MESH MODIFICATION PROCEDURES FOR

GENERAL 3D NON-MANIFOLD DOMAINS

by

Xiangrong Li

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Mechanical Engineering

Approved by the
Examining Committee:

_____
Dr. Mark S. Shephard, Thesis Advisor

_____
Dr. Joseph E. Flaherty, Member

_____
Dr. Kenneth E. Jansen, Member

_____
Dr. Suvranu De, Member

_____
Dr. Mark W. Beall, Member

_____
Dr. Jean-Francois Remacle, Member

Rensselaer Polytechnic Institute
Troy, New York

July 2003
(For graduation August 2003)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Mesh Size field Representation

$\boldsymbol{e}_i$	unit vector representing the *i*-th direction.
P	a point.
$\boldsymbol{T}(P)$	the mesh metric tensor at point P.
$\boldsymbol{Q}(P)$	the transformation representation of $\boldsymbol{T}(P)$.
$h(P,\boldsymbol{e}_i)$	the desired mesh edge length at point P along direction $\boldsymbol{e}_i$.
x_i	the *i*-th component of local coordinates in physical space.
x'_i	the *i*-th component of coordinates in transformed space.

Domain Representation

M_i^d	the *i*-th mesh entity of dimension d, d=0 for a vertex, d=1 for an edge, d=2 for a face and d=3 for a region.
G_i^d	the *i*-th entity of dimension d in geometry model.
$\sqsubset$	classification symbol used to indicate the association of one or more mesh entities with an entity in geometry model [8].
$\{M^d\}$	unordered group of topological mesh entities of dimension d.
$M_i^d\{M^D\}$	all mesh entities of dimension D adjacent to M_i^d.

ACKNOWLEDGMENT

I would first like to thank my advisor, Prof. Mark S. Shephard, for his support and insightful guidance throughout my doctoral research work. Mark has taught me the importance of quality in research and the necessity of simple but precise presentation of technical work. I also thank Mark for his quick and valuable inputs that improve the overall presentation in going through the drafts of this thesis.

I thank the members of my thesis examination committee for their time and effort. I am grateful to Prof. Kenneth E. Jansen for his invaluable technical advice and providing me the flow simulation environment based on stabilized finite element method. Ken also reminded me the main purpose of meshing procedure is to perform simulation, and showed me the desired characteristics of anisotropic meshes for CFD. I thank Prof. Joseph E. Flaherty and Prof. Jean-Francois Remacle for providing the simulation environment based on discontinuous Galerkin method. Many fruitful discussions with Jean-Francois helped take this research work to a new level of excellence by solving many interesting transient flow problems using anisotropic adaptive meshes.

I thank Dr. Mark W. Beall, president of Simmetrix Inc., for financial support and sharing his ideas. Mark has educated me about meshing techniques, provided his insight to address the curved geometry issue in this thesis, and given me the chance to work in Simmetrix to commercialize this research work. I also appreciate the many interactions I had with Simmetrix colleagues, Robert M. O'bara, Ravi Sakalikar and Saurabh Tendulkar.

The staff and my student colleagues at the Scientific Computation Research Center provided a helpful and stimulating environment. I take this opportunity to express my gratitude to all their contributions. In particular, I would like to thank Rao V. Garimella, Nicolas Chevaugeon, Dibyendu Datta and Sunitha Nagras.

I thank my lovely wife, Yun Li, for being here to share my life with. Finally, I would not be what I am today without the love, support and understanding of all my family members. I shall be eternally grateful to all of them.

ABSTRACT

The application of finite element techniques requires the ability to alter the shape and size of the elements of a given mesh, commonly referred to as *h*-version mesh adaptation, to align with the shape and size distribution as indicated by an error estimation / indication procedure as well as 3-D curved geometries. An effective procedure for anisotropic tetrahedral mesh adaptation for general 3-D geometries based on local mesh modifications has been developed to address this issue.

A mesh metric field has been used to represent the element shape and size distribution. Definition, significance of the mesh metric field and the conformity criteria between the mesh and the mesh metric field are given. At low level, the mesh adaptation procedure operates as a sequence of mesh modification operators ordered to make the mesh conform to a given mesh metric field.

The mesh modification procedure consists of four related high level components: mesh refinement, mesh coarsening, projecting boundary vertices onto curved geometry and element shape correction. The mesh is efficiently aligned to the mesh metric field by incremental refinement and coarsening based on edge length analysis with respect to the mesh metric field. Several techniques have been developed to ensure effective alignment to the mesh metric field, for example, simultaneous split of a set of longest mesh edges, collapsing the shortest mesh edge every other vertex, diagonal edge selection in case of ambiguity and the use of lookup tables. The curved geometry approximation issue is addressed by the procedure of projecting boundary vertices onto geometry. The procedure first effectively projects as many vertices as possible using local mesh modifications. The possible remaining un-projected vertices after mesh modifications can be dealt with by a generalized local cavity re-meshing procedure. The element shape correction procedure improves the alignment to mesh metric field by eliminating sliver elements. Element shape analysis techniques guide the effective determination of the best mesh modification.

Two approaches of adaptively specifying mesh metric field are provided, one using second derivatives and the other using both second derivative and gradient information. Both have been integrated with the mesh adaptation procedure for anisotropic adaptive simulation on 3-D general geometries. Results of adaptivity show the effectiveness and the alignment of adapted meshes to both geometry and mesh metric field.

CHAPTER 1

Introduction and Historical Review

1.1 Background

Computer aided design and simulation methods have been an important tool for industry and scientific research for a wide range of problems. Three dimensional geometric modeler systems, for example Parasolid^{TM}[28], ACIS^{TM}[82], Pro/Engineer^{TM}[68], have matured and can provide complex geometry definitions from a single part to the whole product assembly. In general, the geometric representation is curved and non-manifold[1]. Simulation systems add attributes onto the geometry domains and perform various analysis for stress, heat transfer, vibration, acoustic, or fluid flow as governed by partial differential equations that mathematically describe the physical phenomenon.

The finite element method is a powerful tool for solving partial differential equations on complex geometry domains. Its mathematical properties have been well established, involving a discretization process of the geometry domain and the construction of a piecewise interpolated approximation [36,86]. Accuracy of finite element analysis depends on the domain discretization and the piecewise interpolation. There are theoretically rigorous error analysis techniques to estimate the error in the numerical solution in several norms [1,94]. The procedures of adaptivity are designed to modify the discretization or interpolation as indicated by error estimation/indication procedures to reduce the errors with improved efficiency [5,21,63,86]. Two versions of adaptivity and their combination are commonly used: *h*-method and *p*-method, where *h*-method alters the size of elements and *p*-method changes the interpolations. This thesis is concerned with issues related to the *h*-method.

When *h*-adaptivity methods are applied to practical three dimensional curved non-manifold model and unstructured tetrahedral meshes (the discretization of geometry domain), three types of mesh adaptation techniques indicated below arise:

- Adaptive re-meshing methods [66,35,2];
- Element subdivision methods [3,7,9,11,12,41,54,55,75,83];
- Fixed order mesh modification procedures [19,20,23,25,38,65].

Adaptive re-meshing methods obtain adaptive meshes by regenerating the entire mesh through automatic mesh generation algorithms governed by desired element size and shape infor-

1. Simply speaking, non-manifold models consist of general combinations of solids, surfaces and wires. For a rigorous definition and examples see references [59,89].

mation (see for example [66,35,2]). For example, Peraire *at al.* and Moller *at al.* have used adaptive re-meshing based on anisotropic advancing front mesh generation technology [66,62] and P. L. George applied adaptive re-meshing in terms of anisotropic Delaunay kernel [35]. The benefit of adaptive re-meshing is that curved domains are considered by the mesh generation algorithm and mesh coarsening is unnecessary. However, it is inefficient for refining only a few elements in an already refined mesh, and introduces complexities in the transfer of solution fields from one mesh to another.

Element subdivision methods can control element shape by specific split orders. Rivara has described an approach based on longest-edge refinement algorithm for general triangulations [74,75,76]. Provided an initial good quality triangulation, her algorithm creates good-quality meshes with linear time complexity; Liu and Joe have proposed a successive bisection procedure that controls split orders in terms of an affine transformation, maintaining element quality as well if given an initial good quality mesh [53,54,55]; There are also algorithms such as red-green algorithm [9,12,41] and newest vertex bisection algorithm [60] that can control element shape. However, although effective, these methods tend to over refine and do not consider curved geometry domains and the generalization of changing elements into desired shape. The coarsening processes of these algorithms simply undo the refinement [11,41,74,83], therefore, can't coarsen past the initial mesh.

The third mesh adaptation techniques apply local mesh modifications[1] in fixed order without analysis of situations. For example, Briere *at al.* [25], De Cougny *at al.*[23], and Joe *at al.* [38,39] have improved the quality of an existing mesh using procedures consisting of four local mesh modification operations: swap, collapse, split and relocation. Buscaglia *at al.* [19], Castro-Diaz *at al.* [20] and Pain *at al.* [65] have made the applications of these four local mesh modification operations governed by a desired mesh size and shape distribution for adaptivity purpose in two dimension [19,20] and recently in three dimension [65]. The advantage of the third method is that it is a local process in which collapse-based mesh coarsening can be applied [23], and the transfer of solution fields can be executed incrementally as each modification is applied. However, the issue of accounting for curved geometry domains in refinement is not addressed yet, and the effectiveness of local mesh modification procedure in terms of efficiency and quality is a strong function of the manner in which various mesh modification operations are considered.

Curved domains require that the adaptivity procedure handles the geometric approximation issues [23,40]. In particular, in refining a mesh, new nodes created on curved boundary need

1. Refer to Chapter 4 or references [25,23] for specific mesh modification operations.

to be placed on that boundary. Otherwise, the correct problem will not be solved. However, the simple projection of vertices to the boundaries may produce invalid or unacceptable elements for a number of these new boundary vertices, especially when the local mesh is coarse with respect to the local curvatures of the geometry boundaries.

The desired element shape is strongly influenced by the details of the solution field which is constantly evolving in transient problems [37,71]. Although equilateral elements are good where solution field is isotropic, many physical problems exhibit strong anisotropic phenomena, which can be extremely strong in cases such as high Reynolds number flow problems and phase change problems. The use of isotropic elements in these cases will force the creation of small elements in all directions, therefore leading to prohibitively large meshes. Adaptively constructing anisotropic meshes requires a procedure capable of effectively creating and maintaining control of anisotropic elements that satisfy both element size and shape distribution as indicated by error estimation / indication procedures.

1.2 Motivation

The goal of this thesis is to develop the effective tools and algorithms for *h*-version adaptivity on three dimensional complex geometric domains and capable of handling anisotropic elements. Particularly, given:

- any tetrahedral mesh;
- general 3-D non-manifold domain the given mesh is classified[1] onto;
- a desired element size and shape distribution throughout the domain;

develop an effective mesh modification procedure to adaptively modify the given mesh until the given element size and shape distribution is satisfied with curved domains being properly dealt with.

1.3 Organization

The thesis consists of four related parts: desired element size and shape definition and implementation (Chapter 2 and Chapter 3); mesh modification tools (Chapter 4); mesh modification procedures (Chapter 5 and Chapter 6); and application in adaptive simulations (Chapter 7).

---

1. Classification defines the relationship between the mesh and the model of geometric domains. The mesh is a discretization of geometric domain(s). Each mesh entity M_i^d is uniquely classified onto a topological model entity G_j^D written in notation as $M_i^d \sqsubset G_j^D$ ($d \leq D$). A set of mesh entities can be classified onto the same model entity. For rigorous definition and its application see reference[8].

The particular structure is as follows: Chapter 2 introduces the representation of element shape and size distribution. The concept of anisotropic mesh size field is introduced for this purpose. Also the issue of determining the conformity between mesh and anisotropic mesh size field is addressed. Chapter 3 presents implementation of the anisotropic mesh size field. An abstract interface for mesh size field interrogation is first described, then two specific implementations used in the examples of this thesis are given: (i) analytical definition over model domain; and (ii) a piecewise definition over the mesh to be modified. Chapter 4 introduces the mesh modification operators used in the mesh modification procedure, and discusses the evaluations of these mesh modifications with respect to the anisotropic mesh size field and element quality criteria. Chapter 5 deals with the issue of curved domain approximation. It presents a two-stage mesh modification procedure to improve local geometric approximation in case the elements on the boundaries of 3-D curved domains are refined. Chapter 6 describes an effective *h*-version mesh adaptation procedure governed by anisotropic mesh size field, and discusses its technical aspects. It extends the previous work of de Cougny [23] in two aspects: effectiveness and respecting anisotropic mesh size field. Chapter 7 demonstrates the application of the mesh adaptation procedure in adaptive flow simulations using five examples, including a brief introduction of adaptive specification of the anisotropic mesh size field. Chapter 8 concludes the thesis by summarizing the contributions and discussing future work.

CHAPTER 2

Anisotropic Mesh Size Field

An essential requirement for an anisotropic mesh adaptation procedure is a means to define desired element size and shape distribution. The mesh metric field can be used to represent an anisotropic mesh size field that specifies the distribution of element size and stretching (see for example [31,62,66,93]). This chapter introduces the definition of the mesh metric field, then discusses the criteria of determining the conformity between a mesh to a mesh metric field that can guide mesh modifications.

2.1 Definition of Mesh Metric

Consider Figure 2.1. The anisotropic mesh size at point P is defined as a 3×3 symmetric positive definite mesh metric tensor ***T(P)*** such that the desired directional mesh edge length distribution at this point follows an ellipsoidal surface

$$\boldsymbol{X}\boldsymbol{T}(P)\boldsymbol{X}^T = 1 \tag{2.1}$$

where $\boldsymbol{X} = \{x_1, x_2, x_3\}$ is a coordinate vector.

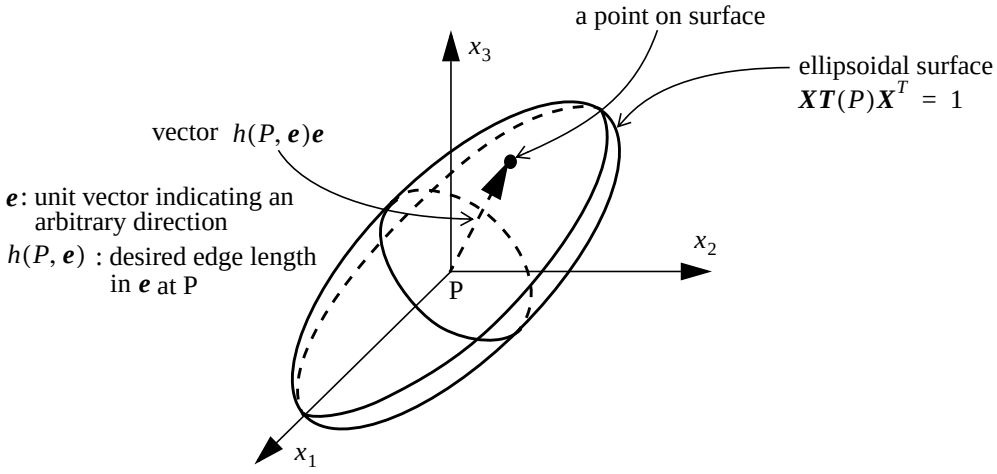


Figure 2.1 Definition of mesh metric tensor at a point.

The mesh metric tensor must be symmetric positive definite to ensure the geometric representation of quadratic form $\boldsymbol{X}\boldsymbol{T}(P)\boldsymbol{X}^T = 1$ is an ellipsoidal surface [10]. Moreover, (EQ-2.1) gives an implicit expression of directional variation of desired edge length in terms of ***T(P)***. Let ***e*** denote a unit vector emanating from P in an arbitrary direction and ***h(P,e)*** be the desired edge

length along $\boldsymbol{e}$ at P, then vector $h(P,\boldsymbol{e})\boldsymbol{e}$ should point to a point on the ellipsoidal surface and satisfy the quadratic form, thus,

$$(h(P,\boldsymbol{e})\boldsymbol{e})\boldsymbol{T}(h(P,\boldsymbol{e})\boldsymbol{e})^T = 1$$

$$h(P,\boldsymbol{e}) = \frac{1}{\sqrt{\boldsymbol{e}\boldsymbol{T}(P)\boldsymbol{e}^T}} \tag{2.2}$$

which is the explicit expression of directional variation of desired mesh edge length at point P.

For example, consider a mesh metric tensor $\begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$. The ellipsoidal surface associated with this metric tensor is:

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 1$$

namely

$$\frac{x_1^2}{0.5^2} + x_2^2 + \frac{x_3^2}{2^2} = 1$$

Figure 2.2 shows the ellipsoidal surface of this equation. The desired mesh edge lengths along axis x_i (i=1,2,3) are 0.5, 1, 2 respectively. For the direction $\begin{bmatrix} \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 \end{bmatrix}$ indicated by an arrow, the desired mesh edge length can be computed as follows:

$$\frac{1}{\sqrt{\begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.25 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} & \frac{\sqrt{3}}{3} \end{bmatrix}^T}} \approx 0.7559$$

2.2 Conformity Criteria

In general, mesh metric varies with position and a criteria is required to measure the degree of satisfaction between a mesh entity and the mesh metric field. To develop such a criteria, it is useful to introduce the transformation associated with each mesh metric tensor which maps the mesh entity of interest into a space where the mesh metric tensor becomes identity[66,91]. Measuring satisfaction of the mesh entity is done in the transformed space.

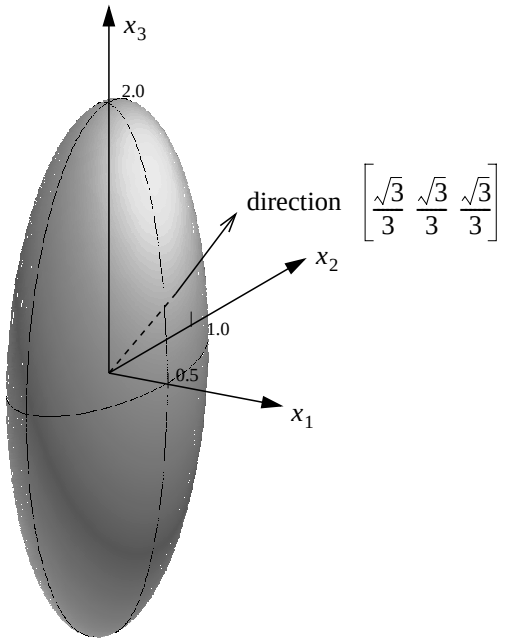


Figure 2.2 Example to show the geometric significance of a metric tensor.

The structure of the following subsections is: first, the transformation associated with the mesh metric tensor is introduced (Section 2.2.1). An ideal conformity criteria based on mesh edge length computation in the transformed space can then be developed (Section 2.2.2). Finally consideration is given to a relaxed criteria suitable for mesh modification procedures.

2.2.1 Transformation Interpretation of Mesh Metric Tensor

Due to symmetry and positive definiteness, mesh metric tensor $\boldsymbol{T}(P)$ can always be decomposed into two matrices $\boldsymbol{Q}(P)^T$ and $\boldsymbol{Q}(P)$ as:

$$\boldsymbol{T}(P) = \boldsymbol{Q}(P)\boldsymbol{Q}(P)^T \tag{2.3}$$

where

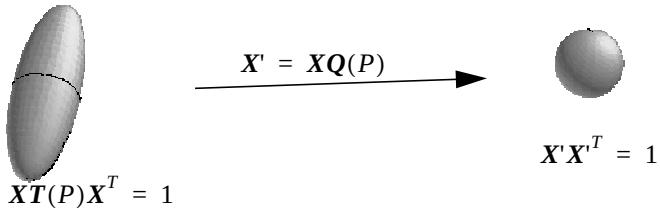
$$\boldsymbol{Q}(P) = \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix}}_{rotation} \underbrace{\begin{bmatrix} \sqrt{\lambda_1} & 0 & 0 \\ 0 & \sqrt{\lambda_2} & 0 \\ 0 & 0 & \sqrt{\lambda_3} \end{bmatrix}}_{distortion} \tag{2.4}$$

with e_i (i=1,2,3) being the normalized eigenvectors (row vectors) and λ_i (i=1,2,3) being eigenvalues of $\boldsymbol{T}(P)$[1]. Note that matrix $\boldsymbol{Q}(P)$ can be considered as the representation of a linear transformation from the physical space to a rotated distorted space [45] and we can define a new coordinate vector after the distortion and rotation as $\boldsymbol{X}' = \boldsymbol{X}\boldsymbol{Q}(P)$. Since:

$$\boldsymbol{X}'\boldsymbol{X}'^{T} = \boldsymbol{X}\boldsymbol{Q}(P)(\boldsymbol{X}\boldsymbol{Q}(P))^{T} = \boldsymbol{X}\boldsymbol{Q}(P)\boldsymbol{Q}(P)^{T}\boldsymbol{X}^{T} = \boldsymbol{X}\boldsymbol{T}(P)\boldsymbol{X}^{T} = 1 \tag{2.5}$$

which is an equation of a unit sphere surface, i.e., the transformation represented by matrix $\boldsymbol{Q}(P)$ maps the mesh metric tensor into a space in which the desired element size and shape distribution is isotropic and normalized.

Figure 2.3 illustrates the geometric significance of this transformation. It transforms the ellipsoidal surface associated with $\boldsymbol{T}(P)$ into a unit sphere.



(a) before transformation (b) after transformation

Figure 2.3 The transformation associated with mesh metric tensor.

