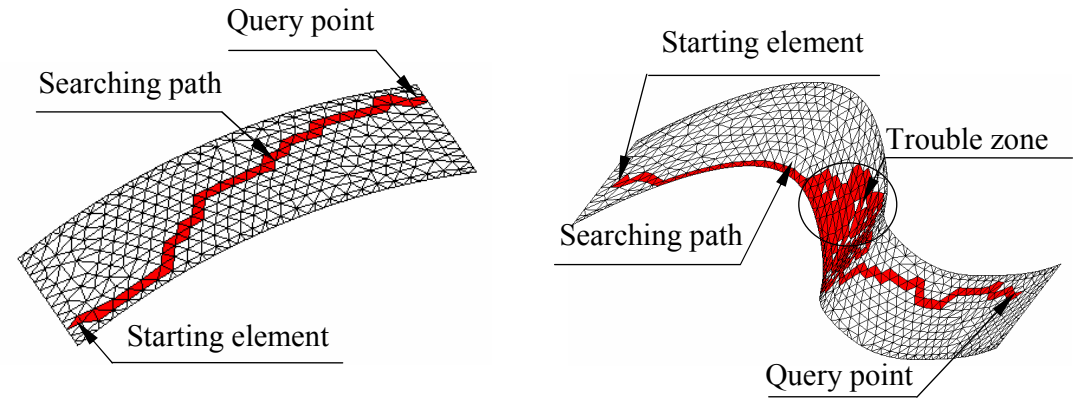


图 10.7 三维空间中的搜索路径
Fig 10.7. Searching path in 3D curve surfaces

.

10.4.5 算法实现

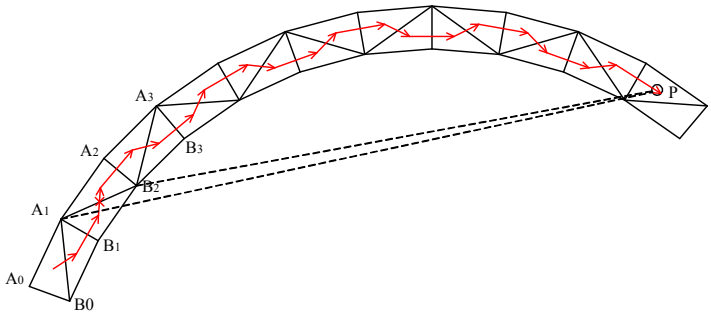


图 10.8 凹域的可回退的点的定位查找算法
Fig. 10.8 Backward search in non convex field

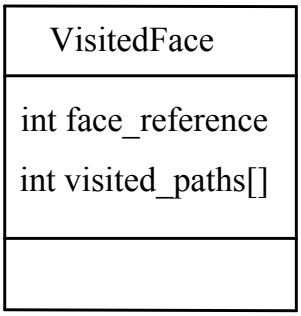


图 10.9 已访问的三角形链表
Fig. 10.9 List of visited faces

如图 10.8 所示，假如待查点为 P，初始三角形是 $\Delta A_0B_0A_1$，那么红色箭头线段是 $\Delta A_0B_0A_1$ 到点 P 是唯一一条路径。但是如果用经典的 CCW 或 BCS 方法，路径在边 B_1B_2 就不能继续进行了，也就是上述两种方法对这个算例都是失效的。为了让算法对类似这样的算例有效，必须使用第二主方向或者附方向(如 $\Delta A_1B_1B_2$ 的边 A_1B_2，这个边在 CCW 或 BCS 方法中是不可用的搜索边)。在本章算法中，用一个堆栈来存储已经搜索的三角形和已经使用过的边。图 10.9 是已使用过的三角形的数据结构，在这里已访问路径(visited_path)初始值为 FALSE(0)，可回退的点的定位查找算法详细步骤如下：

(1) 设点 P 是待查点
(2) 设 $\Lambda(F)$ 是一个网格剖分中所有单元的数组
(3) 设 $S(F)$ 已访问的单元的集合
(4) 设 $S_K(V)$ 已访问的单元堆栈
(5) 从 $\Lambda(F)$ 随机选择一个单元 F_i 作为起始单元
(6) 把 F_i 压入 $S_K(V)$
(7) 当 $S_K(V)$ 非空时
(8) 设布尔变量 C_k 的初值为 FALSE
(9) IF F_i 非活动状态, THEN
(11) $F_i = S_K(V)$ 顶部单元
(12) C_k = TRUE
(13) IF F_i 所有方向被使用, THEN
(14) 删除 $S_K(V)$ 顶部的元素
(15) End if
(16) End if
(17) 把 F_i 放到 $S(F)$ 中
(18) IF $S(F)$ 的大小 = $\Lambda(F)$ 大小，返回 -1
(19) 设整形值 $ic = 0$ 用来作为 F_i 中有向边和点 P 的定位算子为正的个数
(20) 遍历 F_i 中的所有边，对当前边 $E_i(F_i)$
(21) If C_k=TRUE 并且 $E_i(F_i)$ 已经被遍历, 继续遍历下一条边
(22) Else 设 $E_i(F_i)$ 为 F_i 中已经被遍历的边
(23) End if
(24) 设 F_{i+1} 是 F_i 的共享 E_i 的相邻单元
(25) If $E_i(F_i)$ 和 P 的定位算子非正
(26) $F_i = F_{i+1}$ Go to 步骤(7)
(28) Else
(29) 把 F_{i+1} 放到 $S_K(V)$
(30) $ic = ic + 1$

(31) End if
(32) End for
(33) If $ic = F_i$ 边的个数, 返回 F_i，即 F_i 就是要找的单元
(34) Else F_i 为非活动单元
(35) End if
(36) End while

在上述算法的步骤(5)中，默认情况下是从当前三角剖分 $\Lambda(F)$ 中随机选择一个三角形 F_i 作为初始三角形，实际上如果可以这个初始三角形在点 P 附近，就可以大大缩短搜索过程。有两种方法可以迅速找到 P 附近的三角形：第一种是预先在当前剖分中添加一个 $n \times n$ 的栅格，把每个三角形的重心和某个栅格相对应，这样搜索待查点时首先找到所处的栅格，由这个栅格对应的三角形开始搜索；第二种方法是首先用 W. Randolph Franklin[135]的方法找到和待查点 P 最近的一个 $\Lambda(F)$ 中的节点，由这个节点所连接的任何一个三角形作为初始三角形开始搜索。在三维情况下，上述算法的步骤(20)中，对于当前 F_i，首先建立一个局部坐标系，构成 F_i 的节点和待查点 P 首先转换到这个局部坐标系下，然后在这个局部坐标系下进行算法的后续步骤。上述算法的数据结构可以参照本文第 3 部分描述的基于拓扑连接的网格数据结构。

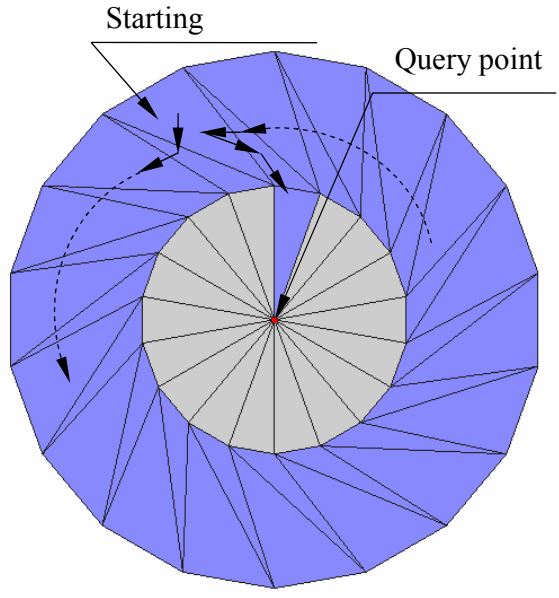


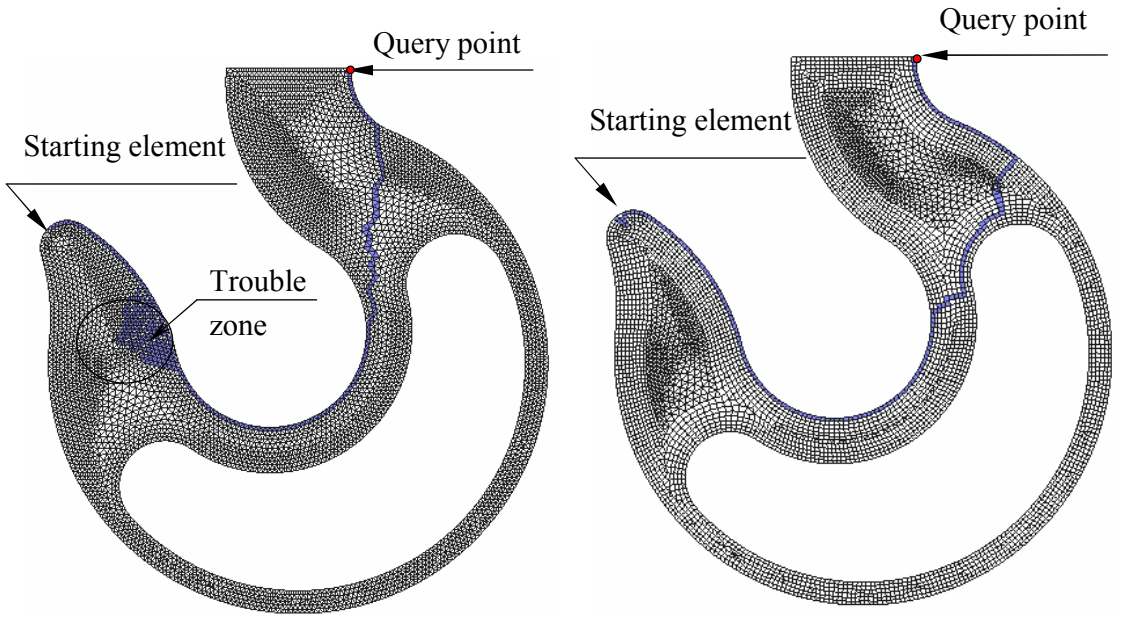
图 10.10 圆形域中点的定位
Fig. 10.10 Point location in a circular field

10.6 数值算例

第一个算例是10.3.2节所论述CCW或BSC方法搜索路径可能形成环的情况。如图10.10所示，在本章论述的算法中，当同一个三角形被第二次使用时，第二主方向或附方向就被激活使用，从而使搜索路径可以跳出环，搜索到待查点。

第二个算例是一个二维的凹域，如图 10.11 所示，目的是测试本算法的健壮性。初始单元特意选在和待查点较远的位置，另外本算例还验证了混合网格的情况(三角形和四边形)，着色部分单元是本算法的搜索路径。由图 10.11a 可以看出为了找到目标点本算法会局部回退从而出现局部冲突区域(trouble zones)，冲突区域的大小和网格本身以及初始单元有关。另外本算法的搜索结果虽然不是一条最短路径，但是由于避免了计算重心坐标，所以查找过程还是很高效的。

图 10.12 是一些三维复杂曲面剖分中点定位例子，在这些例子中，待查点和初始单元都是相距较远的位置，充分验证了本文算法的健壮性。



a) 三角剖分　　b)混合剖分(三角形和四边形单元)

a) Triangle mesh　　b) Mixed mesh (triangle and quadrilateral)