Consider mesh metric $\begin{bmatrix} 3 & 1 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$ for an example of the transformation interpretation. The eigenvalue λ_i and the unit eigenvector $\boldsymbol{e}_i$ (i=1,2,3) of this mesh metric are:

$$\begin{cases} \lambda_1 = 4 \\ \lambda_2 = 2 \\ \lambda_3 = 0.25 \end{cases} \qquad \begin{cases} e_1 = \left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right) \\ e_2 = \left(-\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0\right) \\ e_3 = (0, 0, 1) \end{cases}$$

The transformation matrix $\boldsymbol{Q}(P)$ associated with the mesh metric can be constructed:

---

1. In terms of (EQ-2.2), eigenvalue λ_i is related to the desired mesh edge length h_i in direction e_i, namely $h_i = 1/(\sqrt{\lambda_i})$.

$$\underbrace{\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 0.5 \end{bmatrix}}_{\mathbf{D}} = \underbrace{\begin{bmatrix} \sqrt{2} & -1 & 0 \\ \sqrt{2} & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}}_{\mathbf{Q}(\mathrm{P})}$$

where **R** is the rotation matrix and **D** is a diagonal matrix representing distortion. Hence, the relation between the coordinate vectors of the two space is

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} \sqrt{2} & -1 & 0 \\ \sqrt{2} & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

The ellipsoidal surface equation associated with this mesh metric is:

$$3x_1^2 + 2x_1x_2 + 3x_2^2 + 0.25x_3^2 = 1$$

By replacing x_i in terms of x'_i *(i=1,2,3)*, the ellipsoidal surface equation becomes $x'^2_1 + x'^2_2 + x'^2_3 = 1$, which represents a unit sphere.

Note that the rotation matrix associated with the mesh metric is not unique since the eigenvectors of the mesh metric tensor is not unique. However, no matter what eigenvector is used, the ellipsoidal surface associated with the mesh metric is uniquely determined. For example, if two eigenvalues of the mesh metric are identical, the mesh metric is associated with a spheroid surface as illustrated in Figure 2.4. Clearly, any pair of orthogonal directions on the indicated plane can be used as the eigenvectors to define the transformation, and it does not affect the shape of the spheroid surface.

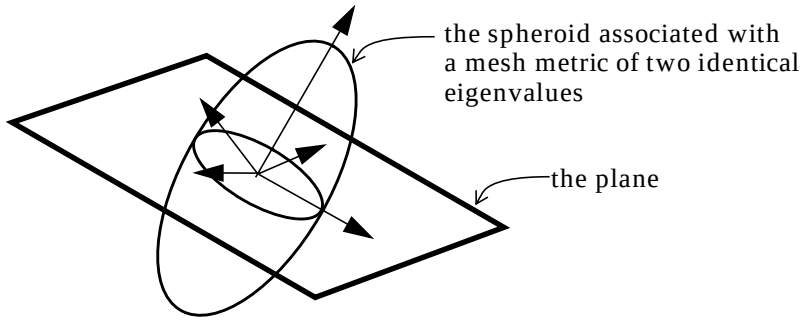


Figure 2.4 Example of possible infinite choices in defining rotation matrix.

2.2.2 Ideal Criteria

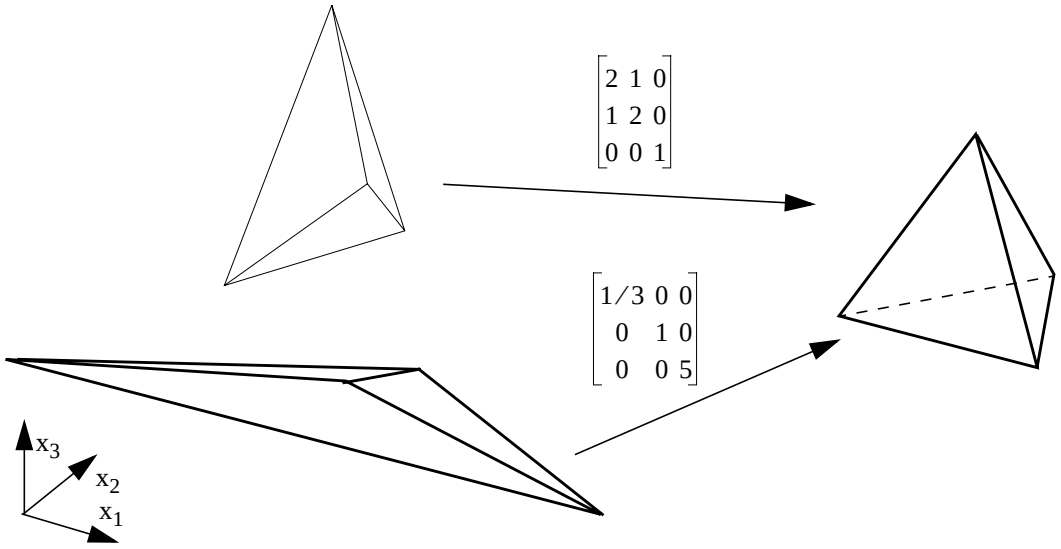
Ideally, the conformity criteria can be described from two aspects:

- Unit mesh edge length in the transformed space, and
- Aligning to the principle directions of the ellipsoidal surface as much as possible.

Since the mesh metric field represents the transformation that maps an ellipsoid into a unit sphere, any tetrahedron that perfectly satisfies the mesh metric field should be a unit equilateral tetrahedron in the transformed space. Here, a tetrahedron is referred to as a unit equilateral tetrahedron if all edges that bound the tetrahedron are straight and of unit length. Figure 2.5 depicts three dimensional examples to demonstrate this concept. Figure 2.5(a) depicts two desired anisotropic tetrahedra of different size and shape in physical space, while the transformation associated with the mesh metric field is indicated by the two matrices. As illustrated in Figure 2.5(b), both tetrahedra are transformed into a unit equilateral tetrahedron with respect to their corresponding transformation matrix.

Given an arbitrary tetrahedron in physical space and the mesh metric field over the tetrahedron, the degree of satisfaction between the tetrahedron and the mesh metric field can be measured by examining the difference between a unit equilateral tetrahedron and the mapped tetrahedron in the transformed space. One component of this is done by examining the lengths of the six mesh edges that bound the tetrahedron in the transformed space. The second component is to ensure the resulting tetrahedron is not flat in the transformed space as discussed (see Section 2.2.3). A tetrahedron is considered as possibly satisfying the mesh metric field if the six mesh edges are close to unitary in the transformed space. Note that it is assumed that a straight / planar tetrahedron in physical space can still be considered straight / planner after the transformation.

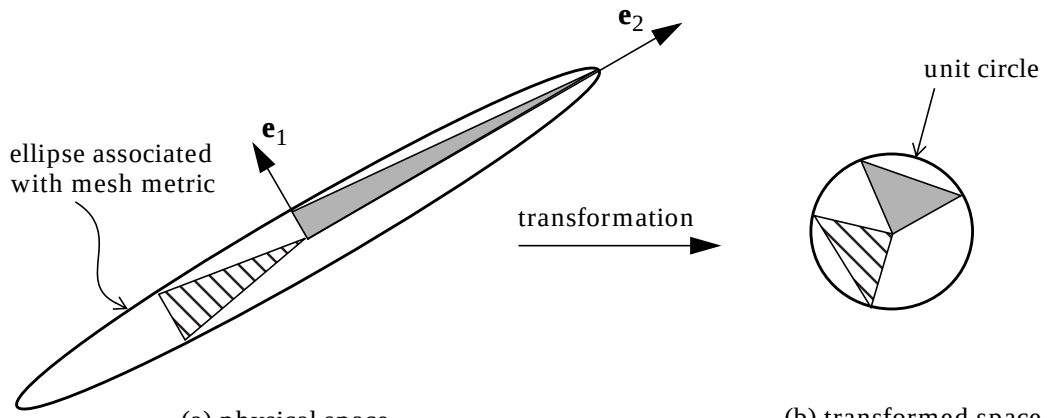
The second aspect of the ideal criteria can be illustrated using the 2D example depicted in Figure 2.6, where two triangles (indicated by different shade patterns) are depicted in physical space and the associated transformed space. The ellipse indicates the desired directional variation of mesh edge length in physical space, which is transformed into a unit circle in transformed space. Vector $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ indicate the principle directions of the ellipse. It can be seen that, although the two triangles are in the same degree of satisfaction to the unit circle in the transformed space (the two triangles are congruent), their shape and size in physical space still have difference. The one with two edges aligned with the principle direction $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ better catches the anisotropy, therefore, preferred.



(a) tetrahedra of different desired size and shape in physical space

(b) unit equilateral tetrahedron in transformed space

Figure 2.5 Example of transforming into a unit equilateral tetrahedron.



(a) physical space

(b) transformed space

Figure 2.6 2D example to show the needs of aligning to principle directions.

The first aspect of the ideal criteria requires to compute the mesh edge length in the transformed space, which is addressed in Section 2.2.2.1. For the second aspect, the alignment to principle direction(s) can simply be measured by computing the maximum / minimum angles between edges of a tetrahedron and principle direction(s) to be respected in the physical space. The determination of principle direction(s) to be respected requires to distinguish three cases: (i) if the geometric representation of the mesh metric tensor is an ellipsoid, all principle directions of the ellipsoid need to be respected; (ii) if the geometric representation is a spheroid, only the direction

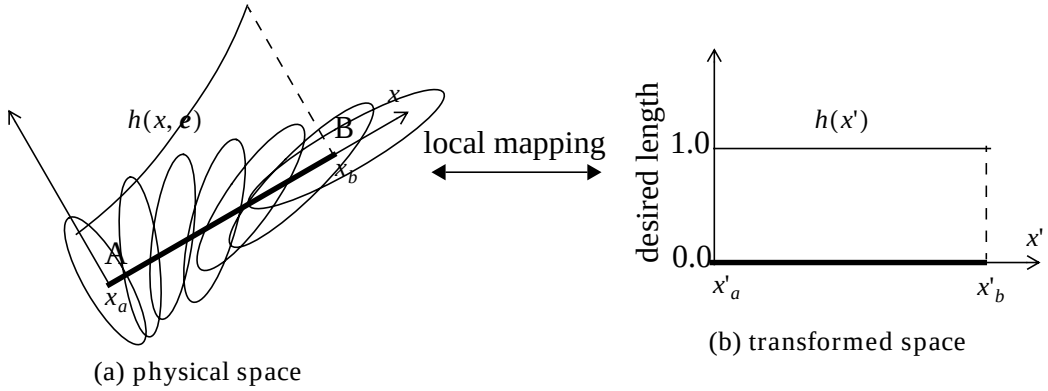
with distinct desired edge length should be respected; and (iii) if the geometric representation is a sphere (isotropic), no direction should be respected.

2.2.2.1 Length Computation in Transformed Space

Assume that mesh metric field is continuously specified over the domain and consider an arbitrary mesh edge in the mesh as depicted in Figure 2.7(a), where the edge is indicated using a thick black segment, and mesh metrics over the edge are shown by a set of ellipses. Let the x-axis along the edge with $x = x_a$ at point A and $x = x_b$ at point B, $\boldsymbol{T}(x)$ represent the given mesh metric at position x on the edge ($x_a \le x \le x_b$), and $\boldsymbol{e}$ denote the unit vector associated with the edge. Then, the desired mesh edge length along the edge is:

$$h(x, \boldsymbol{e}) = \frac{1}{\sqrt{\boldsymbol{e}\boldsymbol{T}(x)\boldsymbol{e}^T}} \tag{2.6}$$

The curve in Figure 2.7(a) shows one example of possible $h(x,\boldsymbol{e})$ distributions.



(a) physical space (b) transformed space

Figure 2.7 Local transformation that normalizes $h(x,\boldsymbol{e})$.

Figure 2.7(b) shows the edge in the transformed space where interval $[x_a, x_b]$ is mapped into $[x'_a, x'_b]$ and the desired edge length becomes one everywhere, i.e., $h(x') = 1$. Such a local mapping can be obtained by relating the differential length of the two space as follows:

$$dx' = \frac{dx}{h(x, \boldsymbol{e})} \tag{2.7}$$

which is the one dimensional version of transformation $\boldsymbol{X}' = \boldsymbol{X}\boldsymbol{Q}(P)$ in a differential sense, with dx denote the infinitesimal length at point x and dx' is its corresponding infinitesimal length in the mapped space.

Since $x = x_a$ is mapped to $x' = x'_a$, the integral relation between x and its corresponding coordinate x' in the transformed space is:

$$x' = x'_a + \int_{x_a}^{x} \frac{1}{h(x, \boldsymbol{e})} dx \tag{2.8}$$

Thus, length of edge AB in the transformed space is:

$$L'(AB) = x'_b - x'_a = \int_{x_a}^{x_b} \frac{1}{h(x, \boldsymbol{e})} dx \tag{2.9}$$

Plugging (EQ-2.6) leads to:

$$L'(AB) = \int_{x_a}^{x_b} \sqrt{\boldsymbol{e}\boldsymbol{T}(x)\boldsymbol{e}^T} dx \tag{2.10}$$

To illustrate the matching of desired mesh size distribution in terms of (EQ-2.10), consider generating an one dimensional mesh consisting of five edges that satisfy analytical mesh size field:

$$h(x) = 10 - 9e^{-0.08x} \tag{2.11}$$

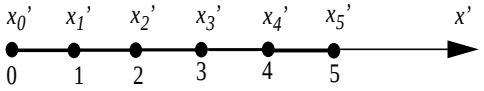
Figure 2.8(a) shows a mesh of five unit mesh edges in the transformed space. The coordinate x_i'of the six mesh vertices in transformed space is 0, 1, 2, 3, 4, 5, respectively. Let $x_0' = 0$ be mapped to $x_0=0$ in physical space. In terms of (EQ-2.8), the mapping between the two space is:

$$x' = \int_0^x \frac{1}{h(x)} dx = \frac{x}{10} + 1.25 \ln(10 - 9e^{-0.08x}) \tag{2.12}$$

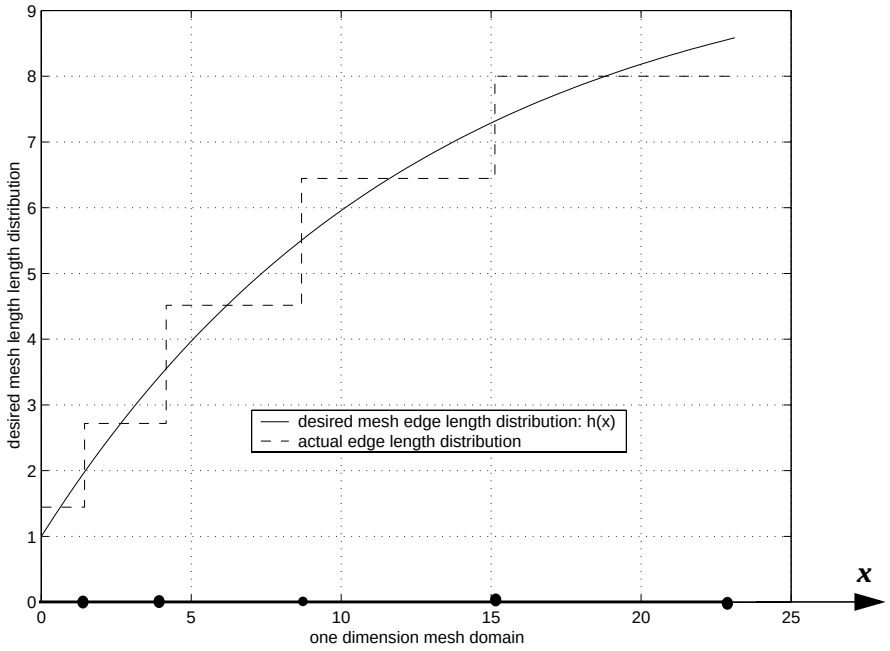
The inverse mapping from transformed space to physical space can also be obtained analytically:

$$x = -12.5 \ln \frac{10}{e^{0.8x'} + 9} \tag{2.13}$$

The mesh that exactly matches the given $h(x)$ can be obtained by mapping locations x_i' $(i=0,1,2,3,4,5)$ in transformed space back to physical space using (EQ-2.13). Table 2.1 tabulates these locations. The length of the five edges in physical space from left to right in turn are: 1.445, 2.719, 4.515, 6.445 and 8. Figure 2.8(b) shows the mesh on the horizon axis using bold segments and black bullets, where the solid smooth curve shows the desired mesh size distribution over the domain, while the dashed step curve shows the actual mesh size distribution. It can be seen that the step curve approximates the curve of desired mesh size distribution.



(a) five unit edges in transformed space



(b) mesh and mesh size distribution in physical space

Figure 2.8 An example of exactly matched 1D mesh.

Table 2.1 Vertex location mapping between physical and transformed space.

i	0	1	2	3	4	5
x_i'	0	1	2	3	4	5
x_i	0	1.445	4.164	8.679	15.124	23.124

2.2.3 Relaxed Criteria for Mesh Modifications

The lack of any packing proofs for elements of given size and shape ensures that any mesh created will not perfectly match the given mesh metric field. Therefore, the definition of an acceptable representation of the mesh metric field is needed. The one used here is based on:

- Considering a mesh edge to satisfy the mesh metric field if its transformed length falls into an appropriate interval close to one where the interval is defined so that infinite loop of refinement and coarsening is avoided.
- When edges that bound a tetrahedron satisfy the relaxed length criteria, the tetrahedron must be in good quality (not flat) in the transformed space.

The first criteria relaxes edge length requirement from a value into an interval. The second element level criteria is required since, in three dimension, an element with all bounding edges close to unit length can have very small volume, *i.e.* a sliver (see Figure 2.9), which can cause problems in simulations.

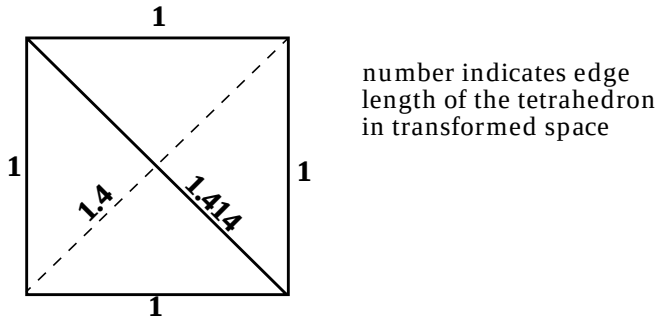


Figure 2.9 Example of sliver tetrahedron in transformed space.

The next three subsections discuss these two relaxed criteria in detail and present the needed computation of any additional quantity in the transformed space.

2.2.3.1 Allowed Interval for Relaxed Length Criteria

The relaxed mesh edge length criteria introduces an interval of allowed mesh edge length, referred to as $[L_{upper}, L_{lower}]$ hereafter. To prevent the possible oscillation between mesh refinement and mesh coarsening, the selection of L_{upper} and L_{lower} should at least satisfy:

$$L_{lower} \leq 0.5 L_{upper} \tag{2.14}$$

Note that, whenever a mesh edge is decided to be split, its length in transformed space should be greater than L_{upper} and the two new mesh edges should both be greater than or equal to L_{lower} in the transformed space, otherwise the mesh coarsening operation would un-do the splitting. Furthermore, the splitting operation always creates two new edges, one longer than or equal to the half edge length in transformed space, while another always shorter or equal to the transformed half length. (EQ-2.14) ensures that, whenever bisecting a mesh edge longer than L_{upper} in trans-

formed space, the length of the two new mesh edges are longer than L_{lower} in transformed space. This factor of 0.5 should be accordingly reduced in case of not bisection in transformed space.

The selection of the interval is not unique after satisfying (EQ-2.14). It can be set to many reasonable values, for example $\left[\frac{\sqrt{2}}{2}, \sqrt{2}\right]$, [*0.4, 1.0*] and etc. The procedure presented in this thesis does allow the user to adjust one of the parameters, within limits, while the second is calculated to ensure (EQ-2.14) is satisfied.

2.2.3.2 Selection of Element Quality Measure

Many element quality measures can be extended into the transformed space and satisfy the need of the second element level criteria. Some of these measures are listed below. More introductions can be obtained in references [52,53,24,26,30,27].

- Angle (pp 7 of [26]), $\phi \equiv C(1 + \cos\varphi_{max})$, where φ_{max} is the maximum dihedral angle of a tetrahedron and C is normalization constant.
- Radius ratio [52], $\rho \equiv 3 \times r_i / r_o$, where r_i is the inscribed radius of the tetrahedron and r_o the radius of circumsphere.
- Aspect ratio [26, 27], $\delta \equiv \frac{\sqrt{6} \times h_{min}}{2 \times l_{max}}$, where h_{min} is the minimum height of the tetrahedron and l_{max} the maximum edge length.
- Face area based measure [24], $S \equiv 108 \times V^4 / \left(\sum_{i=1}^{4} A_i^2 \right)^3$, where V is tetrahedron volume and A_i is the area of the i-th face that bounds the tetrahedron.
- Mean ratio (edge length based) [53], $\eta \equiv 12 \times (3V)^{2/3} / \sum_{i=1}^{6} l_i^2$, where V is tetrahedron volume and l_i denotes the length of i-th edge of the tetrahedron.

Since these measures are “equivalent” in a mathematical sense such that they attain a maximum value only for the equilateral tetrahedron [52], which is used does not make any difference. Hence, the one of least computational cost should be used and extended into the transformed space, which is:

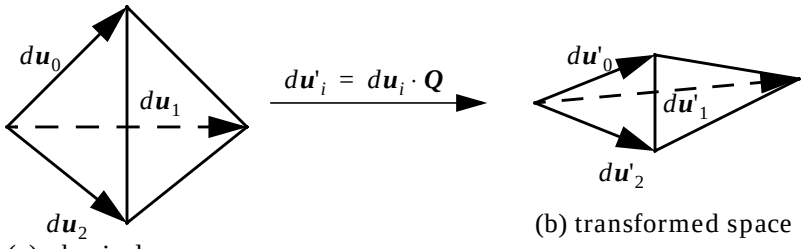
$$\eta' = 15552 \times V'^2 / \left(\sum_{i=1}^{6} l_i'^2 \right)^3 \tag{2.15}$$

where V' denotes the tetrahedron volume in transformed space, l'_i is transformed length of mesh edges that bound the tetrahedron. η' is the cubic of the mean ratio measure in the transformed space.

In this thesis, the selected element quality measure is used for two purposes: (i) determine sliver tetrahedra in the shape correction process of the mesh adaptation procedure (see Section 6.4.1), (ii) in cases where more than one local mesh modification are possible, determine the best one in terms of this quality measure in the process of placing vertices onto boundaries (see Section 5.4) and the shape correction process of mesh adaptation procedure (see Section 6.4.3).

2.2.3.3 Volume Computation in Transformed Space

The element level requires the computation of tetrahedral volume in transformed space. Consider an infinitesimal tetrahedron specified by three differential vectors $d\boldsymbol{u}_i$ *(i=1,2,3)* as depicted in Figure 2.10, its volume in physical space is:



(a) physical space (b) transformed space

Figure 2.10 A tetrahedron in physical and transformed space.

$$dV = \frac{(d\boldsymbol{u}_1 \times d\boldsymbol{u}_2) \cdot d\boldsymbol{u}_3}{6} = \frac{1}{6} \begin{vmatrix} d\boldsymbol{u}_1 \\ d\boldsymbol{u}_2 \\ d\boldsymbol{u}_3 \end{vmatrix} \tag{2.16}$$

Let $\boldsymbol{Q}$ be the transformation matrix the metric field specified at this infinitesimal tetrahedron, and $d\boldsymbol{u}_i$ be transformed into $d\boldsymbol{u}'_i$ by this transformation, that is $du'_i = d\boldsymbol{u}_i\boldsymbol{Q}$. Then, the infinitesimal volume in the transformed space is:

$$dV' = \frac{(d\boldsymbol{u}'_1 \times d\boldsymbol{u}'_2) \cdot d\boldsymbol{u}'_3}{6} = \frac{1}{6}\begin{vmatrix} d\boldsymbol{u}'_1 \\ d\boldsymbol{u}'_2 \\ d\boldsymbol{u}'_3 \end{vmatrix} = \frac{1}{6}\begin{vmatrix} d\boldsymbol{u}_1 \cdot \boldsymbol{Q} \\ d\boldsymbol{u}_2 \cdot \boldsymbol{Q} \\ d\boldsymbol{u}_3 \cdot \boldsymbol{Q} \end{vmatrix} = \frac{1}{6}\begin{vmatrix} d\boldsymbol{u}_1 \\ d\boldsymbol{u}_2 \\ d\boldsymbol{u}_3 \end{vmatrix} \cdot |\boldsymbol{Q}| \tag{2.17}$$

Thus,

$$dV' = |\boldsymbol{Q}| \cdot dV = \sqrt{|\boldsymbol{T}|} \cdot dV \tag{2.18}$$

where $\boldsymbol{T} = \boldsymbol{Q}\boldsymbol{Q}^T$ is the mesh metric at this infinitesimal tetrahedron. For a finite tetrahedron in which $\boldsymbol{T}$ varies, the transformed volume can be obtained by integrating (EQ-2.18):

$$V' = \int_V \sqrt{|\boldsymbol{T}|}\, dV \tag{2.19}$$

CHAPTER 3

Mesh Size Field Implementation

The mesh size field is implemented in a two level hierarchy. The first level is simply an abstract interface, serving as the base of all mesh size fields. The second level consists of instantiations of the abstract interface, which can include infinite versions of implementation possibilities. This chapter first presents the abstract mesh size field interface. Discussion is then focused on two specific anisotropic mesh size field implementations used in the examples of this thesis:

- analytical definition,
- an efficient piecewise definition option.

3.1 Abstract Interface

The abstract mesh size field class is designed to be wrappers around all functionalities that mesh modification procedures need. Figure 3.1 illustrates the role of this abstract class. It can be viewed as a gate all meshing algorithm must go through regardless of its underlying implementations. On the right side, there can be infinite implementation possibilities, for example, analytical definition, piecewise definition-1, piecewise definition-2 and etc. All need to be derived from this abstract class and override the abstract version of virtual functions [85]. The gain by doing so is generality and flexibility. This ensures that any addition of new mesh size field definitions is always consistent with mesh modification procedures.

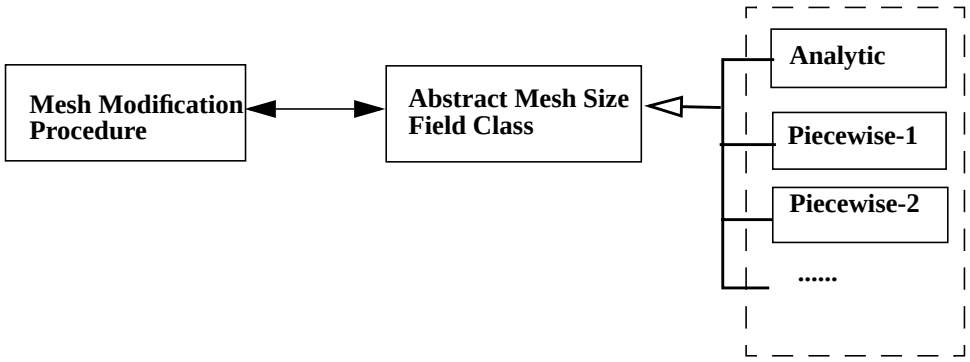


Figure 3.1 Role of abstract mesh size field class.

The definition of this class, “SizeFieldBase”, is given in Figure 3.2. It collects together all of the functionalities that the mesh modification procedures in this thesis use. The functionalities include geometry computation in transformed space, specifically, volume, length and middle point computation. There are also functions to evaluate size information at a point, set size infor-

mation onto a mesh entity[1] and get the representation type of mesh size field. The *type()* function identifies three representation types: analytically, using a background mesh, or using the current mesh to be modified.

```
class SizeFieldBase {
public:
    // compute transformed edge length
    virtual double length(pEdge);
    // compute transformed volume
    virtual double volume(pRegion);
    // compute center point of an edge
    virtual void center(pEdge,double*);
    // evaluate mesh size at a point
    virtual pMSize eval(double[3],pEntity);
    // attach mesh size info onto an entity
    virtual void set(pEntity,pMSize);
    // get the type of mesh size field representation
    virtual int type();
};
```

Figure 3.2 Abstract class of mesh size field.

3.2 Two Instantiations

From a practical point of view, the mesh size field may be defined in two ways: as an analytical function of position, or as interpolations over a structure. The first way is independent of meshes, and can provide exact mesh size information everywhere efficiently, however, it can not be used in adaptive analysis where mesh size is adaptively specified in terms of error indicators. The second way can have infinite versions by using different underlying structures and interpolations. A background mesh appears to be the most commonly used structures of defining mesh size field [66,67,58,35,31,48,64]. The benefits of keeping a background mesh is to separate mesh size definition from the mesh to be generated or being modifying. The major deficiency is its lack of efficiency. To evaluate the size information at a point, it has to search the background mesh to determine the element in which the point resides and perform interpolation. This searching process is in the complexity of *O(N)* with *N* be the number of elements in background mesh, and *N* can be a big number in case the mesh in previous step of adaptive analysis is used as background mesh. The use of octree as background mesh can make the searching process *O(logN)*, however, lead to unnecessary efforts to build the octree and relate octree to mesh size information. To gain efficiency, here, developing an instantiation that attaches mesh metrics onto vertices of the current

1. The *set()* function is needed in case a mesh is used to represent the mesh size field.

mesh is of interest. Although such an instantiation need account for the possible evolution of mesh size field, it overcomes the deficiency of background mesh. The idea of using the current mesh is also mentioned by George *at al.* in a recent paper [34].

The subsections that follow describe two instantiations to be used in the examples of this thesis. The first one represents mesh size field using analytical expression(s). The second is an efficient piecewise definition over the current mesh.

3.2.1 Analytic Definition over Model Domain

In this instantiation, analytical expression(s) of mesh metric is attached to model entity(entities) so that, at any point of the problem domain, mesh metric information is defined.

Analytical expressions are specified by selecting three local orthogonal directions at each point and defining these directions and desired edge lengths in these directions as expressions of position. For example, to specify a cylindrical anisotropic mesh size field over domain Ω, first pick the axial directions of cylindrical curvilinear coordinate system as orthogonal directions for an arbitrary point P *(x,y,z)* in Ω, namely:

1. radial direction: $(\cos\theta, \sin\theta, 0)$;

2. azimuthal direction: $(-\sin\theta, \cos\theta, 0)$;

3. z-axis: *(0,0,1)*;

where $\cos\theta = x/r$, $\sin\theta = y/r$ and $r = \sqrt{x^2 + y^2}$. Then define the desired edge length in these three directions also as functions of position:

$$\begin{cases} h_r = 0.1(1 - e^{-|r^2 - 0.25|}) + 0.0001 \\ h_\theta = h_z = 0.1 \end{cases} \tag{3.1}$$

The equivalent transformation matrix representation, in turn, is:

$$\boldsymbol{Q}(x, y, z) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/h_r & 0 & 0 \\ 0 & 1/h_\theta & 0 \\ 0 & 0 & 1/h_z \end{bmatrix} \tag{3.2}$$

Since mesh metric can be efficiently obtained at any point of the domain, the length and volume in the transformed space can simply be computed by applying Guass quadrature to (EQ-2.10) and (EQ-2.19). The computation of edge center needs searching so that the two new segments have the same transformed length in terms of (EQ-2.10).

3.2.2 A Piecewise Definition Option over Current Mesh

In this instantiation, mesh metric is attached to all vertices of current mesh. Since this mesh metric field definition depends on the mesh to be modified, accounting for the possible mesh size field evolution due to mesh vertex insertion and repositioning is needed. Specifically, the mesh metric at new location needs to be interpolated and attached to the new vertex (or the moved vertex).

The specific algorithms the piecewise definition option uses to instantiate the virtual functions of the abstract interface are as follows.

3.2.2.1 Transformed Edge Length

As illustrated in Figure 3.3, given a mesh edge AB and mesh metric $\boldsymbol{T}_i$ *(i=0,1)* at the ends of edge AB, the needed functionality is computing the transformed length, *L'(AB).*

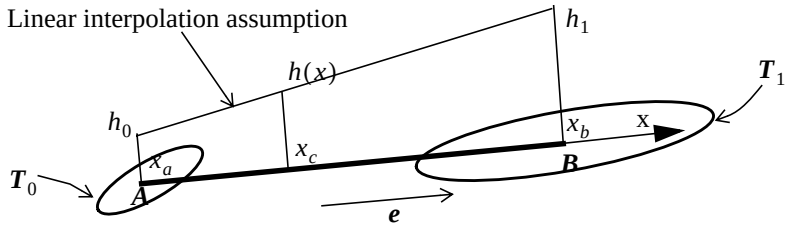


Figure 3.3 Illustration of transformed length computation for edge AB.

Let $\boldsymbol{e}$ be the unit vector along the direction of edge AB. In terms of the definition of mesh metric, the desired edge length along $\boldsymbol{e}$ at the two ends of AB are:

$$h_i = \frac{1}{\sqrt{\boldsymbol{e}\boldsymbol{T}_i\boldsymbol{e}^T}} \qquad i = 0, 1 \tag{3.3}$$

Also, in terms of (EQ-2.9), the transformed length of edge AB depends on the desired edge length distribution over AB, *h(x),* which is not completely specified. Therefore, a reasonable assumption is required to account for the variation of *h(x)* from $h(x_a) = h_0$ to $h(x_b) = h_1$. The following linear interpolation has been assumed in current piecewise definition:

$$h(x) = \frac{x_b - x}{L}h_0 + \frac{x - x_a}{L}h_1 \qquad x_a \le x \le x_b \tag{3.4}$$

with $L = |x_b - x_a|$. The transformed length of edge AB can then be obtained by plugging (EQ-3.4) into (EQ-2.9) and integrating:

$$L'(AB) = \frac{L}{h_1 - h_0}\ln\left(\frac{h_1}{h_0}\right) \tag{3.5}$$

3.2.2.2 Edge Center in Transformed Space

Let C denotes the center point of edge AB in transformed space, i.e.,

$$L'(AC) = 0.5L'(AB)$$

Plugging (EQ-2.9) at both sides of above equation leads to:

$$\int_{x_a}^{x_c}\frac{1}{h(x)}dx = 0.5\int_{x_a}^{x_b}\frac{1}{h(x)}dx \tag{3.6}$$

where x_c is the coordinate of center C (refer to Figure 3.3). Following the same linear interpolation assumption and plugging in *h(x)*, the expression for x_c can be derived out:

$$x_c = x_a + \frac{1}{1 + \sqrt{h_1/h_0}}(x_b - x_a) \tag{3.7}$$

3.2.2.3 Transformed Volume of Tetrahedron

To be conservative, the tetrahedral volume in the transformed space is computed as:

$$V' = min(\sqrt{|\boldsymbol{T}_i|}\cdot V) \qquad i = 0, 1, 2, 3 \tag{3.8}$$

where $\boldsymbol{T}_i$ denotes the mesh metric at i-th bounding vertex of the tetrahedron.

3.2.2.4 Interpolation

Given a mesh entity and a point in its closure, the needed functionality is to evaluate mesh metric at the given point by interpolating mesh metrics attached to bounding vertices of the mesh entity.

There are at least three ways to interpolate mesh metrics. The first way is to interpolate the six components of mesh metrics, which is easy to implement and efficient. The second way, as proposed by George and Hecht in [35], is to interpolate two mesh metrics by solving a generalized eigenvalue problem. The third way uses the ellipsoidal interpretation of mesh metric. It consists of obtaining the principle directions of the ellipsoid and desired edge lengths in these principle directions for each mesh metric attached to bounding vertices, and interpolating these directions and desired edge lengths at the given point.

In the current implementation, the third interpolation method is used. The first component by component interpolation is not effective since it does not take rotation effect into consideration. The second method is out of the consideration for two reasons:

- the interpolation operation is not associative;
- the computation of generalized eigenvalue problem is expensive.

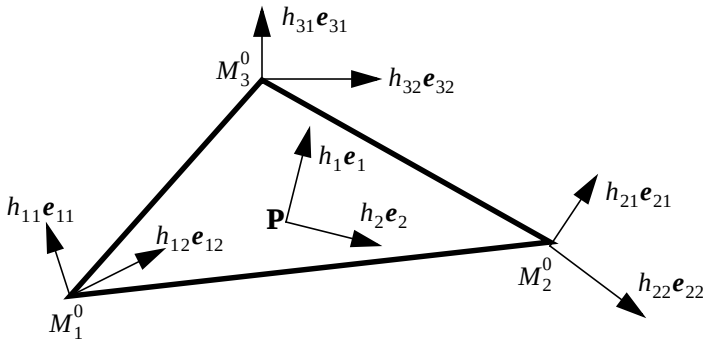


Figure 3.4 Illustration of mesh metric interpolation over a triangle.

Figure 3.4 depicts a 2D example to illustrate the third mesh metric interpolation method, where mesh metrics are represented by orthogonal vector pairs indicating the two semi-axes in desired edge length of the ellipse associated with mesh metric. The goal is to determine the vector pair at point P as functions of orthogonal vector pairs attached to vertex M_i^0 (i=1,2,3). The used index convention in Figure 3.4 is as follows: boldface $\mathbf{e}_{ik}$ denotes the unit vector in the *k*-th (*k=1,2*) principle direction of the mesh metric attached to vertex M_i^0. h_{ik} represents the desired edge length in $\mathbf{e}_{ik}$ with $h_{i1} < h_{i2}$ (*i=1,2,3*). Using linear interpolation shape functions, the desired length h_k and the corresponding principle direction $\boldsymbol{e}_k$ at point P can be expressed as:

$$\begin{cases} h_k = N_i \cdot h_{ik} \\ \boldsymbol{e}'_k = N_i \cdot \boldsymbol{e}_{ik} \\ \boldsymbol{e}_k = \boldsymbol{e}'_k / |\boldsymbol{e}'_k| \end{cases} \tag{3.9}$$

where N_i is the linear shape function of the triangle element at vertex M_i^0, and Einstein summation convention is adopted. The first equation interpolates desired edge lengths, the second equation interpolates principle directions and the third equation normalizes interpolated principle directions. In case of three dimension, the three interpolated directions at P may not be orthogonal. Whenever not orthogonal, the size and the direction of the smallest desired edge length should be respected and those of the other two directions may be adjusted.

CHAPTER 4

Mesh Modification Operation

The mesh modification procedures operate as a sequence of local operations ordered to make the mesh conform to a given mesh metric field. This chapter introduces the set of local mesh modification operators used (Section 4.1), and discusses the evaluation of these local mesh modification operators with respect to a given anisotropic mesh size field and element quality criteria (Section 4.2). Section 4.3 presents implementations.

4.1 Operators

The local mesh modification operators used in the tetrahedral mesh modification procedures consist of four single step operators [23,25,65,60,79]:

- splitting;
- collapsing;
- swapping;
- vertex repositioning;

and compound operators, which chain multiple single step operators. Each local mesh modification has an associated cavity consisting of a number of tetrahedra the local mesh modification affects. The application of the local mesh modifications is to replace one cavity triangulation with another cavity triangulation, while the new triangulation is uniquely determined by given information or only limited configurations are possible.

Most existing procedures published in the literature present specific algorithmic application of mesh modification operators. Typical algorithms are:

- Refinement methods that control element shape by specific split orders [3,7,9,11,12,41,54,55,75]. These procedures tend to over refine and do not consider the generalization to anisotropic case;
- Methods that coarsen only by undoing refinement [11,41,74,83]. These methods can not coarsen past the initial mesh;
- Fixed order modifications without analysis of situation [19,20,23,25,65]. These methods do not most effectively provide the best results.

A general approach is to construct a cavity by deleting a number of tetrahedra then remesh the cavity using mesh generation algorithms (Section 5.5 discusses one such approach). Although limited use of such general approach could be acceptable to ensure solving a specific

problem, any extensive use of the general approach should be avoided since it is complicated, time consuming and would make a local mesh modification procedure not as effective as would be by simply re-meshing the whole domain. Furthermore, the result mesh configuration of the general approach can not be effectively predicted.

This section introduces the four single step operators and compound operators. It should be indicated that all single step operators are taken from previous works, so interested reader can also consult other references, for example [23,79]. Compound operations are new contribution of this thesis.

4.1.1 Splitting

As illustrated in Figure 4.1, in three dimensions, the split operators are: region split, face split and edge split. Region split inserts a vertex inside a mesh region and splits the mesh region into four child regions; Face split introduces a vertex with the same classification as the face, splits the face and the two mesh regions the face bounds; and edge split creates a vertex with the same classification as the edge, splits the edge and all higher dimension mesh entities the edge bounds. Let M_i^3, M_i^2 and M_i^1 denote the region, the face and the edge to be split. The cavity associated with these three operators in turn consists of M_i^3, tetrahedra in set $M_i^2\{M^3\}$, and tetrahedra in set $M_i^1\{M^3\}$, respectively. Splitting is always possible and only modifies local mesh topology in case the mesh entities to be split is classified on model region or on polyhedra boundary. If the mesh entity to be split is classified on a curved model boundary, the new vertex has to be placed onto the model entity it is classified on [50].

When multiple edges are marked for splitting, tetrahedra can be quickly subdivided through the application of refinement templates [23,79]. Refinement template differs from a single split in that it only considers one tetrahedron at a time, and split the tetrahedron regardless of its neighboring tetrahedra. Therefore, it would yield a non-conforming mesh if the subdivision process not handle the conformity issue. Consider all possibilities of marking refinement edges[1], there are forty-two surface triangulation options to subdivide a tetrahedron (refer to Figure 4.2). Of these options, four require creating an interior vertex (steiner point) since the splitting of a tetrahedron may require to triangulate a prism sub-domain that can not be tetrahedralized without the introduction of an interior vertex [87] (see Figure 4.3 for an example); and several have ambiguities in creating diagonal edges (detailed discussion is given in Section 6.3.3).

1. Refinement edge is referred as to the mesh edge to be split.

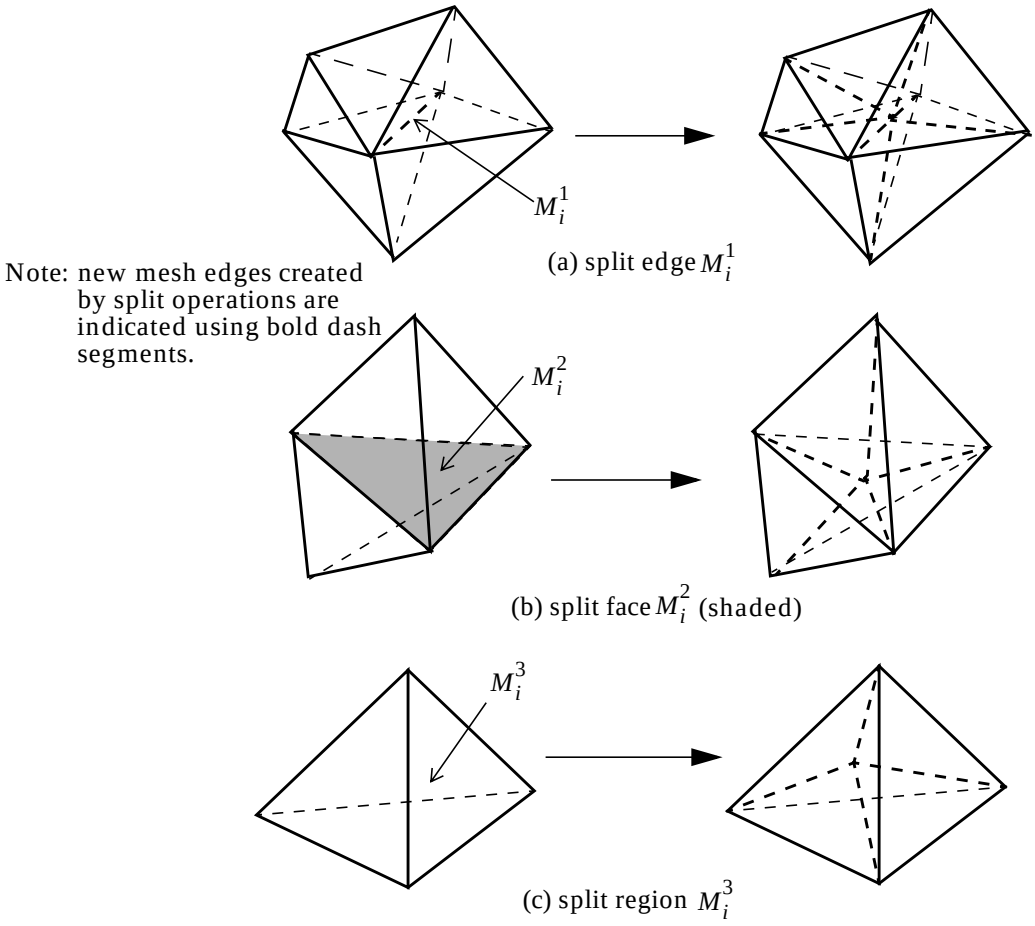


Figure 4.1 Split operators.