图 10.11 凹域中点的定位

Fig. 10.11 Point location in a complex field

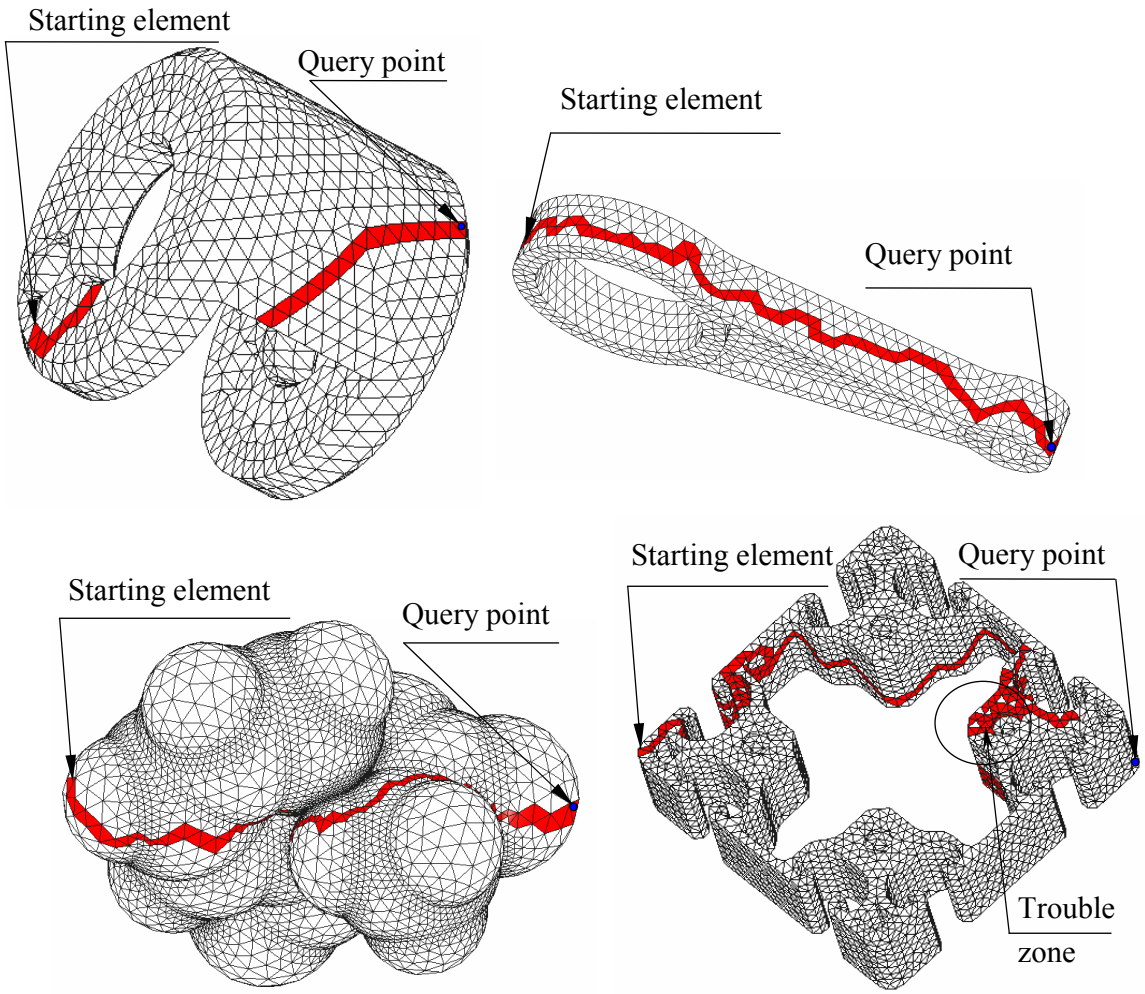


图 10.12 三维复杂域中点的定位

Fig. 10.12 Point location by backward method on 3D complicated surface meshes

结　论

本文主要研究和实现了二维和三维自适应有限元剖分算法，并且介绍和实现了网格剖分程序和多种 CAD 软件通信的 B-Rep 接口、基于拓扑连接的数据结构以及和剖分程序密切相关的点的查找和定位算法，还给出了本文算法在工程项目中的实际应用。大量数值算例表明，本文的自适应有限元网格生成器不但可以反映模型本身的几何特征(如曲率、近亲关系等)，而且可以根据有限元分析结果生成反映诸如应力、应变等特征。本文的研究和算法实现工作主要包括以下几个方面：

(1) 有限元前处理模块和造型软件的公共通信接口及网格生成的总体框架，提出了基于几何和拓扑的网格剖分框架。通过公共通信接口让有限元网格剖分模块几何信息的获取不再和特定的造型软件相关，使造型软件之间模型不同的几何和拓扑描述和表示能够以统一的形式提供剖分程序使用。在这个剖分框架中实现大部分的网格剖分程序所需的公用功能。通过这个框架以统一的形式扩展功能，从而减少近一步研究和开发的费用。

(2) 提出并实现了基于拓扑连接的网格数据结构。该数据结构不但包含了传统的节点和单元信息，同时还包含了连接这些节点和单元的中间信息，简化了网格剖分和优化过程中连接信息的获取及更新操作。该数据结构使信息的完整性与时间/空间效率得到平衡，为有限元网格生成和优化打下坚实基础。

(3) 研究并实现了基于黎曼度量的二维自适应有限元网格生成算法。该部分详细介绍平衡四叉树背景网格的引入以及黎曼度量场的梯度化过程，大大提高了网格生成质量和算法的健壮性。该算法不但能生成各向同性的有限元网格，还可以生成各向异性的有限元网格，从而满足计算流体力学分析的需要。在金属破坏研究的成功运用表明该算法无论在健壮性还是效率上都能满足工程实际的需要。

(4) 研究并实现了三维复杂组合曲面的自适应网格生成算法。详细介绍了把曲线/曲面的几何特征转化转化为有限元网格尺寸信息的方法；平衡八叉树背景网格和三维黎曼度量的引入，将由曲面本身几何特征产生和由外界因素产生的自适应源以统一的形式进行描述，使得二维自适应剖分程序稍加改动就可以用于三维参数曲面的自适应剖分。另外该部分还针对周期性曲面的特殊性，提出了不需要引入虚边界的迁移算法，提高了周期性网格生成程序的健壮性和网格生成的质量。该算法在金属冲压成型研究中的成功应用表明该算法无论在健壮性还是效率上都能满足工程实际的需要。

(5) 改进了三维复杂域四面体剖分的波前推进法(AFT)，背景网格的构造、前沿推进的详细过程以及剖分后单元的优化等。采用 AFT 进行四面体剖分的关键是内核的剖分，本文对内核剖分主要采用回退的方法，回退过程中通过设定不同的前沿优先因子来提高

成功回退的可能性，对于少数不能用回退完成剖分的多面体采用基于线性规划的插点方法进行分解。在四面体单元剖分完成后进行单元的优化以提高网格的质量。算例表明，本文的算法不但能生成高质量的网格，而且整个算法有很好的时间特性。

(6) 提出并实现了四面体-六面体扩展转换模板，可将四面体分解为不同数目、不同密度过渡形式的六面体单元。这样，初始四面体网格不需要划分得很细，生成的六面体单元数量可以通过采用不同规格的扩展转换模板而得到控制。算例表明，本文所提出方法是有效的，生成的网格质量较高。

(7) 改进了传统的在网格剖分算法中广泛使用的基于逆时针法和重心坐标法的点的搜索和定位算法。提出了可回退的点的搜索和定位算法，从而避免了环的形成，使得该算法不但可以适用于单连通凸域的点的定位搜索，而且可以适用于多连通凹域。另外本文还将该算法推广到三维曲面网格的点的搜索和定位中，提高了算法的应用范围。

本文需要进一步解决的问题有：

(1) 几何模型特征的识别和智能修复。曲面网格的自适应生成是网格生成的一个难点，作者的经验表明程序失败的很多原因在几何特征的识别上。在复杂几何模型中很多细小的几何特征会导致生成的网格与用户的需求不相适应，甚至导致剖分失败。因而在网格剖分前几何模型中特征的识别和智能修复是今后进一步努力的一个方向。