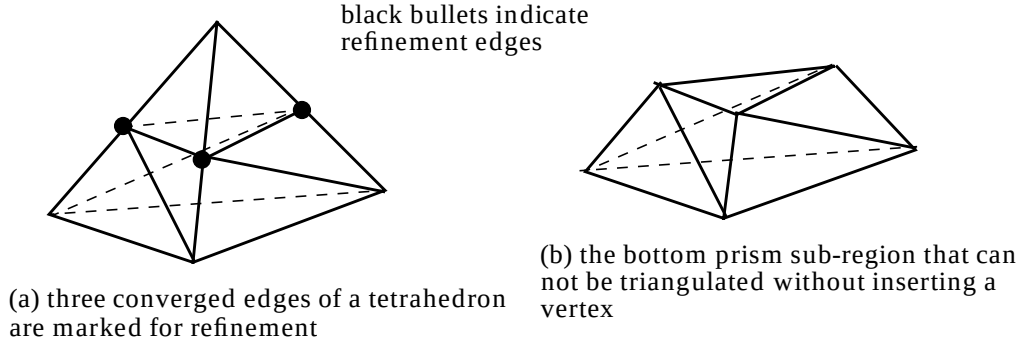


Figure 4.3 A refinement template that needs insert a interior vertex.

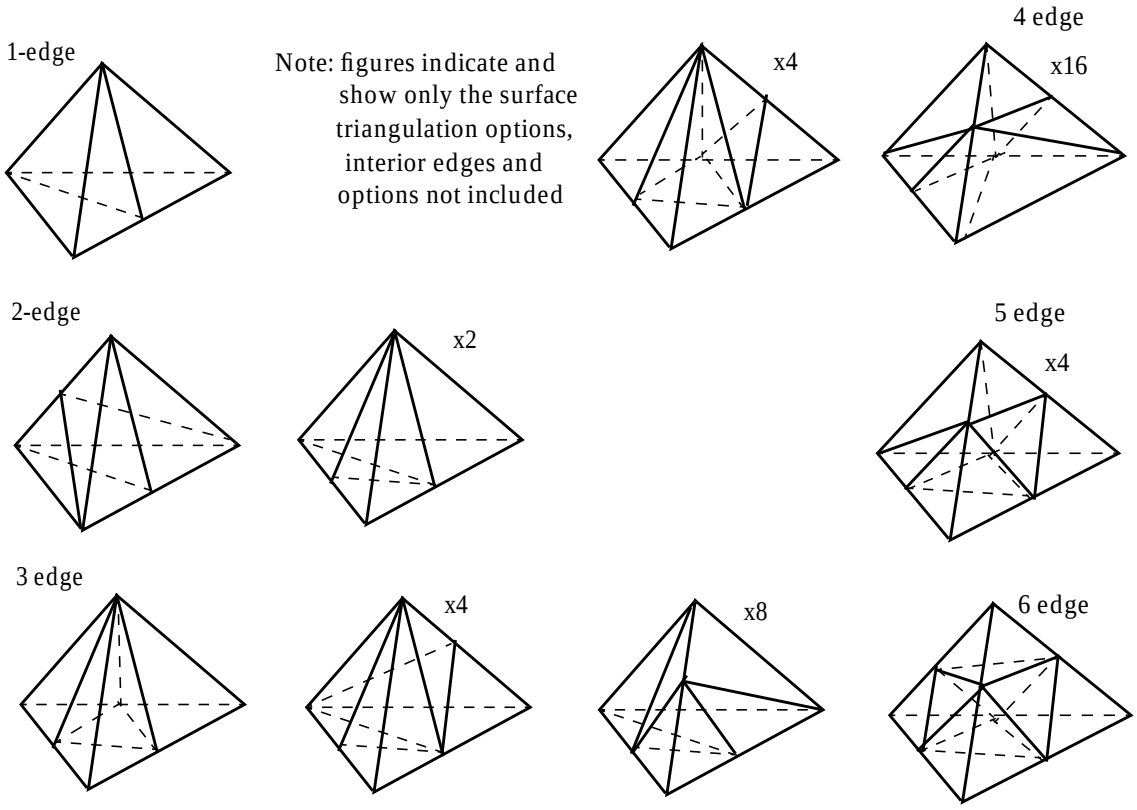


Figure 4.2 Tetrahedral subdivision templates.

4.1.2 Collapsing

Let M_i^1 represent an edge to be collapsed and M_d^0 be the end vertex of M_i^1, and consider the operation of collapsing M_i^1 with M_d^0 removed. As illustrated in Figure 4.4, the cavity associated with this collapse operation consists of tetrahedra in set $M_d^0\{M^3\}$, and the operation can be seen as a retriangulation of the cavity by "pulling M_d^0" to M_r^0 [23]. Specifically, the operation includes three steps: (i) delete M_i^1 and all mesh entities M_i^1 bounds; (ii) move M_d^0 to the position of M_r^0; and (iii) merge M_d^0 and M_r^0 together as well as all duplicated mesh edges and mesh faces.

Edge collapsing can not always be performable. In many cases, it can not be applied since it may destroy topological compatibility [81] between mesh and model, or violate geometric restrictions (see Section 4.2.1). Furthermore, collapsing a boundary edge should consider the changes to the geometry of mesh domain it will introduce.

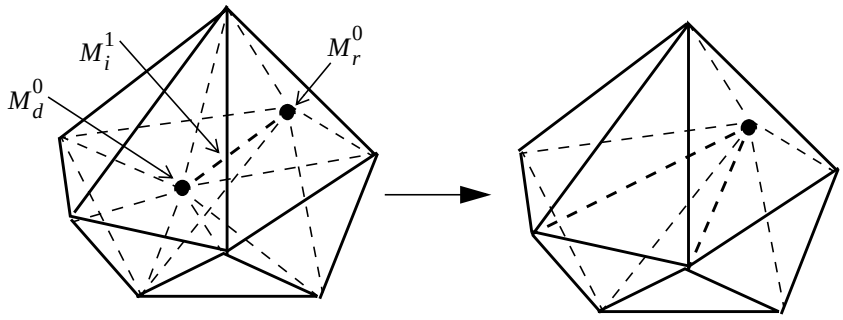


Figure 4.4 Collapse M_i^1 with M_d^0 removed.

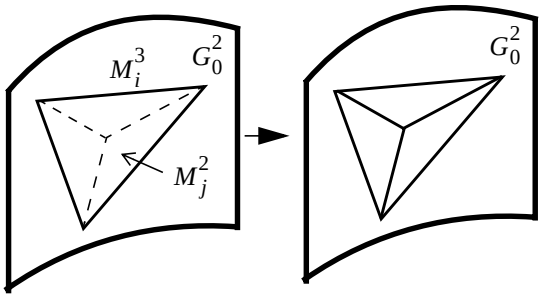
Another useful collapse operation is region collapsing, which deletes a tetrahedron next to boundary and re-classifies the mesh entities being brought onto boundary (pp38-39 of [26]). As illustrated in Figure 4.5, three types of region collapsing are: (i) removing a boundary face; (ii) removing two boundary faces, and (iii) removing three boundary faces. Similar to edge collapsing, region collapsing can not be applied if it geometrically introduces a hole to mesh domain, or violates the topological compatibility between mesh and model. In case of the first type of region collapse where an interior vertex is re-classified onto boundary (Figure 4.5(a)), the re-classified vertex has to be relocated onto the model entity it is classified on [50].

4.1.3 Swapping

Swap operation modifies the local mesh connectivity in a cavity. As depicted in Figure 4.6, two swap operators are most commonly used: face swap and edge swap. Let M_i^1 represent an edge to be swapped and M_j^2 be the face to be swapped, the associated cavity is $M_i^1\{M^3\}$ for swapping M_i^1, and $M_j^2\{M^3\}$ for swapping M_j^2.

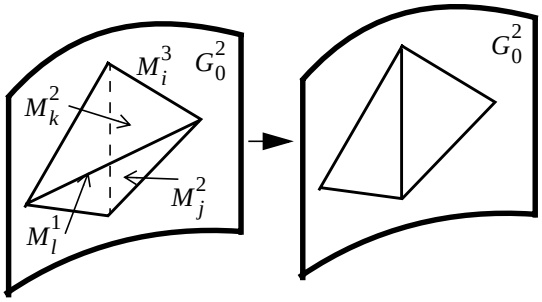
Edge swapping retriangulates the cavity associated with an edge to remove this edge (Figure 4.6(a)). It is important to realize that, since there is a large number of possibilities to retriangulate the polyhedron in case the edge to be swapped is classified on model region, determining a new triangulation could be an expensive operation in three dimensions. Let n be the number of tetrahedra connected to the edge, N_n be the number of possible retriangulation of the cavity, then,

$$N_n = \sum_{i=3}^{n} N_{i-1} N_{n+2-i} \tag{4.1}$$

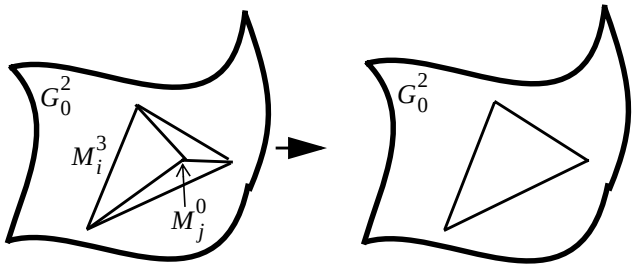


Note: figures depict only a piece of model boundary and the sliver element next to the boundary. After collapsing, only the survived faces are shown.

(a) remove M_i^3, M_j^2 and reclassify entities being brought onto boundary



(b) remove M_i^3, M_j^2, M_k^2, M_l^1 and reclassify entities brought onto boundary



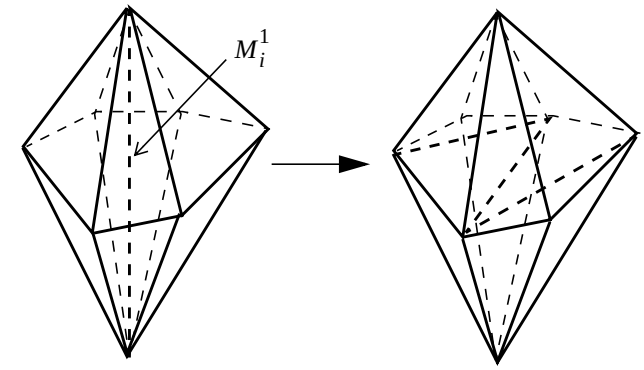
(c) remove M_i^3, M_j^0, the edges and faces connected to M_j^0 and reclassify the new boundary face

Figure 4.5 Region collapsing.

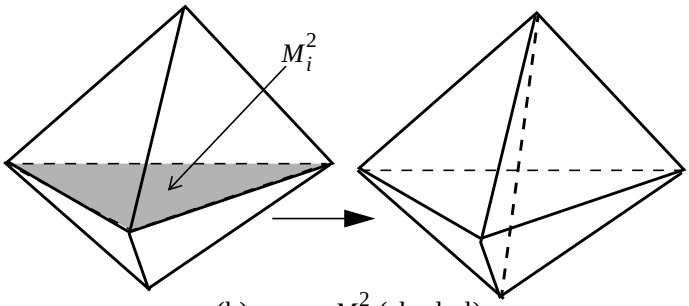
with $N_2 = 2$ [25]. Table 4.1 lists N_n as function of n. Thus, to be efficient, n has to be limited ($n \leq 6$ is used in current implementation).

Table 4.1 Number of possible triangulations.

n	3	4	5	6	7	8	9	10	11	12
N_n	1	2	5	14	42	132	429	1430	16796	58786



(a) swap M_i^1



(b) swap M_i^2 (shaded)

Figure 4.6 Swap operators.

Face swap retriangulates the cavity associated with the mesh face to delete the face and create an interior edge (Figure 4.6(b)), in other words, replace two regions by three mesh regions ($2 \rightarrow 3$ configuration). It is a process that reverses the edge swap case of $n = 3$ ($3 \rightarrow 2$ configuration).

Similar to collapsing operation, face/edge swapping can not be performed if it yields topological incompatibility or violates geometric any geometric restriction, and its application on curved boundary should consider the changes to the geometry of mesh domain.

4.1.4 Vertex Repositioning

Vertex repositioning operation relocates the position of a vertex, which changes geometry but preserves mesh topology. The cavity associated with repositioning a vertex consists of all mesh regions connected to the vertex. Vertex repositioning can not be performed if it would invert any tetrahedron in the cavity. When repositioning a vertex classified onto a model face or model edge, the new target position should always locate on that model face or model edge.

The key issue in repositioning a vertex is to determine an appropriate target location. In general, this is a constrained optimization problem in terms of a selected objective function (see for example [6,29]). In [24], de Cougny proposed an effective method that computes target location based on explicit improvement of worst shape of connected elements (referred to as explicit smoothing method hereafter). To determine a target location, explicit smoothing involves two steps: (i) determine an optimal direction that would improve the quality of the worst shaped element(s); and (ii) on this direction, define an interval of uncertainty and look for the best of the worst shape among all the connected elements. Frey and George described method similar to de Cougny's in their book (pp230-234 of [31]) by defining "ideal points" in terms of element shape (or size) criteria. One can also use Laplace method (which use the centroid of the cavity as target [24,31]) or weighted Laplace method [31,44] that is efficient in terms of speed and usually leads to "better looking" mesh, but ensures nothing.

In this thesis, de Cougny's methodology of explicit smoothing is followed and extended from two aspects to meet the needs of anisotropic mesh adaptation presented in Chapter 6: (i) respect mesh metric field. In particular, perform all geometric computation in transformed space, which includes the element quality measure computation, optimal direction computation and etc.; and (ii) in cases where the goal of vertex repositioning is to eliminate a short edge, explicit improvement of shortest connected mesh edge(s) in transformed space is used as the criteria.

For technical details of target location computation, see [24].

4.1.5 Compound Operators

There are situations where a single mesh modification operation will not yield the desired result. However, in many of these situations, the chaining of a small number of these operators, which can be easily identified and evaluated, will yield the desired result.

Four compound operators are found most useful:

- Double split plus collapse operator;
- Split plus collapse operator;
- Split plus reposition operator;
- Combination of an edge collapsing and edge swapping(s).

Double split collapse operator is aimed to eliminate sliver tetrahedra characterized by very small volume without short bounding edges and small bounding faces. It can be further considered when simple swap(s) fails. Figure 4.7 depicts such a sliver tetrahedron and illustrates reducing this tetrahedron into four mesh faces using the double splits and a collapse operation. As

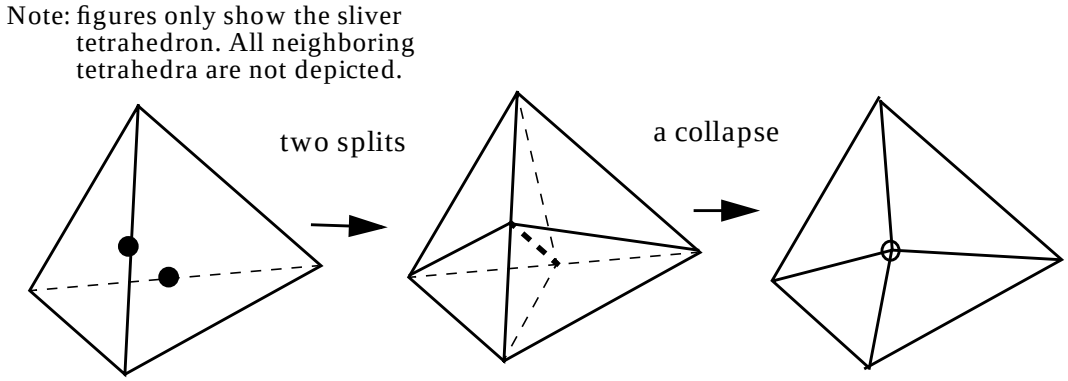


Figure 4.7 Double split collapse compound operator.

shown, given a mesh region and a pair of its opposite edges (indicated by black bullets), the operation first splits the pair of opposite mesh edges, then collapse the just created interior diagonal edge. Note that, the double split collapse operation introduces a new vertex (indicated by the circle in Figure 4.7). In case this new vertex is classified onto a curved boundary, it has to be placed onto where it is classified [50].

Split collapse operator is designed to eliminate sliver mesh faces characterized by a very small area without short bounding edges. Also it can be further considered in case swap operation fails. As depicted in Figure 4.8, given a sliver face and the longest mesh edge that bounds the sliver face (indicated by black bullet), the compound operation first splits the indicated edge, then collapse the new interior edge by removing either end vertex. It can be seen that this operation reduces the sliver face into two mesh edges. In case where the vertex indicated by black bullet survives, it needs to be placed onto its classified model boundary if it is on curved boundary.

Split plus reposition operation is another alternative to eliminate sliver tetrahedra. Given a mesh face or a mesh edge, the operation splits the edge or face, then relocate the new vertex. Split plus reposition operation can be seen as a generalized edge/face split operation in the sense that the new vertex created has the same classification as the mesh face/edge, but does not reside on the face/edge.

The goal of combined swap(s) and edge collapse operation is to collapse an edge. Swap operations are needed to make the edge collapsing successful or make up the dropped element quality. The input of the compound operation is the configuration of collapsing an edge (an edge and one of its bounding vertex to be removed), while swap(s) are figured out by the compound operation itself using the same technique described in Section 5.4.2. Simply speaking, to identify swap operation(s), it first determines the flat tetrahedron the edge collapsing would produce, then

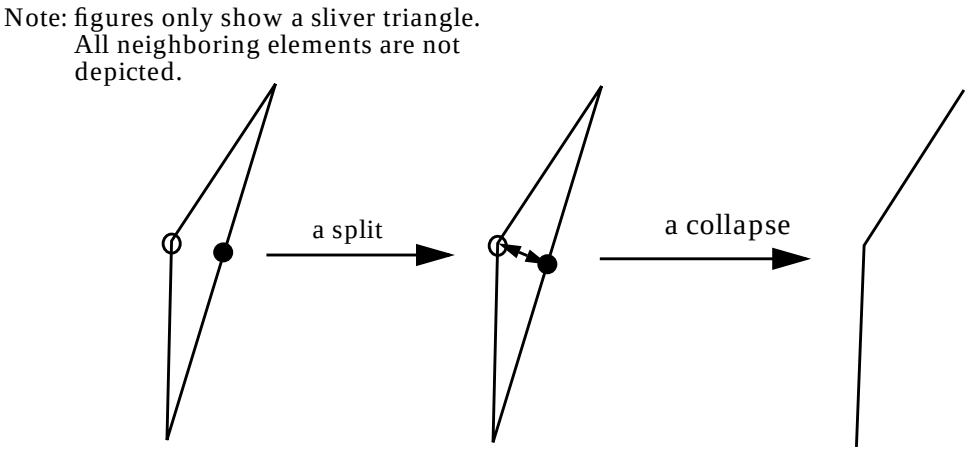


Figure 4.8 Split collapse compound operator.

identify the key mesh entities to eliminate the flat tetrahedron (or tetrahedra) incrementally, i.e., the tetrahedron becomes flat earlier is eliminated first. The compound operation fails if any flat element can not be eliminated. Figure 4.9 gives a 3D example to clarify the determination of this compound operation, where vertex M_i^0 can not be collapsed to M_j^0 since M_k^3 would become flat. By investigating M_k^3, it is easy to determine the modification that swaps edge M_l^1 to eliminate M_k^3. Thus, the compound operation that swaps M_l^1 plus collapsing out M_i^0 solves the problem.

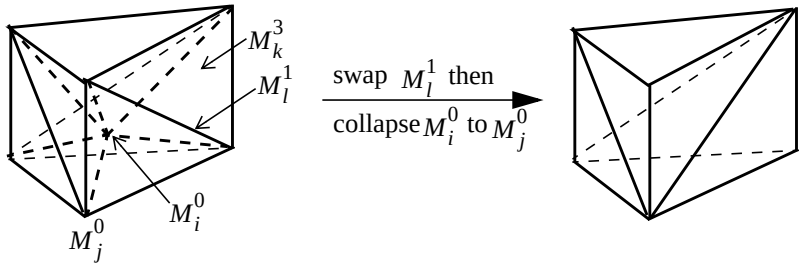


Figure 4.9 3D example of swap plus collapse operator.

4.2 Evaluation

The evaluation of local mesh modifications serves two purposes:

- Ensure satisfying topological and geometric restrictions;
- Guide the determination of local mesh modifications.

4.2.1 Topological and Geometric Restrictions

The basic topological restriction requires that the mesh after local mesh modification must be topologically compatible with the model topology the mesh is classified on. Definitions and examples of compatibility between mesh and its classified model have been discussed in references [81,77]. Figure 4.10 depicts an example to illustrate one possible topological incompatibility edge collapse operation may cause. In the example, mesh edge M_i^1, M_j^1 can not be collapsed since they bridge model edge G_i^1 and G_j^1. Note that, this paragraph just gives a simple introduction of topological restrictions and, in current implementation, the topological restriction check for all simple step operations is taken from previous works.

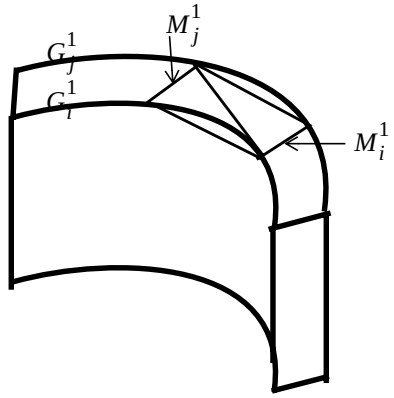


Figure 4.10 Illustration of topological incompatibility to be prevented.

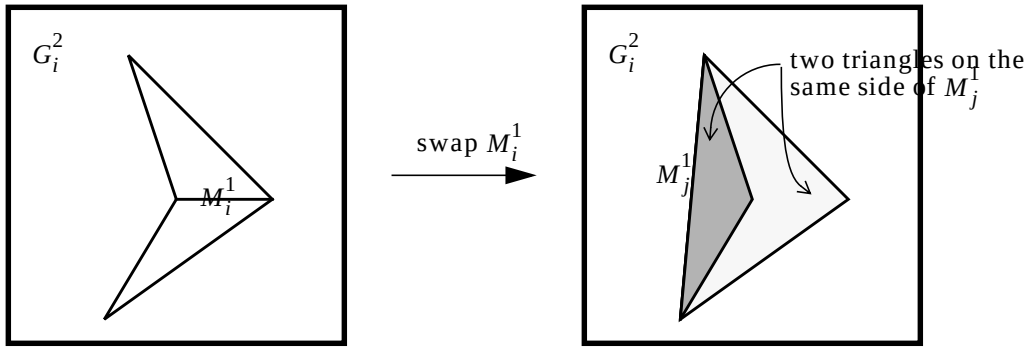


Figure 4.11 2D illustration of geometric invalidity to be prevented.

The geometric restriction of tetrahedral mesh requires that, for any face that bounds two regions in the mesh, the two mesh regions must locate on different side of the face and both are not in zero volume. Figure 4.11 depicts a 2D example where an edge swap operation violates the geometry restriction. The left figure depicts only the two triangles in the mesh that the horizon

edge M_i^1 bounds, one on top side and another on bottom side. The right shows the mesh of swapping M_i^1 into M_j^1 that violates the geometric restriction since the two new triangles (both shaded) lie on the same side of M_j^1.

4.2.2 Predicting the Local Mesh Configuration to be Created

In many cases, several mesh modifications are possible and would yield different result. Thus, to determine the most appropriate one, the capability of evaluating the interested quantity of the local mesh configuration to be yielded is needed. For example in the mesh adaptation described in Chapter 6, two quantities are of interest and need to be predicted in selecting local mesh modifications:

- the maximum length of new mesh edges in the transformed space, and
- the worst element quality in the transformed space.

Since, when local mesh modification replaces one cavity triangulation with another, the given information can uniquely determine the new triangulation, or only a few new triangulations are possible, the quantity of interest can always be predicted by constructing a simple data structure using existing information of the current mesh. Figure 4.12 illustrates this idea using an edge collapse operation that removes vertex M_d^0. To evaluate any quantity related to new mesh edges to be created, the constructed data structure is simple a list of mesh edges (the list consists of mesh edge M_0^1, M_1^1 and M_2^1 in the depicted 2D example) with the position of M_d^0 replaced by the position of M_r^0. To evaluate element level quantity, the data structure is simple a list of mesh region (the four shaded triangles in the depicted example) again with the position of M_d^0 replaced by that of M_r^0. The data structure for evaluating different local mesh modifications are different. For instance in case where edge swap is evaluated, the data structure would better be a list of vertices that indicates the new edges (or regions) to be created; in case where region split is evaluated, the data structure could simply be four vertices of the region to be split, each can be replaced by the given location to split, and etc.

Note that it is not permitted to modify any mesh topology in evaluation since what data have been attached to mesh entities is unknown at the local mesh modification level. Furthermore, to support anisotropic mesh adaptation and infinite possible measures of element quality, the evaluation process should be able to account for an abstract mesh metric field definition and an abstract element quality measure definition. Figure 4.13 shows the relation between mesh size field, element quality computation and the evaluation of local mesh modifications.

Also note that, although it is acceptable to evaluate a small set of possible local mesh modifications to obtain the most appropriate one, determining a local mesh modification based on

extensive use of evaluations must be avoided since it can severely reduce efficiency. Effective selection of local mesh modification is one of the key issues discussed in the two chapters that follow.

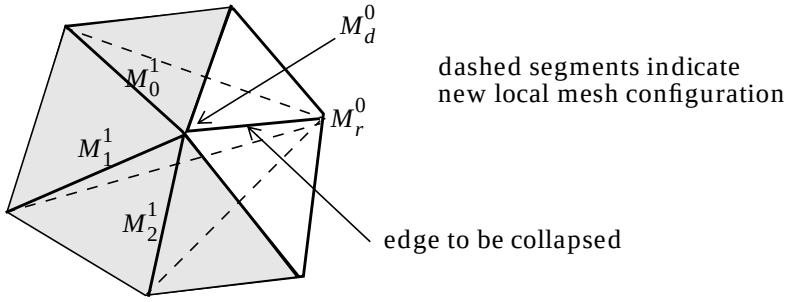


Figure 4.12 Example to show the data structure in evaluating edge collapse.

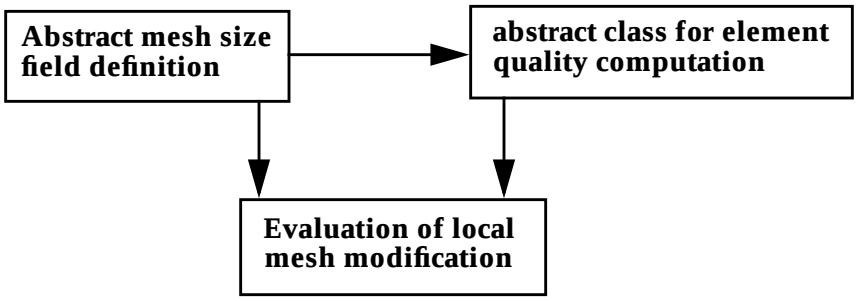


Figure 4.13 Relation between evaluation, size field and quality computation.

4.3 Implementation

All local mesh modifications discussed earlier are implemented in an object oriented methodology. Previous works, including the execution and topological restriction checks of single step local mesh modifications, are updated to support a callback mechanism[1] and incorporated into the implementation as specific member functions. The capability of evaluating the result

1. A callback mechanism has been developed to allow users to handle their data related to the mesh entities in the cavity when the mesh modification is executed. It temporally stops the execution of local mesh modification, and passes both elements to be modified and elements to be created to user.

mesh configurations in terms of abstract mesh size field and element quality measure are added for the full set of local mesh modifications again as specific member functions.

This section first introduces an abstract class of element quality measure (Section 4.3.1) so that the evaluation of local mesh modifications can be general, i.e. independent of specific quality measure. An overview of these local mesh modification objects and their relationship are then given in Section 4.3.2.

4.3.1 Abstract Class of Element Quality Measure

Since there are many ways used to measure element quality (see for example Section 2.2.3.2) and, to be general, the evaluation of local mesh modifications should not depend on any specific shape measure, an abstract element quality measure class is designed to be wrappers to be all functionality that the evaluations of local mesh modifications need. Figure 4.14 shows the abstract element quality class and its relation with other classes. The role of the abstract class is similar to that of abstract mesh size field class (see Section 3.1).

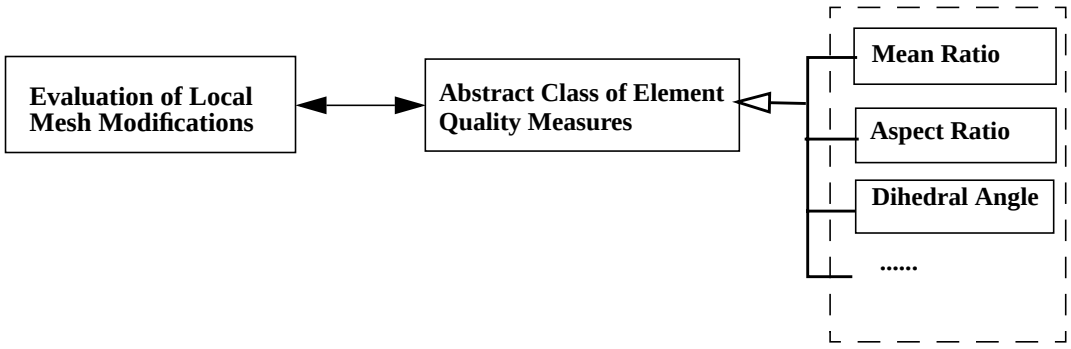


Figure 4.14 Abstract element quality class and related classes.

The definition of this abstract class, “QualityBase”, is given in Figure 4.15. The only functionality of the abstract class is to compute a value in terms of the definition of an element. It has assumed that (i) the returned value was scaled onto the interval of [0,1] with zero be the worst element quality value and one be the best; and (ii) the bigger the value, the better the element quality. Also note that, a pointer to the abstract mesh size field class is the member data of the class. In case the pointer is non-zero, the returned value will be computed in the transformed space the mesh size field defines.

4.3.2 Local Mesh Modification Objects

The class hierarchy for local mesh modification objects is shown in Figure 4.16. The base class *LocMeshMod* wraps around the common data and functionality of all local mesh modifica-

```
class QualityBase {
protected:
    // the pointer to mesh size field
    SizeFieldBase *sizeField;
public:
    // given a region, compute the quality
    virtual double R_shape(pRegion);
    // given mesh size and a set of points that define a region, compute the quality
    virtual double XYZ_shape(double *[3], pMSize);
};
```

Figure 4.15 QualityBase class.

tions. Derived from *LocMeshMod* are the current implementation of each local mesh modification object. *EdgeSplitMod* is the edge split object, *DSplitClpsMod* is the compound operator that splits two opposite edges of a mesh region then collapse the new interior edge, and etc.

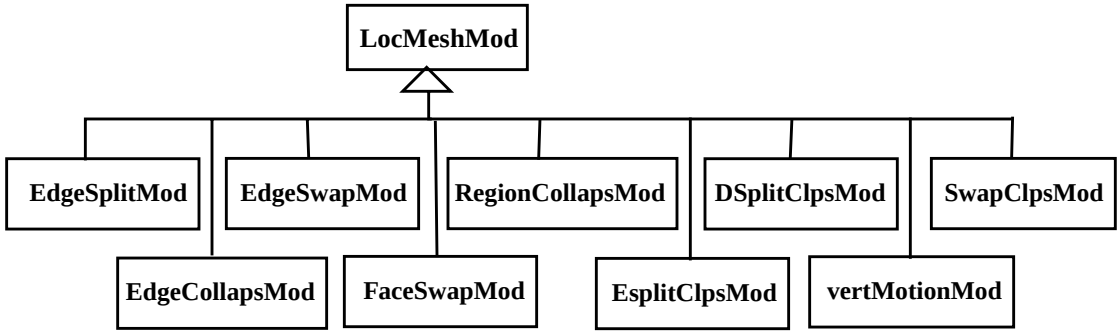


Figure 4.16 Local mesh modification derivation.

The important parts of the *LocMeshMod* class are shown in Figure 4.17. There are three pieces of information common to all local mesh modifications: a pointer to the abstract class of the mesh size field; a pointer to the abstract class of element quality measure; and the callback function to be activated in case a local mesh modification is applied. The virtual functions shown here are valid for all local mesh modifications: inquiring the type of the local mesh modification, check the topological and geometric validity of the local mesh modification, predict the element quality and size information, get the elements that fill the cavity (before and after the execution) and execute the operation.

Each derived class adds additional data and functionality that specifically define an operation. For edge split operation, this is an edge to be split and a location to add the new vertex; For

```
class LocMeshMod {
protected:
    // mesh size field
    SizeFieldBase *pSizeField;
    // element quality measure
    QualityBase *shpMeasure;
    // callback function to activate in execution
    CBfunc function_CB;
public:
    // get the type of current local mesh modification
    virtual int type();
    // check if the local mesh modification will violate topological restrictions
    virtual int topoCheck();
    // check geometric restriction and return the value of worst local quality if requested
    virtual double geomCheck();
    // compute the maximum/minimum edge length of the new triangulation
    virtual void sizeCheck(&min, &max);
    // get a list of regions that define the cavity
    virtual pPList getCavity();
    // execute the local mesh modification
    virtual int apply(pMesh);
};
```

Figure 4.17 Base class of local mesh modifications.

edge collapse operation, this is an edge to be collapsed and the end vertex of the edge to be removed; For edge swap operation, this is simply an edge to be swap; For face swap, this is simple a face; For region collapse, this is a region; For split collapse compound operation, this is a face and one edge that bounds the face; For double splits collapse compound operation, this is a pair of opposite edge in a mesh region and the position to add the new vertex; For vertex reposition operation, this is the vertex to be moved; and for swap(s) collapse operation, this is an edge to be collapsed and the end vertex of the edge to be deleted.

CHAPTER 5

Accounting for Curved Domains

5.1 Introduction

Mesh enrichment by element subdivision, commonly referred to as h-refinement, of tetrahedral elements has been well studied in the context of polyhedral geometry domains (see for example [7,13,23,54,76]) when the domain of the mesh and the geometry is identical for all meshes. In the case of curved domains approximated by straight-sided elements, one can apply standard refinement procedures which split the edges of the initial mesh thus employing that initial geometric approximation through the mesh enrichment process. Although, the results of such a practice may be acceptable in some situations, the solutions obtained are that of a different geometry than the original problem and the solutions to the two problems can be quite different, particularly with respect to local parameters of interest like the stresses in a mechanical part.

The alternative is to use the knowledge of the actual geometric domain during the h-refinement procedures to project the new vertices onto the correct geometry thus ensuring the results converge to the solution on the correct domain. The complexity in this approach is that although the process of projecting the vertices to the curved boundary, to be referred to as snapping in this chapter, is straight forward for most of the refinement vertices on the curved model boundaries, it is common for this process to produce invalid elements for a number of the vertices. Consider Figure 5.1 for such an example. In this case a mesh edge bounding a portion of a circular hole is selected for refinement. Figure 5.1(b) shows just the refined element and the neighboring element that contains the point the new vertex should be snapped to for it to be placed properly on the boundary of the hole. As Figure 5.1(c) shows, the process of snapping the new vertex will invert one of the elements (the shaded triangle) yielding an invalid mesh.

Jones [40] and De Cougny [23] have pointed out the need to project refinement vertices to curved geometries and discussed the application of mesh modification when the projection process would yield invalid elements [23]. However, the procedures presented to date are limited to the ad-hoc application of specific operations with no analysis of ensuring the ability to introduce improvement in the geometric approximation [23]. Thus the current practice is to either not snap the vertices to the curved boundary, or to snap those vertices that easily project and leave any remaining vertices unsnapped. This chapter presents a procedure for snapping vertices to the curved geometry that is based on a more detailed analysis of the modifications that need to be applied for the vertex to move toward the boundary and included operations that can always be applied to yield an improved geometric approximation.

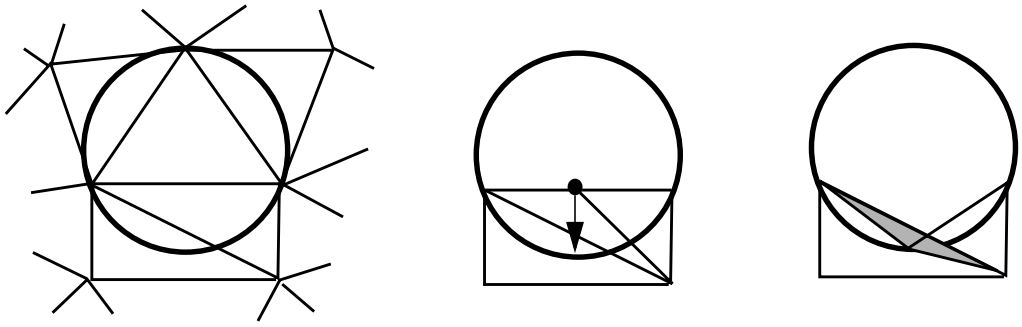


Figure 5.1 Snapping a vertex that creates a geometric invalidity.

Section 5.2 presents an overview of the entire procedure which includes two stages. The first stage applies local mesh modifications such that the vertex being snapped is moved at least as far as the plane that causes the first element to become invalid. Section 5.3 indicates why it is critical to apply modifications that ensure vertex movement to at least this “first problem plane”, while Section 5.4 discusses the determination and application of local mesh modification operations. The second stage of the procedure is applied when the limited number of mesh modifications considered in the first stage do not complete the movement of the vertex to the boundary. Although it is clear that the addition of increasingly complex local mesh modifications would allow more vertices to be snapped to the boundary, there is no clear proof of the minimal set of such operators that will ensure progress on snapping the vertex. Therefore, the second stage applies a cavity retriangulation procedure (explained in Section 5.5) that ensures progress. Section 5.6 demonstrates the application of the procedure on example problems. Note that although throughout the description of the procedures figures of 2D examples are provided to explain specific concepts, all the procedures described have been fully implemented for 3D tetrahedral meshes.

5.2 Overview of the Vertex Snapping Procedure

The snapping algorithm is incorporated within a *h*-version mesh adaptation procedure. It deals with placing the refinement vertices created on mesh edges or faces that lie on curved model boundaries onto those curved boundaries. For computational efficiency, the procedure consists of two stages. Given a list of vertices to be snapped, the first stage snaps those vertices that can move directly to the boundary with no problem and those that can be addressed by application of simple local mesh modifications. The second stage addresses the small number of vertices that the first stage does not address through the application of a local retriangulation process.

Figure 5.2 gives the overall pseudo-code for the first stage of the procedure. For any vertex in the list, this stage first tries to snap it by a repositioning operation. If repositioning is not successful, a set of local mesh modification operations are considered with the goal of moving the vertex at least as far as the first plane where a connected element in the original mesh would have become invalid. If a local modification is found that allows the vertex to be moved to that plane it is performed and the vertex is moved onto that plane. Remove the vertex from the list if it has reached the boundary, otherwise leave it in the list. Process the next vertex in the list. Since vertices in the list may be snapped incrementally and topologically neighboring vertices may interact with one other, this stage will traverse the list repeatedly until it is empty or none of the remaining vertices can be further moved by the available modification operators.

```
While any vertex in the list still can be moved forward
   determine a target move for the next vertex in the list
   if the next vertex can be moved to target location without a problem, then
      move to that location and remove the vertex from list
   else
      determine the first problem plane preventing the motion
      determine the best local modification to move the vertex to at least its first problem plane
      if such a local modification is acceptable, then
         perform this modification and remove the vertex from list if it is now on boundary
      end if
   end if
end while
```

Figure 5.2 Pseudo-code to snap vertices by repositioning and local modification.

The second stage is required only in the case when the list is not empty after the first stage meaning that some vertices are not yet on the appropriate curved boundary. In the second stage the remaining vertices in the list are addressed through the application of a cavity construction and triangulation algorithm. If any new boundary vertices are created during cavity triangulation that need to be snapped, they will be inserted into the list and processed. As discussed in Section 5.4, any such new vertices are on a new local boundary triangulation that is an improved approximation to the curved geometry. It is possible to apply the local triangulation process so long as the cavity size has not degraded to the point where there are numerical precision problems. Since the cavity triangulation procedure does have the flexibility of introducing new vertices on the cavity boundary when required, the process is not provably guaranteed to converge in a fixed number of steps. In the large number of test cases performed the procedure has always converged in a small number of steps (Section 6 presents more information on this) due to the fact that

the cavity mesher only infrequently introduces new vertices on the boundary and those vertices are closer to the curved geometry than the original refinement vertex that forced the application of cavity triangulation.

Since the goal of the vertex snapping process is to place each new boundary vertex on the appropriate boundary such that a valid mesh of well shaped elements is maintained, there is not a unique target location to be used for that process. At any point in the process of moving a vertex to the boundary the target location is a current suggested location that is selected to be placed such that the mesh entities connected to it have appropriate sizes. One target location that is efficient to generate is the average parametric value as depicted for a simple two-dimensional case in Figure 5.3. In this case the half parametric value is very near the half arc-length yielding well shaped elements. Cases when the parametric space is distorted such that the average parametric value is not a good selection, edge cord lengths are used as approximations to the arc length to estimate and adjust parameter value to be used. This process will repeat until the difference in the edge cord lengths is not too large. Note that when a mesh modification is performed and the mesh vertex is only moved partway toward the model boundary, it is necessary to recalculate a new target location to account for the new mesh topology.

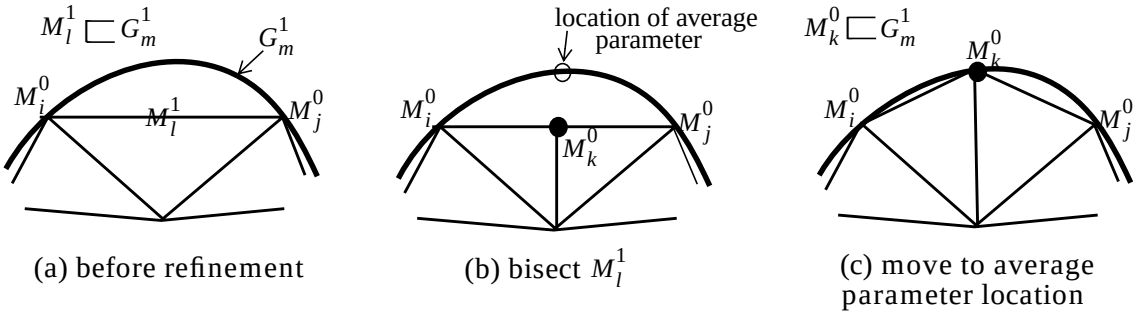


Figure 5.3 An example of target location calculation.

5.3 The Need to Move at Least to the First Problem Plane

A critical aspect of the selection and application of mesh modification operations is ensuring that a selected mesh modification is making appropriate progress toward the goal of placing the refinement vertex on the model boundary. As indicated in this section, the basic requirement for this is ensuring that the mesh vertex moves at least to the first problem plane.

5.3.1 Definition of the "First Problem Plane"

Definition 5.1. A mesh face is “opposite” to a mesh vertex with respect to a tetrahedral mesh region if the mesh face and mesh vertex are on the closure of the mesh region, but the mesh vertex is not on the closure of the mesh face.

Definition 5.2. A “problem plane” with respect to a mesh vertex to be snapped to a given target location is the plane defined by the opposite mesh face when the vertex and the target location are on opposite sides of the plane. The “first problem plane” is the problem plane that the vector from the vertex to be snapped to the given target location intersects first.

Figure 5.4 demonstrates the concept of first problem plane when snapping vertex M_0^0. As indicated by dashed lines, there are five potential problem planes, P_1 through P_5, one each for the elements connected to M_0^0. For this example the problem planes are P_1 through P_3 with the first problem plane being P_1. Figure 5.5 indicates situations where the first problem plane corresponds to more than one opposite face because the point of intersection with the planes of those faces are coincident. Figure 5.5(a) depicts a coplanar situation, where the first problem plane contains four mesh faces opposite to M_0^0 (the four shaded triangles). Figure 5.5(b) shows a case where there are two problem planes that become active at the same time. In this case either plane can be considered the first problem plane.