(2) 近亲信息的快速准确获取。本文用平衡的八叉树来存储组合曲面中的自适应信息，实践证明这种方法对曲线和曲面的曲率特征信息能够得到很好的近似。然而对于模型中近亲信息，比如如何快速准确地识别复杂模型中邻近的曲线和曲面？作者尝试过文献中给出的方法，但效率较低。本文给出的方法虽然较快，但是内存占用比较多。因此近亲信息的快速准确地获取仍旧需要进一步的研究。

(3) 基于 STL 文件的自适应网格生成被证明具有非常广阔的应用前景，这也是近年来网格生成领域研究的一个方向。如何快速准确的识别 STL 文件中所包含的几何特征；STL 网格到有限元网格的转化；新生成节点的定位等是这一方法需要解决的关键几个问题。

(4) 自适应网格生成算法并行化和分布化。并行化计算环境对于大规模、超大规模科学计算以及高端工程应用是必需的，并行计算环境与分布式计算环境的控制软件日趋成熟，为算法的并行化、分布化开发提供了强有力的技术支持。AFT 方法具有并行化优势，在计算效率、内存管理、生成单元质量等方面有许多潜力可挖。

创新点摘要

本文系统地建立了自适应有限元网格生成基础框架，研究实现了二维和三维复杂域的自适应有限元网格生成方法。基础框架包括：网格生成与 CAD 平台集成的统一几何拓扑接口；网格生成公共几何数值算法工具箱；基于拓扑连接的网格数据结构；自适应网格生成支撑算法库。本文研究开发的部分组件已在有限元分析软件 JIFEX V5 以及与法国兰斯大学材料、力学与结构实验室合作项目中得到应用。

本文研究工作的主要创新点如下：

(1) 系统性地建立了二维/三维自适应有限元网格生成基础框架。

实现了网格生成与 CAD 平台集成的通用几何拓扑接口，使得有限元网格生成算法不再和特定的几何建模平台相关；提出基于拓扑连接与关系矩阵的网格数据结构，简化了网格生成和优化过程中连接信息的获取及更新操作；引入黎曼度量以及平衡四/八叉树，将二维/三维、实体/曲面以及计算/几何自适应网格生成纳入统一的算法与程序架构。(第 2、3、4、6 章)

(2) 改进了三维复杂组合参数曲面的自适应网格生成算法。

提出了周期性曲面网格生成的迁移算法，有效地克服了周期性曲面的“虚边界”难题，提高了整个算法的健壮性和网格生成质量；基于自适应有限元网格生成基础框架，实现了三维复杂组合曲面自适应网格生成的算法，将几何自适应、计算自适应等多种自适应要求以统一的方式进行处理，大大降低了自适应网格生成核心算法的复杂度。(第 4、6 章)

(3) 改进了三维实体网格剖分的波前推进算法。

针对三维实体 AFT 网格剖分算法的收敛性问题，提出了多种前沿优先因子以扩展前沿回退-推进的路径，提高了回退方法的成功几率；对于少数不能用回退方法完成剖分的内核多面体，采用基于线性规划的内核分解方法进行剖分。采用基于拓扑连接的网格数据结构，提高了前沿的查找和更新的效率。(第 8 章)

(4) 改进了传统的基于逆时针法和重心坐标法的点的定位算法。

针对传统的基于逆时针法和重心坐标法的环和凹域问题，提出了可回溯的点的定位算法，避免了环的形成。使得该算法不但可以适用于单连通凸域的点的定位，而且可以适用于三维复杂多连通域，提高了算法的应用范围。(第 10 章)

参 考 文 献

[1] Zienkiewicz, O.C. Achievements and some unsolved problems of the finite element method. International Journal for Numerical Methods in Engineering, 2000, 47, 9-28.
[2] Shewchuk, J.R. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. Discrete and Computational Geometry, 1997, 18(3), 305-363.
[3] Shewchuk, J.R. Lecture Notes on Geometric Robustness. (Department of Electrical Engineering and Computer Sciences University of California at Berkeley Berkeley, CA 94720, 2006).
[4] Schneider, P.J. and Eberly, D.H. Geometric Tools for Computer Graphics. (Morgan Kaufmann, 2003).
[5] Owen, S.J. A survey of unstructured mesh generation technology. 7th International Meshing Roundtable, 1998.
[6] Soni, B.K. Grid generation: Past, present, and future. Applied Numerical Mathematics, 2000, 32(4), 361-369.
[7] George, P.L. and Kaliakin, V.N. Automatic Mesh Generation; Application to Finite Element Methods. Journal of Engineering Mechanics, 1993, 119, 643.
[8] Lo, S.H. Finite Element Mesh Generation and Adaptive Meshing. Structural Engineering and Materials, 2002, 4(4), 219-242.
[9] Yang, H.T.Y., Saigal, S., Masud, A. and Kapania, R.K. A survey of recent shell finite elements. International Journal for Numerical Methods in Engineering, 2000, 47(1), 101-127.
[10] 关振群, 宋超, 顾元宪 and 隋晓峰. 有限元网格生成方法研究的新进展. 计算机辅助设计与图形学学报, 2003, 1, 1-14.
[11] Mackerle, J. 2D and 3D finite element meshing and remeshing. Engineering Computations, 2001, 18(8), 1108-1197.
[12] Beall, M.W., Walsh, J. and Shephard, M.S. Accessing CAD geometry for mesh generation. 12th International Meshing Roundtable, 2003.
[13] Butlin, G. and Stops, C. CAD data repair. 5th International Meshing Roundtable, pp. 7-12 1996).
[14] White, D.R., Saigal, S. and Owen, S.J. Meshing Complexity of Single Part CAD Models. 12th International Meshing Roundtable, 2003.
[15] Tautges, T.J. The Common Geometry Module (CGM): A Generic, Extensible Geometry Interface. 9th International Meshing Roundtable, 2000.
[16] Panthaki, M.J., Sahu, R. and Gerstle, W.H. An Object-Oriented Virtual Geometry Interface. 7th International Meshing Roundtable, 1997.
[17] Gopalsamy, S., Ross, D.H. and Shih, A.M. API for Grid Generation Over Topological Models. l3th International Meshing Roundtable,, 2004.
[18] Haimes, R. and Crawford, C. Unified Geometry Access for Analysis and Design. 12th International Meshing Roundtable, 2003.
[19] 张洪武, 关振群, 李云鹏 and 顾元宪. 有限元分析与 CAE 技术基础. (清华大学出版社 p261-310 2004 年 11 月第一版.
[20] Botsch, M., Steinberg, S., Bischoff, S. and Kobbelt, L. OpenMesh - a generic and efficient polygon mesh data structure. OpenSG Symposium, 2002.