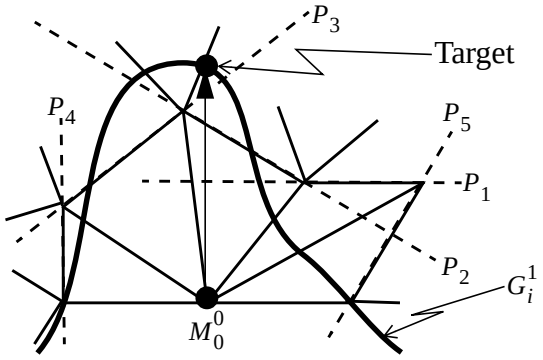
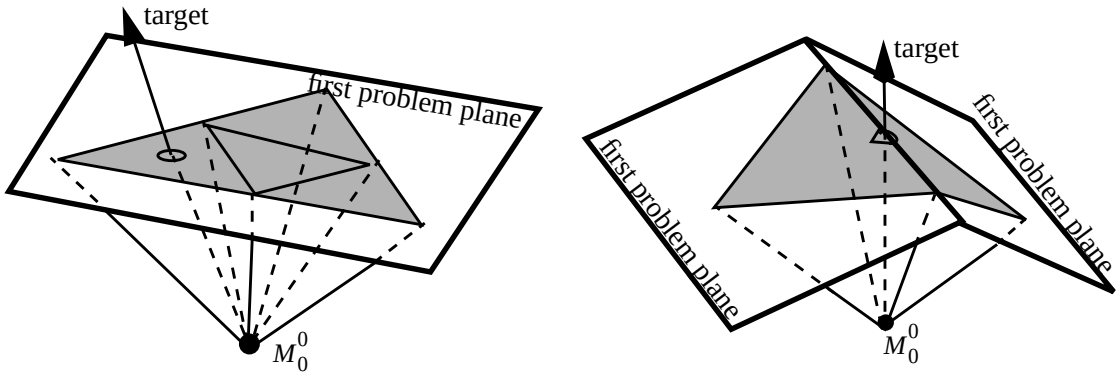


Figure 5.4 A 2D example of the first problem plane.

5.3.2 Need to Pass the First Problem Plane

In many cases, it can take more than one step of mesh modification to snap the vertex to the boundary. In those cases it is important to realize that it is critical to move the vertex at least as far as the first problem plane to avoid situations where the process continues to take ever smaller steps.



(a) co-planar (b) not unique

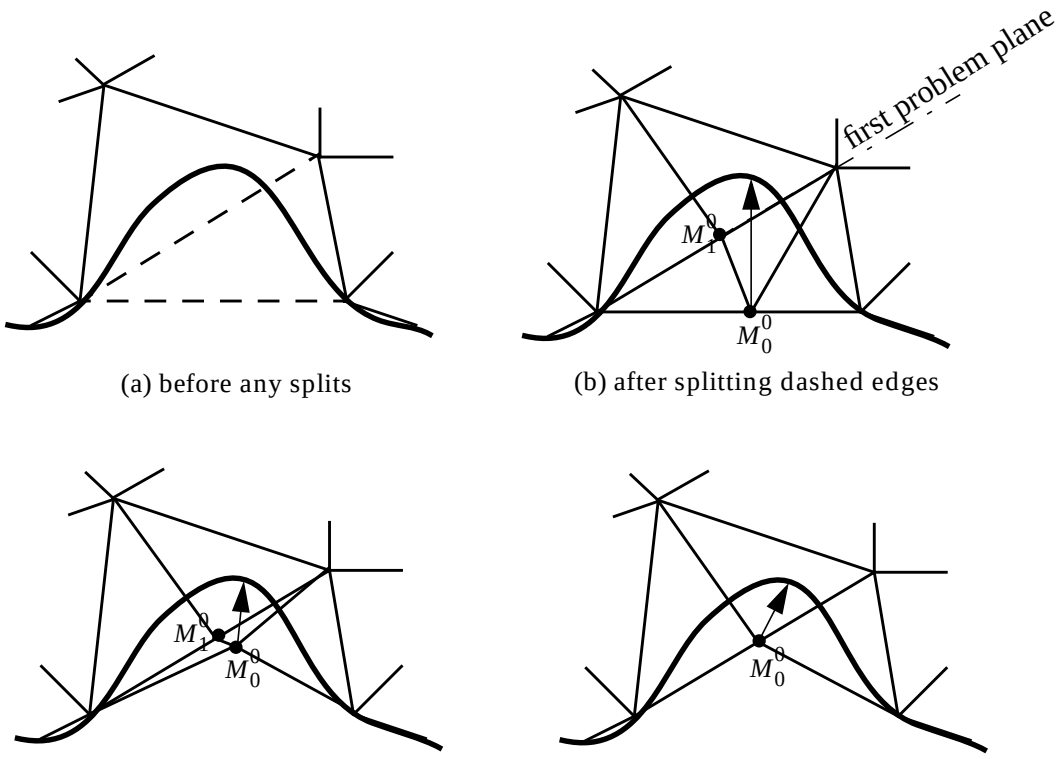
Figure 5.5 Two specific cases of first problem plane.

The basic requirement is “any local modification applied to snap a mesh vertex must ensure that the vertex is moved at least to its first problem plane”. The requirement is justified through the simple example depicted in Figure 5.6. Figure 5.6(a) indicates the edges to be split as dashed lines, while Figure 5.6(b) shows the mesh after splitting as well as indicating the boundary mesh vertex, M_0^0, that needs to be snapped. Figure 5.6(c) shows a movement of M_0^0 that does not pass the first problem plane. Clearly, poorly-shaped elements are created. More importantly, a strategy that allows incremental mesh modification without getting to at least the first problem plane can lead to many increments yielding ever decreasing local mesh quality. On the other hand, performing an operation that requires moving to at least the first problem plane ensures the elimination of the first element that would have its shape continue to degrade. Figure 5.6(d) shows the result after collapsing M_0^0 to M_1^0, which gets M_0^0 onto its first problem plane.

5.4 Application of Local Mesh Modifications

The ability to move a vertex to at least the first problem plane requires the application of local mesh modifications. This section discusses the selection and application of appropriate local modifications when a simple vertex repositioning to a target boundary location does not succeed.

The determination of the appropriate local mesh modification includes (i) determination of a set of locations on the first problem plane and / or a target location on the boundary, and (ii) the determination if there are local mesh modifications to make the move acceptable. When the motion to a location on the boundary will cause one or more elements to become invalid, the procedure focuses on moving the mesh vertex onto the first problem plane. The selection of the



(a) before any splits (b) after splitting dashed edges

(c) not reach the first problem plane (d) get to first problem plane

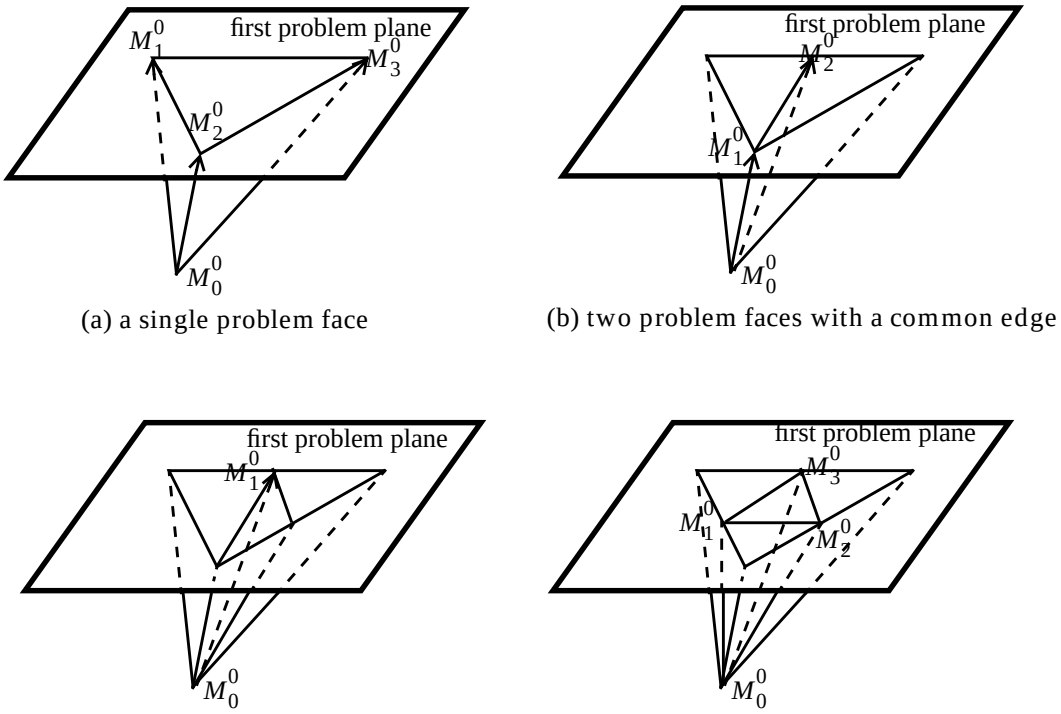
Figure 5.6 2D example to show the need to move past first problem plane.

desired location on the first problem plane is focused on which location provides the most advantageous selection with respect to the resulting mesh. For example, using the location of an existing mesh vertex of an appropriate mesh face on the first problem plane often easily accomplished by a simple collapse operation which does not have an adverse effect on the shapes of the resulting mesh.

5.4.1 Collapse to Mesh Vertices on First Problem Plane

The first local modification considered is edge collapse in which the vertex to be snapped is collapsed to a location of an existing mesh vertex on the first problem plane.

Figure 5.7 shows how the edge collapse can be determined with respect to snapping vertex M_0^0. Figure 5.7(a) depicts a situation where there is only one problem face on the first problem plane. In this case, three edge collapses, collapsing M_0^0 to M_1^0, M_2^0 or M_3^0 (indicated by arrows), are possible. Of the possible collapses that maintain mesh validity the one yielding the best element quality is selected. Figure 5.7(b) depicts a situation of two problem mesh faces with a common mesh edge, where two edge collapses, collapsing to M_1^0 or M_2^0, are possible. If both collapses



(a) a single problem face (b) two problem faces with a common edge

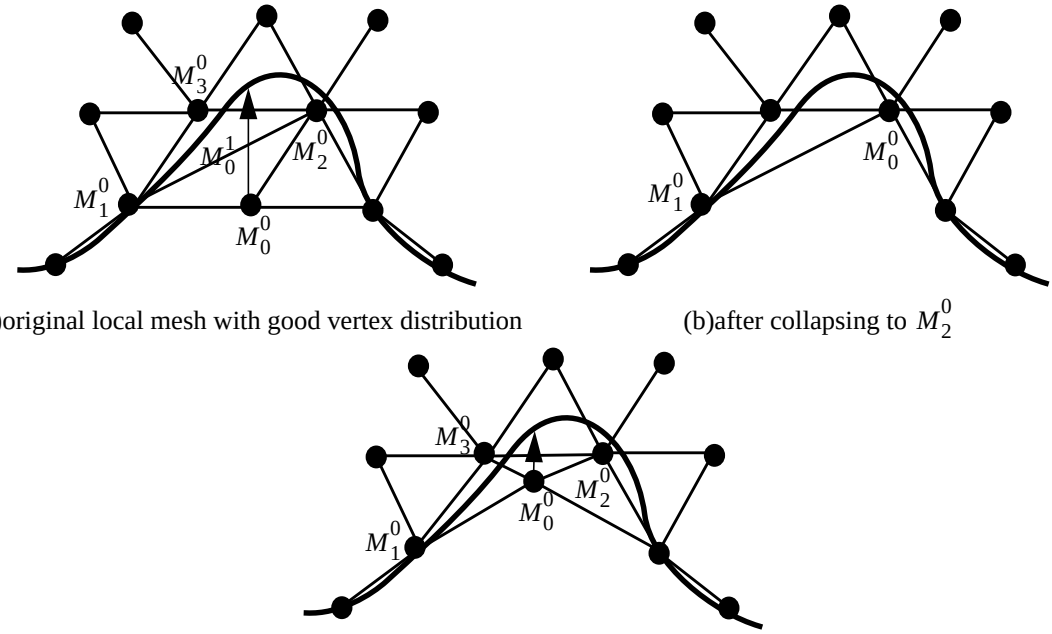
(c) three problem faces with a common vertex (d) four problem faces without a common vertex

Figure 5.7 Edge collapses pulling M_0^0 onto its first problem plane.

maintain mesh validity the one yielding the best element quality is selected. In the situation of multiple problem faces with a common mesh vertex as shown in Figure 5.7(c), if possible, collapsing to the common vertex is the choice. Figure 5.7(d) illustrates a situation without a common vertex. In this case no single step operator can move M_0^0 onto its first problem plane. However, consideration of a compound operator in which M_0^0 is collapsed to an existing mesh vertex on first problem plane, in conjunction with other local mesh modifications that eliminate the tetrahedra that would become flat after that edge collapse is often possible.

A key reason why collapsing to a vertex on the first problem plane is favored is that it does not create or move any vertices. Therefore, most neighboring elements are not affected and thus maintain their current shape quality and size. Although a collapse operation will alter the size of affected elements, the situation under consideration is one where the vertex is being requested to move into the mesh far enough to have element volumes go to zero. Therefore, the collapse usually does not increase the size of remaining elements past the size being requested by the adaptive procedure. Performing other mesh modifications without a collapse included is typi-

cally computationally more expensive and by themselves often do not yield a situation where it becomes possible to move the vertex to the first problem plane.



(a)original local mesh with good vertex distribution

(b)after collapsing to M_2^0

(c) after swapping M_0^1 and moving to a location that is not an existing vertex

Figure 5.8 Example to show why collapsing to an existing vertex is preferred.

Figure 5.8 demonstrates the issue just discussed with respect to a 2D example. Figure 5.8(a) shows the original local mesh and the vertex to be snapped, M_0^0, while Figure 5.8(b) shows the mesh after M_0^0 is collapsed to M_2^0. Note that this collapse operation effectively deletes two elements that were outside the domain the mesh represents and did not alter the size or shape of any remaining elements. Placing M_0^0 on the boundary from this location now requires only a small motion. On the other hand, Figure 5.8(c) shows the resulting mesh after swapping M_0^1 and moving M_0^0 to a location on its first problem plane but not an existing vertex. Three elements have had their size and shape changed and the vertex M_0^0 must still be moved a substantial distance and pass a problem plane to be placed on the boundary. Although the example of Figure 5.8 was constructed to demonstrate specific points, study of the application of different ordering and variants of the mesh modification operators to snap vertices created by adaptive 3-dimensional refinement confirms the importance of first focusing on collapsing to a vertex on the first problem face in terms of the speed of the procedure and quality of the resulting mesh.

5.4.2 Application of Additional Local Mesh Modifications

Although collapsing to an existing vertex is the preferred first mesh modification to consider, there are situations where alternative mesh modification operations either eliminate the problem plane or allow motion to the problem plane. Often these are compound operations that include a collapse. Therefore, in cases where a simple edge collapse is not satisfactory it is useful to consider other mesh modification operations. Although there are a large number of specific combinations of mesh modifications that can be attempted, consideration of a large number of combinations showed a relatively small number have a high enough success rate to justify the computational effort required. Including more than this subset tends to decrease the overall efficiency of the process with no increase in robustness since the vertices not addressed with mesh modification are addressed by cavity meshing.

In cases where the application of an edge collapse is not successful because elements become invalid, it is straight forward to determine if the configuration of those elements is such that they could be eliminated by the efficient region collapse operation.

Application of edge or face swap is a possible means to eliminate problem face. Figure 5.9 shows a simple example of such a situation where snapping vertex M_0^0 to the desired location on the boundary is precluded due to a problem face. It is possible to collapse M_0^0 to M_1^0 on the problem face. However, this is not acceptable since it undoes the requested refinement that introduced M_0^0. If instead the edge M_0^1 is swapped the problem face is eliminated and M_0^0 can be snapped to the desired location. (Note that in this example, the amount of mesh motion in moving to the desired location on the boundary does substantially increase the size of elements so that additional refinement is needed to recover the desired mesh size field.) Another operation that has been found to be potentially useful is to collapse a short edge (with respect to the requested mesh sizes) that is attached to the problem plane and thus eliminate (by deletion or movement) the problem plane.

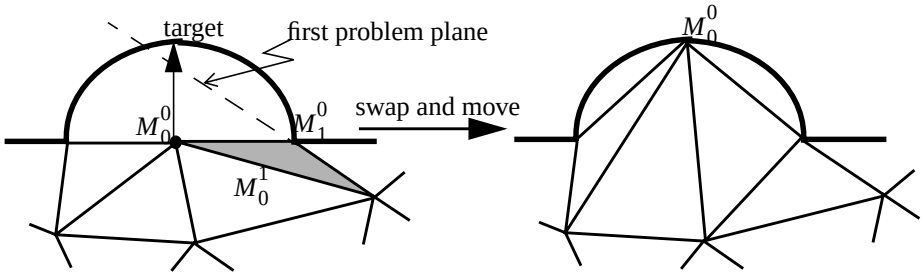


Figure 5.9 Application of a swap operation to eliminate a problem face.

There are a large number of swap operations that can be considered to eliminate a problem face. Since the level of computational effort required to examine all possible swaps is large, it is useful to use information on the desired direction of vertex motion to limit the number of swaps considered. Figure 5.10 demonstrates the simple construct used for this purpose. The plane of the problem face is subdivided into seven subareas defined by extending the mesh edges of the problem face. There is one area associated with each mesh edge of the problem face (areas I) and one associated with each mesh vertex of the problem face (areas II). If the intersection between the plane and the ray from M_0^0 to its target is in the area associated with an edge, then swapping that edge, or the edge opposite it with respect to the tetrahedron are considered. For example, in Figure 5.10 the intersection location is in the area associated with M_0^1. Therefore, swapping M_0^1 and M_1^1 are considered. In the case where the intersection is in the areas associated with a vertex, swapping the face associated with the other three vertices and the three edges bounding that face are considered.

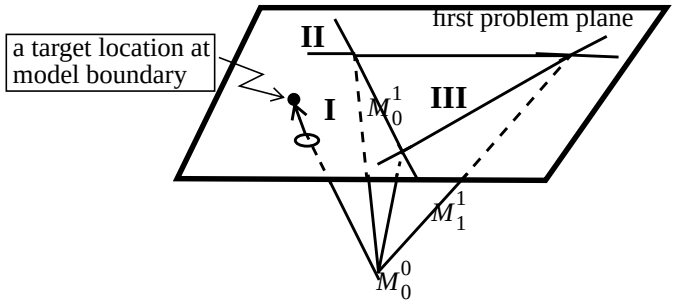


Figure 5.10 Determination of desired swap operations.

Application of splitting an edge (or two edges) and properly collapsing a vertex is a possible means to eliminate a problem face that can not be eliminated by swap operations, especially when swap fails due to topological restrictions. Figure 5.11 illustrates such a situation with respect to snapping M_0^0 in which a sliver problem face can not be eliminated by swapping M_0^1 since M_0^1 is classified on a model edge ($M_0^1 \sqsubset G_0^1$). However, if M_0^1 is split, followed by collapsing M_3^0 to the new vertex, the problem face is eliminated and M_0^0 can be snapped (note that the new vertex needs to be snapped later).

In situations where the distance between model entities is small compared to the local mesh edge lengths, mesh faces on the first problem plane can be classified on other model entities. Figure 5.12 shows such a situation in which it is clear that it is not possible to satisfactorily move M_0^0 onto G_0^1 without the elimination of M_0^1. However, application of a split of the mesh entity and snapping that new vertex to the appropriate model entity will improve the local geometric

approximation of that model entity such that there is room to snap the original vertex under consideration. The application of this process is shown in Figure 5.12.

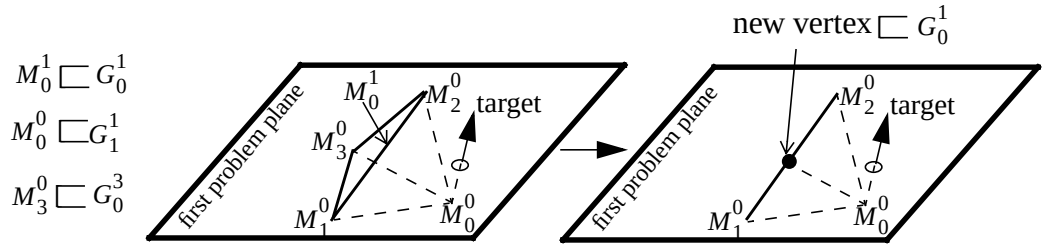


Figure 5.11 Application of split and collapse.

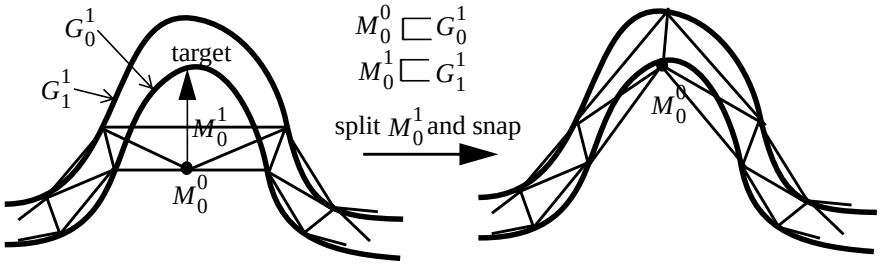


Figure 5.12 Application of splitting an edge classified on a nearby model entity.