[21] Lee, S.H. and Lee, K. Partial entity structure: a compact non-manifold boundary representation based on partial topological entities. 6th ACM symposium on Solid modeling and applications, 2001, 159-170.
[22] Luo, Y. and Lukacs, G. A boundary representation for form features and non-manifold solid objects. Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, 1991, 45-60.
[23] Garimella, R.V. Mesh data structure selection for mesh generation and FEA applications. International Journal for Numerical Methods in Engineering, 2002, 55(4), 451-478.
[24] Remacle, J.F. and Shephard, M.S. An algorithm oriented mesh database. International Journal for Numerical Methods in Engineering, 2003, 58, 349-374.
[25] Schneiders, R. and Bünten.R. Automatic generation of hexahedral finite element meshes. Computer Aided Geometric Design, 1995, 12(7), 693-707.
[26] Schneiders, R., Schindler, R. and Weiler, F. Octree-based generation of hexahedral element meshes. 5th International Meshing Roundtable, 1996.
[27] Tchon, K.F., Hirsch, C. and Schneiders, R. Octree-Based Hexahedral Mesh Generation For Viscous Flow Simulations. 13th AIAA Computational Fluid Dynamics Conference, AIAA-97-1980. AIAA, June 1997).
[28] McMorris, H. and Kallinderis, Y. Octree-advancing front method for generation of unstructured surface and volume meshes. AIAA Journal, 1997, 35(6), 976-984.
[29] Frey, P.J. and Marechal, L. Fast Adaptive Quadtree Mesh Generation. Proceedings, 7th International Meshing Roundtable, Sandia National Lab, USA, 1998, 211-224.
[30] Watson, D.F. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. The Computer Journal, 1981, 24(2), 167.
[31] Wright, J.P. and Jack, A.G. Aspects of three-dimensional constrained Delaunay meshing. International Journal for Numerical Methods in Engineering, 1994, 37(11), 1841-1861.
[32] D.T, L. and Schacter, B.J. Two Algorithms for Constructing a Delaunay Triangulation. International Journal of Computer and Information Sciences, 1980, 3(9), 219-242.
[33] Baker, T.J. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. Engineering with Computers, 1989, 5(3), 161-175.
[34] Joe, B. Construction of three-dimensional Delaunay triangulations using local transformations. Computer Aided Geometric Design, 1991, 8(2), 123-142.
[35] Chen, H. and Bishop, J. Delaunay Triangulation for Curved Surfaces. 6th International Meshing Roundtable, 1997.
[36] Li, X.-Y. Generating well-shaped d-dimensional Delaunay Meshes Theoretical Computer Science, 2003, 296(1), 145-165.
[37] Borouchaki, H., George, P.L., Hecht, F., Laug, P. and Saltel, E. Delaunay Mesh Generation Governed by Metric Specifcations. Part I. Algorithms. Finite Element Analysis and Design, 1997, 25(1-2), 61-83.
[38] Du, Q. and Wang, D. Boundary recovery for three dimensional conforming Delaunay triangulation. Computer methods in applied mechanics and engineering, 2004, 193, 2547-2563.
[39] Shewchuk, J.R. Delaunay refinement algorithms for triangular mesh generation. Computational Geometry: Theory and Applications, 2002, 22(1-3), 21-74.
[40] Secchi, S. and Simoni, L. An improved procedure for 2D unstructured Delaunay mesh generation. Advances in Engineering Software, 2003, 34(4), 217-234.

[41] Borouchaki, H., Laug, P. and George, P.L. Parametric surface meshing using a combined advancing-front generalized Delaunay approach. International Journal for Numerical Methods in Engineering, 2000, 49, 233-259.
[42] Weatherill, N.P. and Hassan, O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. International Journal for Numerical Methods in Engineering, 1994, 37(12), 2005-2039.
[43] Shewchuk, J.R. Delaunay Refinement Mesh Generation. (School of Computer Science, Carnegie Mellon University, 1997).
[44] Shewchuk, J.R. Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery. 11th International Meshing Roundtable (Ithaca, New York), 2004.
[45] Weatherhill, N.P. The integrity of geometrical boundaries in the two-dimensional Delaunay triangulation. Commun. Appl. Numer Meth, 1990, 6, 101-109.
[46] Borouchaki, H. and George, P.L. Aspects of 2-d Delaunay mesh generation. International Journal for Numerical Methods in Engineering, 1997, 40(11), 1957-1975.
[47] George, P.L., Hecht, F. and Saltel, E. Automatic mesh generator with specified boundary. Computer methods in applied mechanics and engineering, 1991, 92(3), 269-288.
[48] Xiangyang, L. Sliver-Free Three Dimensional Delaunay Mesh Generation. (University of Illinois, 2000).
[49] C.K.Lee. Automatic adaptive mesh generation using metric advancing front approach. Engineering Computations, 1999, 16(2), 230-263.
[50] C.K.Lee and R.E.Hobbs. Automatic Adaptive Finite Element Mesh Generation over Arbitrary Two-dimensional Domain using Advancing Front Technique. Computers & Structures, 1999, 71(1), 9-34.
[51] Lau, T.S. and Lo, S.H. Finite Element Mesh Generation Over Analytical Surfaces. Computers and Structures, 1996, 59(2), 301-309.
[52] C.K.Lee and R.E.Hobbs. Automatic Adaptive Finite Element Mesh Generation over Rational B-spline Surfaces. Computers & Structures, 1998, 69(5), 577-608.
[53] R.Tristano, J., Owen, S.J. and Canann, S.A. Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition. Proceedings of the 7th International Meshing Roundtable, 1998.
[54] Nakahashi, K. and Sharov, D. Direct Surface Triangulation using the Advancing Front Method. AIAA Computational Fluid Dynamics Conference, 12th, 1995.
[55] Moller, P. and Hansbo, P. On advancing front mesh generation in three dimensions. International Journal for Numerical Methods in Engineering, 1995, 38(21), 3551-3569.
[56] C.K.Lee. Automatic Metric Advancing Front Triangulation over Curved Surfaces. Engineering Computations, 2000, 17, 48-74.
[57] Ito, Y., Shih, A.M. and Soni, B.K. Reliable isotropic tetrahedral mesh generation based on an advancing frontmethod. Proceedings of the 13th International Meshing Roundtable, 2004.
[58] Seveno, E. Towards an adaptive advancing front method. Proceeding, 6th International Meshing Roundtable, 1997.
[59] Yamakawa, S. and Shimada, K. Anisotropic tetrahedral meshing via bubble packing and advancing front. International Journal for Numerical Methods in Engineering, 2003, 57(13), 1923-1942.

[60] Wang, D., Hassan, O., Morgan, K. and Weatherill, N. Enhanced remeshing from STL files with applications to surface grid generation. COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING, 2006.
[61] Peiró, J., Formaggia, L., Gazzola, M., Radaelli, A. and Rigamonti, V. Shape reconstruction from medical images and quality mesh generation via implicit surfaces. International Journal for Numerical Methods in Fluids, 2006.
[62] Ito, Y., Shum, P.C., Shih, A.M., Soni, B.K. and Nakahashi, K. Robust generation of high-quality unstructured meshes on realistic biomedical geometry. International Journal for Numerical Methods in Engineering, 2006, 65(6), 943-973.
[63] Frey, P.J. Generation and adaptation of computational surface meshes from discrete anatomical data. International Journal for Numerical Methods in Engineering, 2004, 60(6), 1049-1074.
[64] Ito, Y. and Nakahashi, K. Surface triangulation for polygonal models based on CAD data. International Journal for Numerical Methods in Fluids, 2002, 39, 75-96.
[65] Bechet, E., Cuilliere, J.C. and F.Trochu. Generation of a finite element MESH from strtrolithography (STL) files. Computer-Aided Design, 2002, 34, 1-17.
[66] 关振群, 单菊林 and 顾元宪. 基于黎曼度量的复杂参数曲面有限元网格生成方法. 计算机学报, 2006, 29(10), 1423-1433.
[67] Zhenqun, G., Xiaofeng, S., Yuanxian, G. and Yunpeng., L. Automatic finite element mesh generation over 3D combined surfaces. Chinese Journal of Computational Mechanics, 2003, 20(4), 409-416.
[68] Cuilliere, J.C. An Adaptive Method for the Automatic Triangulation of 3D Parametric Surfaces Computer-Aided Design, 1998, 30(2), 139-149.
[69] Rivara, M.C. and Inostroza, P. Using longest-side bisection techniques for the automatic refinement of delaunay triangulations. International Journal for Numerical Methods in Engineering, 1997, 40(4), 581-597.
[70] Rivara, M.C. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. International Journal for Numerical Methods in Engineering, 1997, 40(18), 3313-3324.
[71] Chen, Z., Tristano, J.R. and Kwok, W. Combined Laplacian and optimization-based smoothing for quadratic mixed surface meshes. 12th International Meshing Roundtable, 2003.
[72] Garimella, R.V., Shashkov, M.J. and Knupp, P.M. Optimization of surface mesh quality using local parameterization. Proceedings of the 11th International Meshing Roundtable, 2002.
[73] Hyun, S. and Lindgren, L.-E. Smoothing and adaptive remeshing schemes for graded element. International Journal for Numerical Methods in Engineering, 2001, 17, 1-17.
[74] Zhou, T. and Shimada, K. An angle-based approach to two-dimensional mesh smoothing. Proceedings of the 9th International Meshing Roundtable, 2000.
[75] Knupp, P.M. Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part i-a framework for surface mesh optimization. International Journal for Numerical Methods in Engineering, 2000, 48, 401-420.
[76] Knupp, P.M. Matrix Norms and the Condition Number: A General Framework to Improve Mesh Quality via Node-Movement. 8th International Meshing Roundtable, South Lake Tahoe, CA (US), 10/10/1999--10/13/1999, 1999.

[77] Canann, S.A., Tristano, J.R. and Staten, M.L. An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes. 7th International Meshing Roundtable, 1998.
[78] J.Frey, P. About Surface Remeshing. Proceeding of 9th International Meshing Roundtable, 2000.
[79] Walton, D.J. and Meek, D.S. A triangular G^1 patch from boundary curves. Computer-Aided Design, 1996, 28(2), 113-123.
[80] Xue, D., Demkowicz, L. and Bajaj, C. Reconstruction of G 1 surfaces with biquartic patches for hp fe simulations. 13th International Meshing Roundtable, pp. 323-332 2004).
[81] Max, N. Weights for Computing Vertex Normals from Facet Normals. Journal of Graphics Tools, 1999, 4(2), 1-6.
[82] Floater, M.S. and Hormann, K. Surface parameterization: a tutorial and survey. Advances in Multiresolution for Geometric Modelling, 2005, 1.
[83] Laug, P. and Borouchaki, H. Curve linearization and discretization for meshing composite parametric surfaces. COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING, 2004, 20(11), 869-876.
[84] Surazhsky, V., Alliez, P. and Gotsman, C. Isotropic remeshing of surfaces: a local parameterization approach. 12th International Meshing Roundtable, 2003.
[85] Lau, T.S., Lo, S.H. and Lee, C.K. Generation of Quadrilateral Mesh over Analytical Curved Surfaces. Finite Elements in Analysis and Design, 1997, 27(3), 251-272.
[86] 宋超. 非结构化自适应有限元网格生成的 AFT 方法. (大连理工大学博士学位论文, 2004 年 8 月).
[87] Blacker, T. Automated Conformal Hexahedral Meshing Constraints, Challenges and Opportunities. Engineering with Computers, 2001, 17(3), 201-210.
[88] Blacker, T. The Cooper Tool. Proceedings of the 5th International Meshing Roundtable, 1996, 13-29.
[89] Li, T.S., McKeag, R.M. and Armstrong, C.G. Hexahedral meshing using midpoint subdivision and integer programming. Computer methods in applied mechanics and engineering, 1995, 124(1/2), 171-193.
[90] Price, M.A. and Armstrong, C.G. Hexahedral Mesh Generation by Medial Surface Subdivision: Part ii. Solids With Flat and Concave Edges. International Journal for Numerical Methods in Engineering, 1997, 40(1), 111-136.
[91] Blacker, T.D. and Meyers, R.J. Seams and wedges in plastering: A 3-D hexahedral mesh generation algorithm. Engineering with Computers, 1993, 9(2), 83-93.
[92] Tautges, T.J., Blacker, T. and Mitchell, S.A. The Whisker Weaving Algorithm: A Connectivity-Based Method For Constructing All-Hexahedral Finite Element Meshes. International Journal for Numerical Methods in Engineering, 1996, 39(19), 3327-3349.
[93] Remacle, J.F., Li, X., Shephard, M.S. and Flaherty, J.E. Anisotropic Adaptive Simulation of Transient Flows using Discontinuous Galerkin Methods. International Journal for Numerical Methods in Engineering, 2005, 62, 899-923.
[94] Jansen, K.E., Shephard, M.S. and Beall, M.W. On Anisotropic Mesh Generation and Quality Control in Complex Flow Problems. 10th International Meshing Roundtable, pp. 111-134 2001).