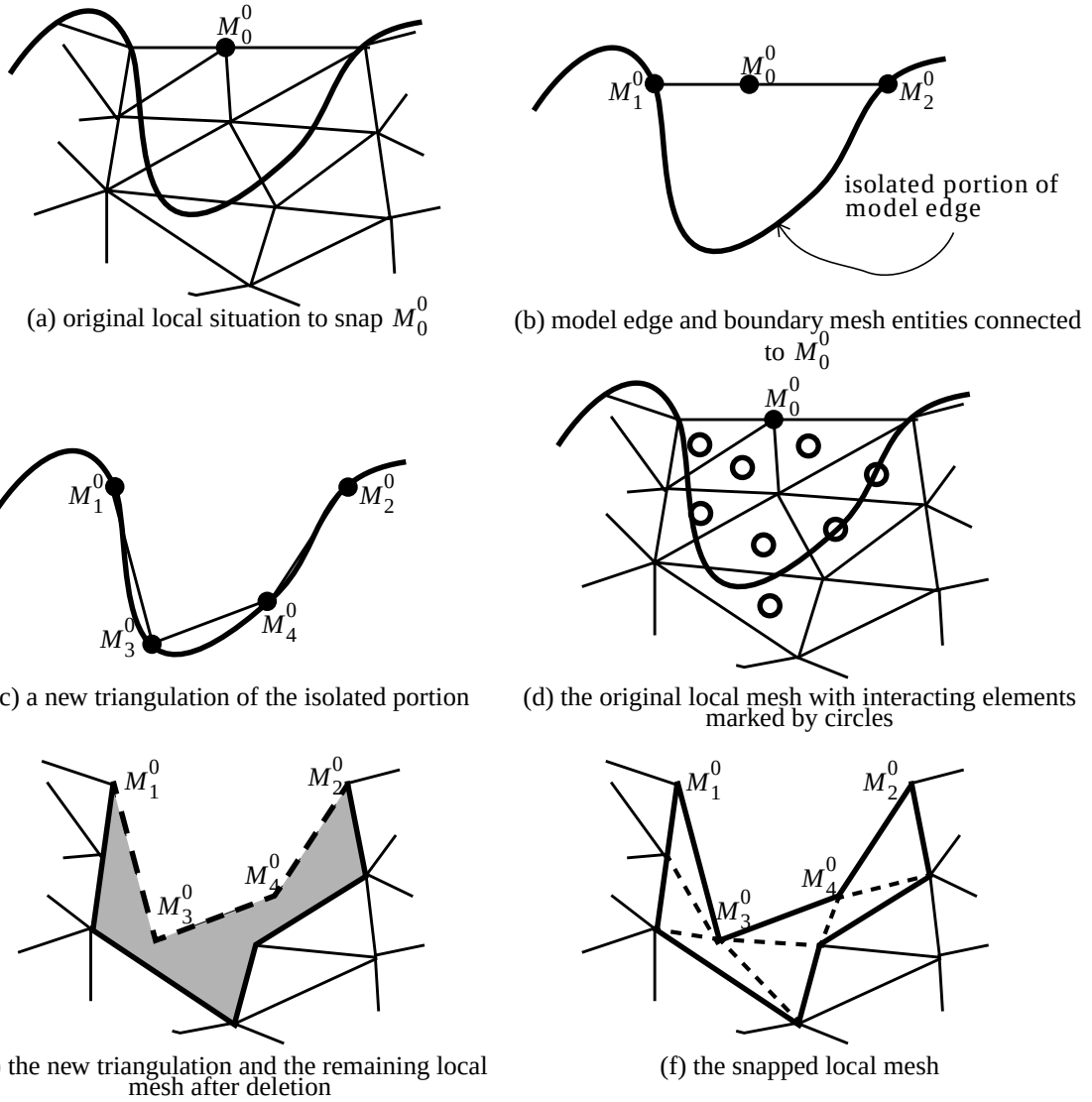
5.5 Local Cavity Remeshing Procedure

As will be demonstrated by the results presented (Section 6) the limited number of mesh modification operations described above resolve the majority, but not all, of the mesh vertices to be snapped to the boundary. It is possible to consider development of more logic to evaluate additional multiple step mesh modification operations. In theory there is such a set of mesh modification operations that can ensure the ability to snap all the vertices to the boundary. However, the inability to define an algorithm that provably ensures one can determine the needed mesh modification steps indicates an alternative procedure is needed. One such alternative is the appropriate application of a reliable cavity meshing procedure [10]. Although we do not have the proof that the application of the cavity meshing procedure ensures snapping of all vertices, it has been found to work in all cases performed to date. Conjecture as to why it is an advantageous and reliable procedure to apply is indicated below.

To understand the cavity retriangulation procedure refer to the 2D example in Figure 5.13. In this example, M_0^0 indicates a boundary mesh vertex created during refinement that needs to be snapped. Figure 5.13(b) shows the model edge and the boundary mesh entities connected to M_0^0.

The two boundary mesh vertices, M_1^0 and M_2^0, can be easily identified (since they are connected to M_0^0 through a boundary mesh edge). The first step in the application of the procedure is to introduce an alternative triangulation of the portion of the model boundary originally covered by those mesh entities that includes one or more new vertices. This triangulation is independently defined and the added mesh vertices are properly positioned on the model boundary they are classified on. As Figure 5.13(c) shows for this simple 2-D case this new triangulation consists of the three mesh edges traversing from M_1^0 to M_3^0 to M_4^0 to M_2^0. The next step is to determine the highest order mesh entities that "interact" with the piece of boundary triangulation just defined. This set of mesh entities includes those that intersect the new triangulation or are "outside" the triangulation. For our 2-D example, these mesh entities are the mesh faces with hollow circles shown in Figure 5.13(d). In the next step the interacting mesh entities are deleted from the mesh. This will create an un-meshed cavity between the newly defined piece of surface triangulation and the remaining mesh (Figure 5.13(e)). This cavity is then processed by the cavity meshing procedure [43] to create a local triangulation (Figure 5.13(f)) thus completing the process in the present example.

The subsections that follow discuss the technical aspects of this process for the three dimensional situation. Section 5.5.1 indicates the determination of mesh edges that define the portion(s) of the model face(s) to be retriangulated and Section 5.5.2 discusses their triangulation. Section 5.5.3 presents the procedure that deletes mesh entities to create an un-meshed cavity between the new triangulation and the remaining mesh. Section 5.5.4 indicates how that cavity is triangulated, discusses the possibility of creating new mesh vertices classified on the boundary of the model that will also need to be snapped and indicates how those vertices are handled.



(a) original local situation to snap M_0^0

(b) model edge and boundary mesh entities connected to M_0^0

(c) a new triangulation of the isolated portion

(d) the original local mesh with interacting elements marked by circles

(e) the new triangulation and the remaining local mesh after deletion

(f) the snapped local mesh

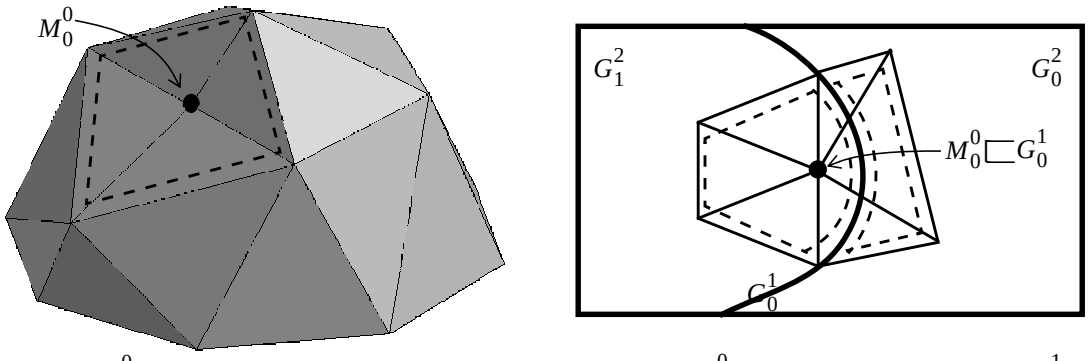
Figure 5.13 2D demonstration of cavity construction and remeshing algorithm.

5.5.1 Determination of Loop to Isolate Portion of Model Face

Definition 5.3: Given M_0^0 as a mesh vertex to be snapped, $P(M_0^0)$ is defined as the set of mesh edges opposite to M_0^0 with respect to mesh faces classified on model faces that M_0^0 bounds. A mesh edge is opposite to M_0^0 if the edge and M_0^0 bound the same mesh face and M_0^0 is not in the closure of the edge.

$P(M_0^0)$ allows us to identify loops that bound the portion of the model face that needs to be retriangulated for each model face M_0^0 bounds. For instance in the example depicted in

Figure 5.14(a), $P(M_0^0)$ consists of four edges, which form a loop (indicated by dashed lines) and isolate a portion of the semi-sphere surface; while in the example of Figure 5.14(b), $P(M_0^0)$ consists of six edges, three classified on G_0^2 and the other three classified on G_1^2, which isolate a segment of the model edge and two pieces of model faces, one piece for G_0^2 and another for G_1^2. These six mesh edges and mesh edges classified on the isolated portion of the common model edge are used to construct two loops, one for each model face.



(a) snap M_0^0 classified on a semi-sphere surface (b) snap M_0^0 classified on a model edge G_0^1

Figure 5.14 Simple examples to explain the definition of $P(M_0^0)$.

5.5.2 Generate a New Triangulation for Isolated Portion of Model Face

Definition 5.4: The new geometry triangulation of the portion of model face(s) isolated by $P(M_0^0)$ is termed $T(P(M_0^0))$.

To ensure that $T(P(M_0^0))$ improves the geometry approximation of the local portion of the model boundary and the mesh sizes match what was requested by the refinement procedure, the triangulation must create new vertices on the model boundary. Specifically, if M_0^0 is classified on a model face, the new triangulation is required to have ≥ 1 mesh vertex generated internal to $P(M_0^0)$; and if it is classified on a model edge, the triangulation requires ≥ 1 new vertex classified on the edge internal to $P(M_0^0)$ and there may, or may not, be new mesh vertices generated internal to the loops for each of the faces as desired.

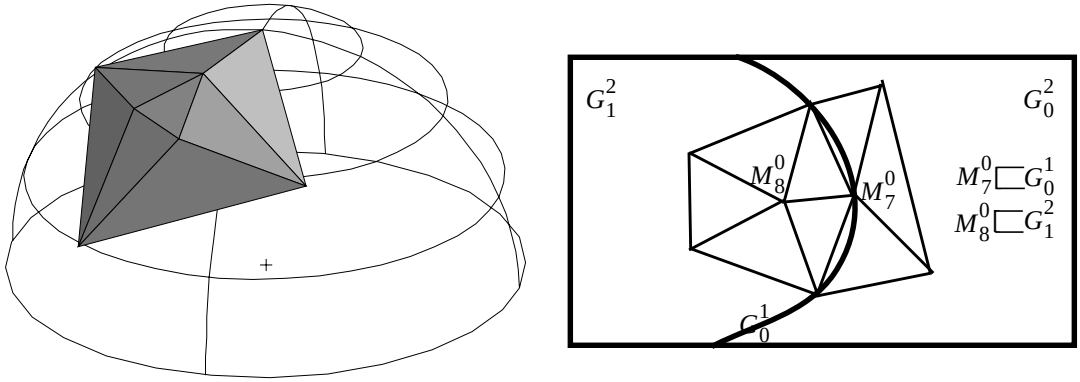
The method to generate $T(P(M_0^0))$ is described in Figure 5.15 in pseudo-code form. For a model face M_0^0 bounds, its portion to be retriangulated is isolated by a loop of mesh edges classified onto the model face closure. If M_0^0 is classified on the model face, the loop is simply defined by all edges in $P(M_0^0)$. If M_0^0 is classified on the model edge there is a loop for each face that edge bounds. For each face the loop consists of new mesh edges generated when remeshing the seg-

ment of model edge and the mesh edges of $P(M_0^0)$ that are classified on that model face or its closure. Once the portion of the model face is isolated, it is sent to a surface mesher [22] for triangulation.

if M_0^0 is classified on a model edge
    remesh the segment of model edge isolated by $P(M_0^0)$
end if
loop over the model faces M_0^0 bounds
    determine the loop of mesh edges that isolates a piece of current model face
    expand the loop if the piece of current model face can not be properly triangulated
    send the loop, the model face and requested mesh size to surface mesher for retriangulation
end loop

Figure 5.15 Pseudo-code to generate $T(P(M_0^0))$.

Figure 5.16 demonstrates the triangulation, $T(P(M_0^0))$, for the two examples given in Figure 5.14. In Figure 5.16(a) M_0^0 is classified on a model face so the new triangulation is generated simply by sending the loop and the face to a surface mesher and requesting the loop be triangulated with at least one vertex added interior to the loop. In Figure 5.16(a) three mesh vertices internal to $P(M_0^0)$ were inserted. In Figure 5.16(b) M_0^0 is classified on a model edge in which case the first step is to place at least two mesh edges on the portion of model edge isolated by $P(M_0^0)$. In this example, mesh vertex M_7^0 was added on the model edge to define two new mesh edges. A new triangulation is then generated for the isolated portion of each face bounding. For this example, there are two model faces, therefore two loops are triangulated. When triangulating the left face M_8^0 was inserted to create mesh edges of the requested length.

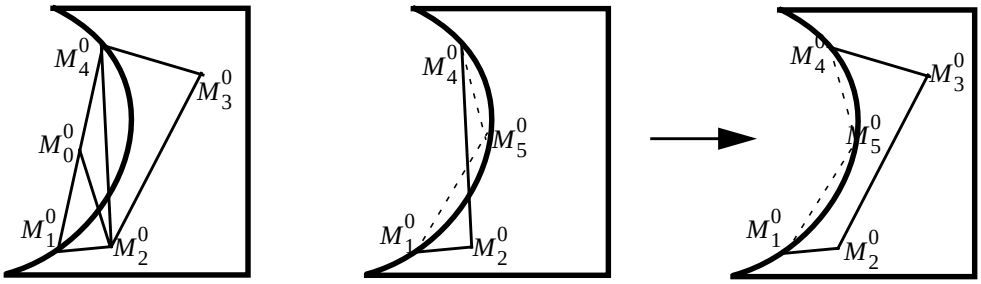


(a) classified on semi-sphere face

(b) classified on model edge

Figure 5.16 Simple examples to demonstrate $T(P(M_0^0))$.

There are two reasons why the loop of mesh edges for a portion of a face to be re-triangulated needs to be expanded. First, when M_0^0 is classified on a model edge it is possible that the loop can not be properly triangulated. The source of this potential problem is the fact that the geometric approximation of the edge is required to be locally improved. Explaining this problem and its solution is easiest in the case where the surface triangulation is done in a projected parametric coordinate system. Consider the example shown in Figure 5.17. Figure 5.17(a) shows a local portion of the model face in the parametric plane and the vertex to be snapped, M_0^0, which is classified on a model edge that is curved in the parametric space. The mesh edges of $P(M_0^0)$ on this model face connect M_1^0 to M_2^0 and M_2^0 to M_4^0. When the new vertex, M_5^0, is introduced on the model edge the loop created connects M_1^0 to M_2^0 to M_4^0 to M_5^0 to M_1^0 (Figure 5.17(b)) which self intersects in the parametric space and can not be triangulated. By expanding the loop to M_1^0 to M_2^0 to M_3^0 to M_4^0 to M_5^0 to M_1^0 (Figure 5.17(c)) the self intersection is eliminated and the loop can be triangulated.



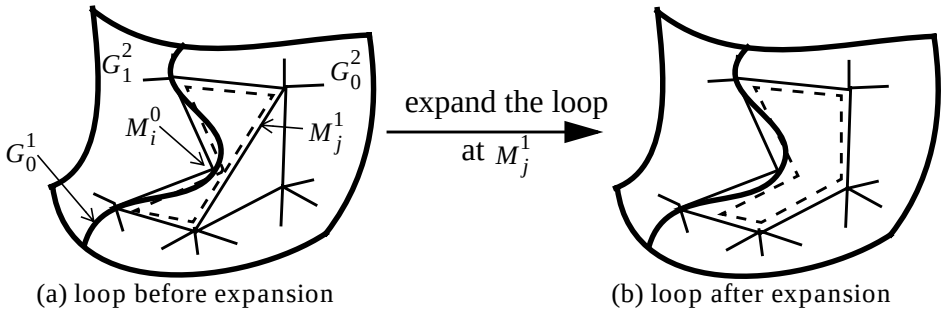
(a) original situation (b) self-intersected loop (c) expanded loop

Figure 5.17 An example to solve self-intersected loop by expansion.

A second reason for loop expansion is to avoid creation of poorly shaped mesh faces. Again this situation is most easily described when the loop is triangulated in the parametric plane. Figure 5.18(a) demonstrated this for a situation where the new mesh vertex, M_i^0, on the model edge is too close to another edge in the loop, M_j^1. As before, it is straight forward to expand the loop as indicated in Figure 5.18(b).

5.5.3 Cavity Creation by Deletion of Interacted Mesh Entities

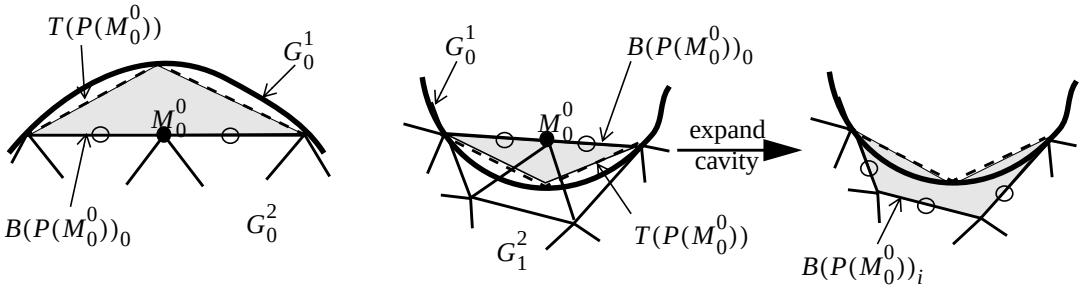
Definition 5.5: The set of mesh faces that are the original triangulation of the piece of model face bounded by $P(M_0^0)$ that contains M_0^0 is termed $B(P(M_0^0))_0$. A subscript is added to indicate that $B(P(M_0^0))_0$ may change during cavity construction process.



(a) loop before expansion (b) loop after expansion

Figure 5.18 Example to expand a loop to improve resulting mesh quality.

Since both $T(P(M_0^0))$ and $B(P(M_0^0))_0$ are bounded by $P(M_0^0)$, and are a triangulation of the same piece of model face, the union of them ($B(P(M_0^0))_0 \cup T(P(M_0^0))$) defines an initial cavity (or cavities). Consider the two 2D examples in Figure 5.19 to understand the initial cavity defined in this manner and further consider if that cavity is empty, thus making it a valid cavity to be triangulated by the cavity mesher. Figure 5.19(a) shows a cavity (the shaded area) that is constructed on top of the existing local mesh. It is empty and its triangulation will conform to the existing local mesh. Therefore, the cavity is valid and can be triangulated. In the example depicted in Figure 5.19(b), the initial cavity (the shaded area) is not empty and can not be triangulated. However, as indicated in rightmost figure, $B(P(M_0^0))_0$ can be expanded to $B(P(M_0^0))_i$ by the deletion of appropriate mesh entities so that the cavity bounded by $B(P(M_0^0))_i \cup T(P(M_0^0))$ is an empty cavity that can be triangulated.



(a) a good initial cavity (b) an initial cavity that needs expanding

Figure 5.19 Two-manifold examples to demonstrate the cavity to be defined.

When the local geometry M_0^0 bounds is non-manifold (e.g. M_0^0 is classified on a model face that bounds two model regions or M_0^0 in classified on a model edge that bounds multiple model regions), $B(P(M_0^0))_i \cup T(P(M_0^0))$ will define multiple cavities, at least one for each model

region M_0^0 bounds. Figure 5.20 shows a 2D example where M_0^0 bounds two model faces and two cavities are created after the appropriate deletion of mesh entities (the two shaded areas with different patterns).

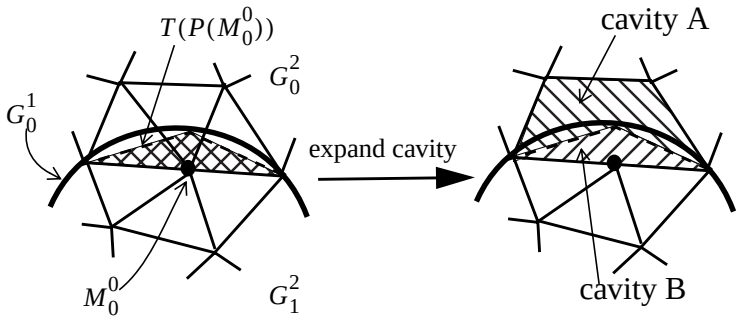


Figure 5.20 Non-manifold example to demonstrate the cavity to be defined.

Given an initial cavity consisting of $P(M_0^0)$, $T(P(M_0^0))$, $B(P(M_0^0))_0$, the subsections that follow discuss the algorithm used to first determine if $B(P(M_0^0))_0 \cup T(P(M_0^0))$ bounds a proper cavity, and then to expand $B(P(M_0^0))_0$ until $B(P(M_0^0))_i \cup T(P(M_0^0))$ creates a valid and acceptable cavity. The expansion process first determines and deletes mesh entities that "intersect" the cavity. It then deletes any disconnected mesh entities in the cavity created during the first step. This process yields a valid cavity. However, further cavity expansion may be desired to produce a cavity that can be triangulated with properly shaped elements.

5.5.3.1 Delete Intersecting Mesh Entities and Expand Cavity

Deletion of intersecting mesh entities is a two step process in which tetrahedra bounded by a mesh edge in $P(M_0^0)$ are deleted first. This is followed by deletion of intersecting tetrahedra not bounded by a mesh edge in $P(M_0^0)$.