[95] Frey, W.H. Selective Refinement:: a New Strategy For Automatic Node Placement In Graded triangular meshes. International Journal for Numerical Methods in Engineering, 1987, 24(11), 2183-2200.
[96] Erhart, T., Wall, W.A. and Ramm, E. Robust adaptive remeshing strategy for large deformation,transient impact simulations. International Journal for Numerical Methods in Engineering, 2006, 65, 2139-2166.
[97] Borouchaki, H., Laug, P., Cherouat, A. and Saanouni, K. Adaptive remeshing in large plastic strain with damage. International Journal for Numerical Methods in Engineering, 2005, 63(1), 1-36.
[98] Li, X., Remacle, J.F., Chevaugeon, N. and Shephard, M.S. Anisotropic mesh gradation control. 13th International Meshing Roundtable, 2004.
[99] Lo, S.H. 3D anisotropic mesh refinement in compliance with a general metric specification. Finite Elements in Analysis and Design, 2001, 38(1), 3-19.
[100] Borouchaki, H., Hecht, F. and Frey, P.J. Mesh Gradation Control. International Journal for Numerical Methods in Engineering, 1998, 43(6), 1143-1165.
[101] Shimada, K., Yamada, A. and Itoh, T. Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles. Proceedings of the 6th International Meshing Roundtable, 1997.
[102] Borouchaki, H., George, P.L. and Mohammadi, B. Delaunay mesh generation governed by metric specifications. Part II. applications. Finite Elements in Analysis and Design, 1997, 25(1-2), 85-109.
[103] Bossen, F.J. and Heckbert, P.S. A Pliant Method for Anisotropic Mesh Generation. 5th Intl. Meshing Roundtable, 1996, 63-74.
[104] Castro-Diaz, M.J., Hecht, F. and Mohammadi, B. New Progress in Anisotropic Grid Adaptation for Inviscid and Viscous Flows Simulations. Proceedings of the 4th International Meshing Roundtable, 1995.
[105] Cunha, A., Canann, S. and Saigal, S. Automatic Boundary Sizing for 2D and 3D Meshes. Trends in Unstructured Mesh Generation, ASME, 1997.
[106] Zhu, J., Blacker, T. and Smith, R. Background overlay grid size functions. Proceedings of the 11th International Meshing Roundtable, pp. 65-74 2002).
[107] C.K.Lee. On curvature element-size control in metric surface mesh generation. International Journal for Numerical Methods in Engineering, 2001, 50(4), 787-807.
[108] Kunert, G. Toward anisotropic mesh construction and error estimation in the finite element method. Numerical Methods for Partial Differential Equations, 2002, 18(5), 625-648.
[109] Zienkkiewicz, O.C. and Zhu, J.Z. Adaptivity and mesh generation. International Journal for Numerical Methods in Engineering, 1991, 32(4), 783-810.
[110] Peraire, J., Peiro, J. and Morgan, K. Adaptive remeshing for three-dimensional compressible flow computations. Journal of Computational Physics, 1992, 103(2), 269-285.
[111] Zhu, J.Z. and Zienkiewicz, O.C. A posteriori error estimation and three-dimensional automatic mesh generation. Finite Elements in Analysis and Design, 1997, 25(1-2), 167-184.
[112] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design patterns: elements of reusable object-oriented software. (Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995).
[113] Freeman, E., Freeman, E., Sierra, K. and Bates, B. Head First Design Patterns. (O'Reilly, 2004).

[114] Martin, R.C. Agile Software Development: Principles, Patterns, and Practices. (Prentice Hall PTR Upper Saddle River, NJ, USA, 2003).
[115] Josuttis, N.M. The C++ Standard Library: a Tutorial and Reference. (Addison-Wesley, 1999).
[116] Peric, D., Vaz, M. and Owen, D.R.J. On adaptive strategies for large deformations of elasto-plastic solids at finite strains: computational issues and industrial applications. Computer methods in applied mechanics and engineering, 1999, 176(1), 279-312.
[117] Hartmann, E. On the curvature of curves and surfaces defined by normalforms. Computer Aided Geometric Design, 1999, 16(5), 355-376.
[118] Borouchaki, H. and Frey, P.J. Adaptive triangular-quadrilateral mesh generation. International Journal for Numerical Methods in Engineering, 1998, 41(5), 915-934.
[119] Jones, M.T. and Plassmann, P.E. Adaptive refinement of unstructured finite-element meshes. Finite Elements in Analysis and Design, 1997, 25(1), 41-60.
[120] Lee, C.K. Automatic adaptive mesh generation using metric advancing front approach. Engineering Computations, 1999, 16(2), 230-263.
[121] Shewchuk, J.R. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. Proceedings of the 11th International Meshing Roundtable, 2002.
[122] P.Lestriez, F.Bogard, J.L.Shan and Y.Q.Guo. Fatigue damage modeling on rolling path under cyclic loading. 12th IFToMM Word Congress Besancon, France, 2007).
[123] Bartuschka, U., Mehlhorn, K. and Naher, S. A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection problem. Proc. Workshop on Algorithm Engineering, 1997, 124-135.
[124] do Carmo, M.P. Differential geometry of curves and surfaces. (Prentice-Hall Englewood Cliffs, NJ, 1976).
[125] 施法中. 计算机辅助几何设计与非均匀有理 B 样条. (高等教育出版社, 2001.8).
[126] ROGER J CASS and BENZLEY, S.E. Generalized 3-D Paving: An Automated Quadrilateral Surface Mesh Generation Algorithm. International Journal for Numerical Methods in Engineering, 1996, 39, 1475-1489.
[127] Guo, Y.Q., Li, Y.M., Bogard, F. and Debray, K. An efficient pseudo-inverse approach for damage modeling in the sheet forming process. Journal of Materials Processing Tech., 2004, 151(1-3), 88-97.
[128] Naceur, H., Guo, Y.Q. and Batoz, J.L. Blank optimization in sheet metal forming using an evolutionary algorithm. Journal of Materials Processing Tech., 2004, 151(1-3), 183-191.
[129] Guo, Y.Q., Batoz, J.L., Naceur, H., Bouabdallah, S., Mercier, F. and Barlet, O. Recent developments on the analysis and optimum design of sheet metal forming parts using a simplified inverse approach. Computers and Structures, 2000, 78(1-3), 133-148.
[130] Piegl, L.A. and Tiller, W. The Nurbs Book. (Springer, 1997).
[131] Ito, Y. and Nakahashi, K. Improvements in the reliability and quality of unstructured hybrid mesh generation. International Journal for Numerical Methods in Fluids, 2004, 45(1), 79-108.
[132] Owen, S.J., Staten, M.L., Canann, S.A. and Saigal, S. Q-Morph: an indirect approach to advancing front quad meshing. International Journal for Numerical Methods in Engineering, 1999, 44, 1317-1340.

[133] Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R. and Wu, A.Y. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. Journal of the ACM, 1998, 45(6), 891-923.
[134] Murphy, M. and Skiena, S.S. Ranger: a tool for nearest neighbor search in high dimensions. Proceedings of the ninth annual symposium on Computational geometry, 1993, 403-404.
[135] Franklin, W.R. Nearest Point Query on 184,088,599 Points in E3 with a Uniform Grid. p. http://www.ecse.rpi.edu/Homepages/wrf/Research/nearpt3/.
[136] Cignoni, P., Montani, C. and Scopigno, R. DeWall: a fast divide and conquer Delaunay triangulation algorithm in Ed. Computer-Aided Design, 1998, 30(5), 333-341.
[137] Devroye, L. A Note on Point Location in Delaunay Triangulations of Random Points. Algorithmica, 1998, 22(4), 477-482.
[138] Guibas, L. and Stolfi, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi. ACM Transactions on Graphics (TOG), 1985, 4(2), 74-123.
[139] Sundareswara, R. and Schrater, P. Extensible point location algorithm. Geometric Modeling and Graphics, 2003. Proceedings. 2003 International Conference on, 2003, 84-89.

攻读博士学位期间发表学术论文情况

[1]. **J.L.Shan**, Y.M. Li, Y.Q. Guo, Z. Q. Guan A Robust Backward Search Method based on Walk-through for Point Location on 3D Surface Mesh. *International Journal for Numerical Methods in Engineering*, Accepted.

[2]. 关振群, **单菊林**, 顾元宪. 基于黎曼度量的复杂参数曲面有限元网格生成方法.计算机学报. 2006:29(10), 1423-1433.

[3]. 关振群, **单菊林**, 顾元宪. 基于转换模板的三维实体全六面体网格生成方法. 计算力学学报, Vol.11, No.1 2005 年 2 月

[4]. **单菊林**, 关振群, 宋超, 顾元宪. 一个高效可靠的三维 AFT 四面体网格生成算法. 计算力学学报(已接受)

[5]. 关振群, **单菊林**, 顾元宪. 面向对象的有限元模型在 CAD 平台上的应用 吉林大学学报(工学版), Vol.33, No.8, 2003 年 9 月.

[6]. M.Dong, K. Debray, **J.L.Shan**, Y Q Guo. *Design and Optimization of Addendum Surfaces in Sheet Metal Forming Process*. Mechanics of Advanced Materials and Structures. Accepted

[7]. **J.L.Shan,** Z. Q. Guan, Y. X. Gu. Common Geometry Interface for Mesh Generation(CGIM). COMPUTATIONAL MECHANICS WCCM VI in conjunction with APCOM'04, Beijing, China 2004, Tsinghua University Press & Springer-Verla.

[8]. Z. Q. Guan, **J.L.Shan**, Y. Zheng, Y. X. Gu. An Extended Advancing Front Technique for Closed Surface Mesh Generation. *International Journal for Numerical Methods in Engineering*. Sumitted

[9]. P.Lestriez, F.Bogard, **J.L.Shan**, Y.Q.Guo. Fatigue damage modeling on rolling path under cyclic loading. 12th IFToMM Word Congress, Besançon, France June 18-21 2007 Sumitted

[10]. Y.M. Li, J.L. Shan*, Y.Q. Guo. Rebuilding of surfaces starting from the grid of "false tools" in the simulation of stamping and elastic return. 8e Colloque National En Calcul Des Structures 21-25 Mai 2007 Giens

致　　谢

硕博连读将近六年来，曾得到很多老师、同学的关心和帮助，谨在本论文即将完成之际，向他们表达诚挚的谢意！

本文工作是在导师顾元宪教授、张洪武教授、关振群教授和法国兰斯大学的郭英乔教授合作指导下完成的。

首先感谢已故的导师顾元宪教授。导师渊博的学术知识，灵活的思维方式，严谨的治学态度以及忘我的工作精神给予作者极大的鼓舞和鞭策，并将成为作者受益终身的精神财富。在此向导师顾教授致以崇高的敬意和深切地缅怀。

感谢导师张洪武教授和关振群教授。关教授把我带入了有限元网格生成的研究领域，他在论文选题，方法构思，程序设计以及论文撰写等诸多方面为作者倾注了大量的精力。关教授多年来对作者工作上高屋建瓴的指导和生活上至理至情的关怀是作者克服困难，坚持到底的不竭动力，是作者求学道路上的良师益友。

感谢法国兰斯大学的导师郭英乔教授，郭教授对学术的钻研精神和执着态度是作者学习的榜样。和郭教授的交流中，得到郭教授很多的指导和启发，在此向郭教授致以诚挚的感谢。

在此也要特别感谢法国兰斯大学的讲师李昱明和 Philippe LESTRIEZ。李老师在本文算法实现以及作者的英文写作方面给以很多实质性帮助和指导。李老师广博的知识和对工作的一丝不苟精神给作者留下了深刻的印象。和 Philippe 老师的合作交流使得本文的算法变得更将健壮，Philippe 老师对工作的敬业精神也是作者学习的榜样。

感谢课题组郭旭教授，李云鹏副教授，赵国忠副教授，陈飚松副教授，亢战副教授，他们在作者求学道路上给予的鼓励和帮助。特别感谢戴磊老师，已毕业的宋超博士，他们与作者进行了有益讨论并给予作者的无私帮助，感谢他们多年来在程序设计方面对作者给予的指导和建议。

感谢工程力学系陈浩然教授，林家浩教授，李锡夔教授，王希诚教授，杨海天教授，赵广玲老师和陈志娟老师给予作者的指导、关心和帮助。

感谢教研室牛聪民，张盛，陈钢，王辉，于文会、刘晓东，王剑，李宁，张虹，廖爱华，曲晓峰，戴磊等诸位博士和硕士研究生与作者进行的开诚布公的讨论和精诚团结的合作，感谢他们对作者给予的真诚帮助和热情鼓励。

衷心感谢父母双亲多年的养育之恩，他们无私的奉献和无尽的关爱使作者感受到爱的崇高和家的温暖。感谢挚爱的妻子顾航，无怨无悔地陪伴着作者走过上下求索的求学之路。感谢所有关心和帮助作者的师长、亲人和朋友们。

大连理工大学学位论文版权使用授权书

本学位论文作者及指导教师完全了解“大连理工大学硕士、博士学位论文版权使用规定”，同意大连理工大学保留并向国家有关部门或机构送交学位论文的复印件和电子版，允许论文被查阅和借阅。本人授权大连理工大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，也可采用影印、缩印或扫描等复制手段保存和汇编学位论文。

作者签名：_____

导师签名：_____

_____年____月____日