Deleting interacting tetrahedra bounded by an edge in $P(M_0^0)$

The pseudo-code to delete interacting tetrahedra bounded by an edge of $P(M_0^0)$ is described in Figure 5.21. The algorithm starts from a M_i^1 on $P(M_0^0)$, and gets the two mesh faces that M_i^1 bounds, one from $T(P(M_0^0))$ (referred to as $M_{T_i}^2$) and another from $B(P(M_0^0))_0$ (referred to as $M_{B_i}^2$). Also get $M_i^1\{M^3\}$, the tetrahedra M_i^1 bounds. Next intersect the closure of each tetrahedron in $M_i^1\{M^3\}$ with the closure of $M_{T_i}^2$ minus the closure of M_i^1 and place those that intersect in the set of tetrahedra to be deleted. Figure 5.22 depicts the three dimension situation with respect to intersecting $M_{T_i}^2$ (the shaded triangle) with a tetrahedron in $M_i^1\{M^3\}$. Since $M_{T_i}^2$ and the tetra-

hedron are bounded by edge M_i^1, they are intersected if and only if M_i^0 is on or between the two half-planes containing $M_{i_1}^2$ and $M_{i_2}^2$, where M_i^0 is the vertex of $M_{T_i}^2$ that does not bound M_i^1, and $M_{i_1}^2$ and $M_{i_2}^2$ are the two mesh faces of the tetrahedron M_i^1 bounds.

```
initialize an empty set
loop over edges of P(M_0^0)
    let M_i^1 to be the current edge
    get M_{T_i}^2, M_{B_i}^2 and M_i^1{M^3}
    if M_{T_i}^2 intersects any tetrahedron in M_i^1{M^3}
        insert the intersecting tetrahedron into the set
        insert tetrahedron in M_i^1{M^3} between M_{T_i}^2 and M_{B_i}^2 into the set
    end if
end loop
delete all tetrahedra in the set and expand B(P(M_0^0))_0
```

Figure 5.21 Pseudo-code to delete interacting tetrahedra.

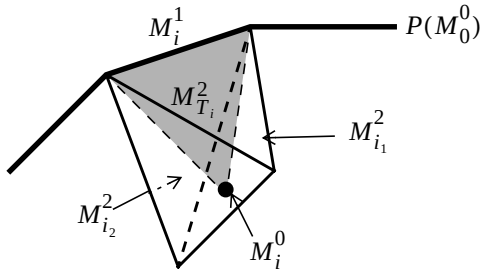


Figure 5.22 3D illustration of intersecting $M_{T_i}^2$ with a tetrahedron in $M_i^1\{M^3\}$.

If there is no intersection between $M_{T_i}^2$ and $M_i^1\{M^3\}$, continue to another edge of $P(M_0^0)$ and repeat. If there is an intersection, add the intersecting tetrahedron and tetrahedra in $M_i^1\{M^3\}$ sharing faces with tetrahedra into the set until reaching $M_{B_i}^2$, then continue to another edge of $P(M_0^0)$ and repeat.

After all edges in $P(M_0^0)$ are processed, delete the tetrahedra placed in the set and update $B(P(M_0^0))_i$. When deleting a tetrahedron, $B(P(M_0^0))_i$ is updated to $B(P(M_0^0))_{i+1}$ by considering the four mesh faces of this tetrahedron. The mesh faces classified in a model region that have no tetrahedron attached are eliminated from $B(P(M_0^0))_i$, otherwise, they are added to $B(P(M_0^0))_i$ (note that faces in $B(P(M_0^0))_0$ have been re-classified in model region since they do not represent the local boundary any more).

Figure 5.23 demonstrates why tetrahedra in $M_i^1\{M^3\}$ between $M_{T_i}^2$ and $M_{B_i}^2$ must also be deleted. Figure 5.23(a) shows the four elements M_i^1 bounds, as well as $M_{T_i}^2$, $M_{B_i}^2$ and a shell defined by the union of $B(P(M_0^0))_i$ and $T(P(M_0^0))$. Figure 5.23(b) shows the results of deleting the intersecting element M_1^3. However, this has not deleted all the elements between $M_{T_i}^2$ and $M_{B_i}^2$ as needed to open that portion of the cavity. Element M_0^3 must be deleted to ensure the space between $M_{T_i}^2$ and $M_{B_i}^2$ is within the cavity (Figure 5.23(c)). In general, there can be multiple elements in this space that have to be deleted.

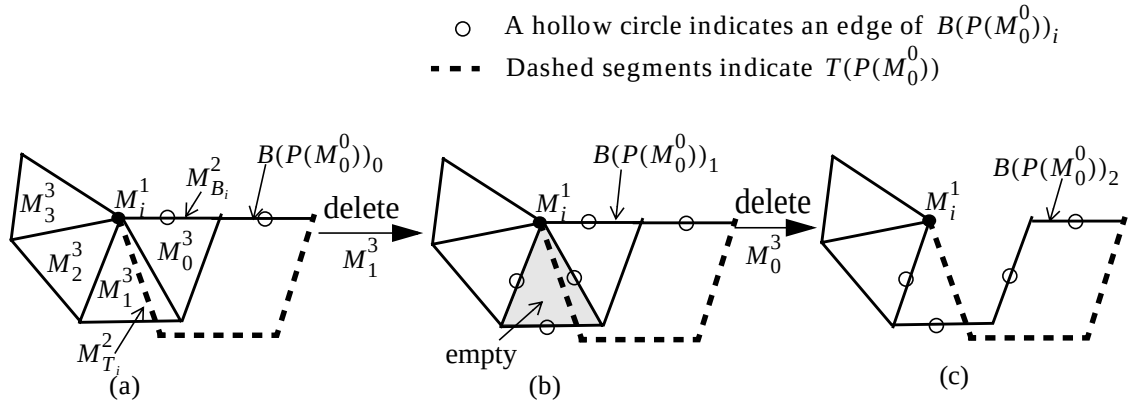


Figure 5.23 Example to show the need to delete M_0^3.

Delete intersecting tetrahedra not bounded by any edge in $P(M_0^0)$

Since the cavity to be constructed must have no self-intersection, the closure of $T(P(M_0^0))$ and $B(P(M_0^0))_n$ intersects only at mesh entities in $P(M_0^0)$. Once all edges in $P(M_0^0)$ are processed, and $B(P(M_0^0))_0$ is the updated to $B(P(M_0^0))_n$, $n \geq 0$, any other intersections between the closure of $T(P(M_0^0))$ and $B(P(M_0^0))_n$ must be determined and dealt with. If intersections are found and the intersecting face from $B(P(M_0^0))_{n+i}$ is classified onto a model region, the intersection can be resolved by deleting this intersecting face, the tetrahedron attached on it, and updating $B(P(M_0^0))_{n+i}$.

In case the intersecting mesh face in some $B(P(M_0^0))_{n+i}$ is classified on a model face (this is rare but possible when the distance between local model entities are small compared to local mesh edge length), the current $T(P(M_0^0))$ and $B(P(M_0^0))_{n+i}$ do not represent a good enough approximation to those portions of the model boundary since the mesh faces classified on those model entities self-intersect. If this arises, see if there are any unsnapped vertices in $B(P(M_0^0))_{n+i}$. If there are, they should be snapped to the boundary to improve the local geometric approxima-

tion. If there are none, or there are still intersections after any unsnapped vertices are snapped, one or both of the local triangulations $T(P(M_0^0))$ and $B(P(M_0^0))_{n+i}$ will need to have their geometric approximation improved through further refinement and snapping.

Figure 5.24 gives the pseudo-code of the algorithm to delete non-bounded tetrahedra. Termination is guaranteed since model entities do not self-intersect and with the expansion of cavity and recursive refinement of all intersecting faces, all self-intersections will be eliminated.

```
do
    get M_B^2 from B(P(M_0^0))_{n+i}, and get M_T^2 from T(P(M_0^0))
    if M_B^2 intersects M_T^2, then
        if M_B^2 is classified on a model region, then
            delete M_B^2 and any tetrahedron attached to M_B^2
            update B(P(M_0^0))_{n+i}
        else if there are unsnapped vertices that bound M_B^2, then
            project unsnapped vertices onto their classified model boundaries
        else
            refine both M_B^2 and M_T^2
            update B(P(M_0^0))_{n+i} and T(P(M_0^0))
        end if
    end if
until the closure of T(P(M_0^0)) and B(P(M_0^0))_{n+i} intersect only at P(M_0^0)
```

Figure 5.24 Pseudo-code to solve self-intersections.

5.5.3.2 Delete Disconnected Mesh Entities and Expand Cavity

After deletion of intersecting entities, $B(P(M_0^0))_0$ has been updated to $B(P(M_0^0))_m$, $m \geq n$. it is possible that the portion of $B(P(M_0^0))_m$ associated with a single cavity has been split into two or more "disconnected" pieces. For the purposes of the current algorithm, two pieces are referred to as "disconnected" if there is no common mesh edge between the two pieces. Figure 5.25 shows a "disconnected" $B(P(M_0^0))_m$ where the only common mesh entity is a mesh vertex.

Figure 5.26 gives a 2D example to demonstrate how mesh entities may become disconnected. Figure 5.26(a) depicts a situation where M_0^0 is to be snapped. In the image $T(P(M_0^0))$ is defined by the bold line and $B(P(M_0^0))_0$ by the edges marked by hollow circles. Figure 5.26(b) shows $B(P(M_0^0))_m$ after M_0^2 and all faces intersecting $T(P(M_0^0))$ are deleted. This process has left the two shaded elements disconnected, and edges in $B(P(M_0^0))_m$ has grown from two edges to

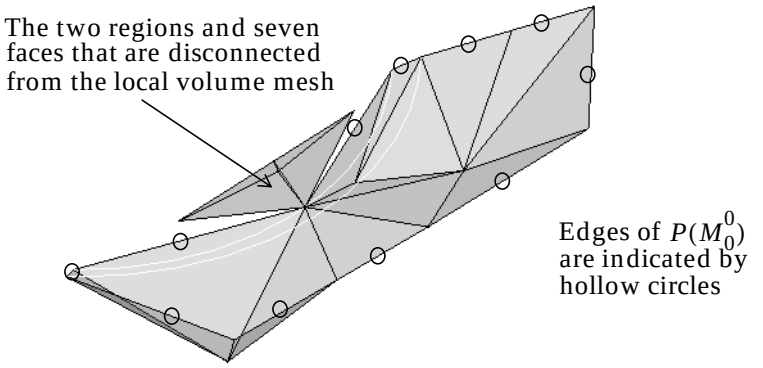


Figure 5.25 Picture of disconnected $B(P(M_0^0))_m$.

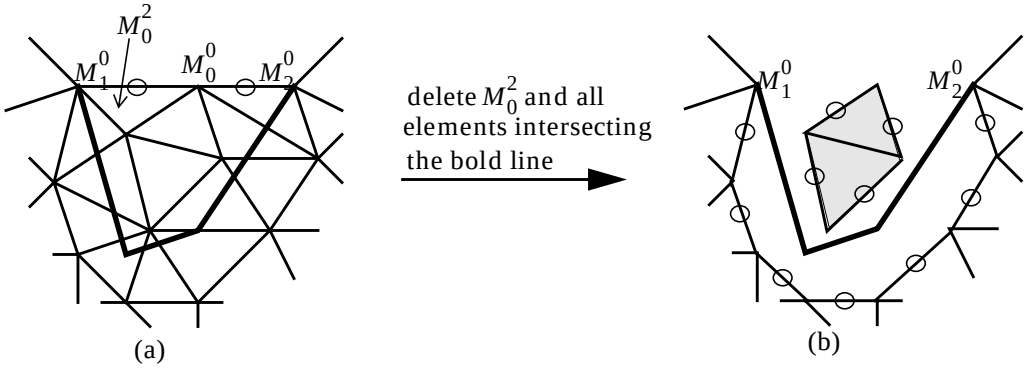


Figure 5.26 2D example to illustrate the creation of disconnected elements.

eleven edges, seven that bound the desired final $B(P(M_0^0))_{m+i}$ and four that bound the disconnected piece of mesh.

Figure 5.27 gives the pseudo-code that checks and eliminates disconnected mesh entities. It begins by marking as connected a mesh face of $B(P(M_0^0))_m$ that is bounded by a mesh edge of $P(M_0^0)$. Other mesh faces of $B(P(M_0^0))_m$ that share an edge with faces marked are also marked as connected. The process terminates when there are no edge neighbors left to mark as connected. Any mesh faces of $B(P(M_0^0))_m$ that are not marked connected after this step are part of a disconnected portion of the mesh that is to be deleted. Any tetrahedron connected to such a face is deleted and any bounding faces of those tetrahedra that remain are appended into $B(P(M_0^0))_m$, but not marked as connected. The process continues until $B(P(M_0^0))_m$ only contains faces that are marked as connected.

```
get a mesh face bounded by an edge of P(M_0^0) from B(P(M_0^0))_m
initialize a face list and insert the mesh face into the list
while the face list is not empty
      pop a face from the list
      if the face is not marked, then
            mark it
            insert its neighboring faces in B(P(M_0^0))_m into the list
      end if
end while
while any face in B(P(M_0^0))_m is not marked
      pop out an un-marked face from B(P(M_0^0))_m
      delete the face, its connected tetrahedra, and update B(P(M_0^0))_m
end while
```

Figure 5.27 Pseudo-code to check and delete disconnected mesh entities.

5.5.3.3 Expand Cavity if Any Mesh Entities of Cavity are Close

The procedures given in Section 5.5.3.1 and Section 5.5.3.2 will update $B(P(M_0^0))_0$ to $B(P(M_0^0))_l$ ($l \geq m$) and will create a valid cavity. However, it is possible that sending it to the cavity meshing procedure will yield some poorly shaped elements. For instance, if the three cavities given in Figure 5.28-5.29 are sent to a cavity mesher for triangulation, the small angle and short distance will remain in the resulting mesh, leading to poorly-shaped elements. Further expansion of the cavity is used to eliminate these problems when possible.

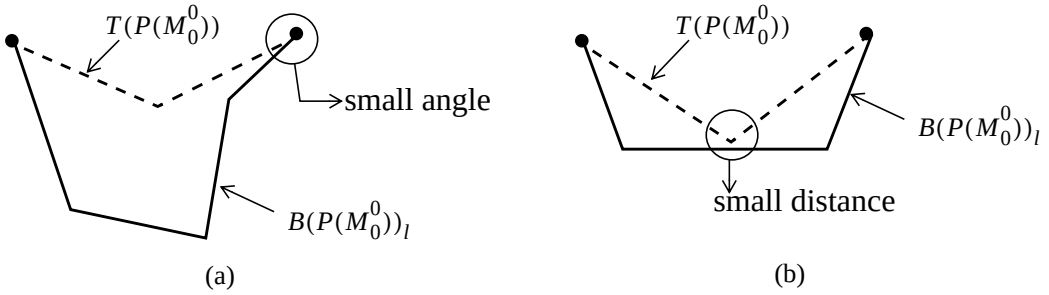


Figure 5.28 2D examples of close cavity mesh entities.

The procedure to further expand the cavity consists of three steps:

- Check and eliminate small dihedral angles

  For each edge in $B(P(M_0^0))_{l+i}$ (including edges on $P(M_0^0)$), get the two faces on the cavity it bounds, and calculate the dihedral angle between the two faces. If the angle is too small, the

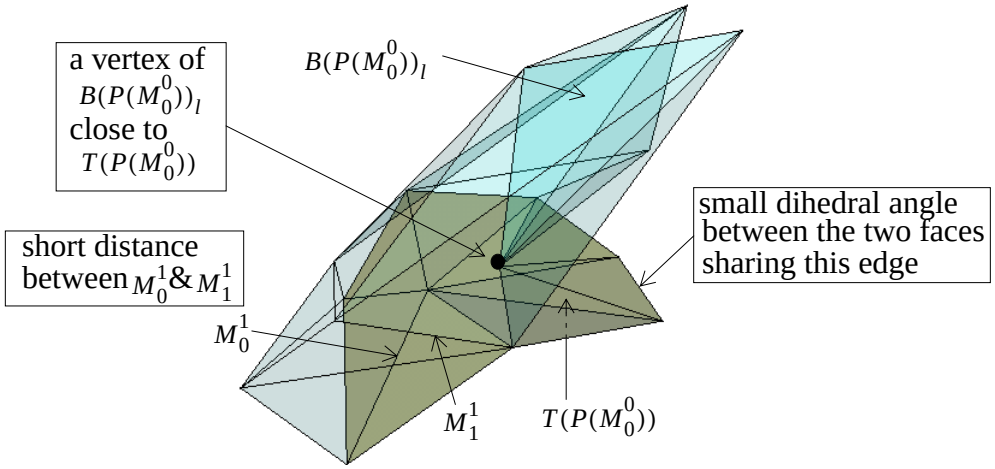


Figure 5.29 A valid but poorly-shaped 3D cavity.

tetrahedron connected to a face in $B(P(M_0^0))_{l+i}$ that is not classified on the model boundary is deleted, and the cavity is expanded.

- Check and eliminate short vertex-face distance

  For each interior vertex of $B(P(M_0^0))_{l+i}$, determine the shortest distance to faces in $T(P(M_0^0))$. If the distance is too small, all tetrahedra connected to the vertex are deleted with the cavity expanded.

- Check and eliminate short edge-edge distance

  For each interior edge of $B(P(M_0^0))_{l+i}$, determine the shortest distance to interior edges in $T(P(M_0^0))$. If the distance is too small, delete all tetrahedra connected to the edge from $B(P(M_0^0))_{l+i}$ and expand the cavity.

5.5.4 Cavity Triangulation and Handling of New Boundary Vertices

Once a cavity is completed it is meshed by a robust cavity triangulation procedure [43]. As with any cavity triangulation procedure, the present procedure is required to not alter the shape of the cavity. However, the robustness of the procedure lies in the fact that it avoids the need to rely on interior vertex insertion only by allowing the mesh edges on the boundary of the cavity to be split. As discussed in reference [43], this process does not create a problem when meshing interior cavities since the existing mesh entities outside the cavity can also be split. The introduction of such vertices on a portion of the cavity boundary belonging to $T(P(M_0^0))$ does require two specific considerations in the present algorithm.

The first consideration arises when a vertex is introduced on a portion of $T(P(M_0^0))$ shared by two cavities. If it is introduced during the triangulation of the first cavity, the second

cavity must be up-dated to reflect its presence before it is triangulated. If it is introduced during triangulation of the second cavity, the appropriate mesh entities in the first cavity must be split.

The second consideration is that the new vertex will need to be snapped to the boundary when it is classified on a curved model face or edge. The way this is handled in the procedure is to simply place any such vertex back into the list of vertices to be snapped and process it using the overall procedure. This approach does, however, have the potential disadvantage of the creation of an infinite loop in the following way: The vertex can not be directly snapped and the mesh modification procedures do not take care of it, in which case it is comes back to the cavity mesher. Although the cavity mesher will eliminate the vertex, it can introduce new vertices that will have to be snapped, thus defining a potential infinite loop. In practice the infinite loop can be avoided by putting a finite limit to the number of iterations of new vertex introduction by cavity meshing. Of course, if that limit were to be hit, the remaining vertices would not be snapped.

A large number of difficult cases have been tested and to date not one example has reached the limit placed in the code. Therefore, all the vertices have been snapped to the boundary. An explanation of why new boundary vertices introduced during cavity meshing have not caused a problem is indicated in Figure 5.30. Figure 5.30(a) shows the local mesh and model boundary (bold line) with respect to snap vertex M_0^0. Figure 5.30(b) shows $T(P(M_0^0))$ (indicated by dashed line), which improves local geometry approximation by adding mesh vertex M_3^0, as well as the cavity (shaded area), and a triangulation of the cavity that introduces vertex M_4^0 on $T(P(M_0^0))$. Since the process of constructing $T(P(M_0^0))$ improves the geometric approximation of the local surface triangulation, vertices created by splitting edges of $T(P(M_0^0))$, in the present case M_4^0, are generally closer to model boundary than the original vertex to be snapped, and are therefore easier to snap to the boundary.

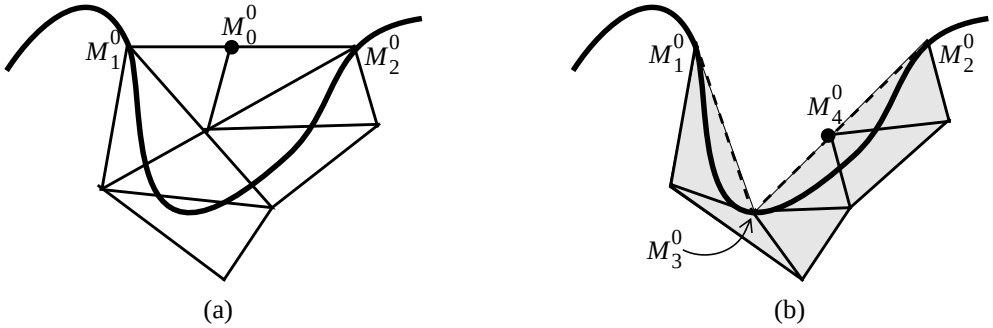


Figure 5.30 An example to show snapping M_4^0 is easier than snapping M_0^0.

5.6 Examples

Four examples are given in this section to demonstrate the application of the vertex snapping procedure. In each case an initial coarse mesh goes through a set of refinement and coarsening steps to match a specified mesh size field. In a refinement and coarsening step all mesh entities are traversed a single time to see if they match the given mesh size field. If mesh edges are too long they are marked for refinement and if too short they are marked for coarsening. Refinement is then performed using refinement templates and coarsening by edge collapse. The vertex snapping procedure is applied after each complete refinement and coarsening step.

5.6.1 Torus with Holes

Figure 5.31 shows the parasolid model of a torus with four cylindrical holes, a coarse initial mesh, and the mesh after the seven steps of mesh adaptation as needed to match the specified mesh size field. Referring to a coordinate system with origin at center of the torus, x-axis along the center-line of the small hole, the isotropic mesh size field to control the adaptation is defined as:

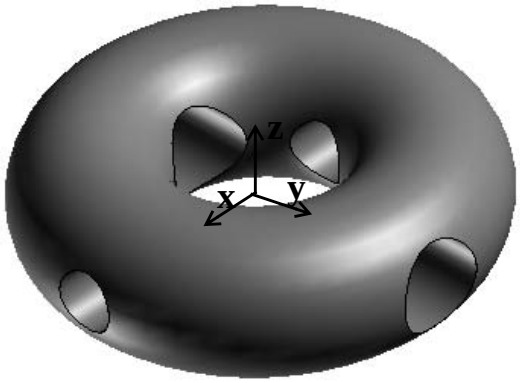
$$h(x, y, z) = \frac{r_0}{4}\left(5 - 4e^{2(r_0 - \sqrt{y^2 + z^2})}\right) \tag{5.1}$$

where r_0 is the radius of the small hole. The close-up picture shows the final mesh around a small hole. Figure 5.32 shows the intermediate meshes after each adaptation step. As shown, for each step, all boundary mesh vertices are snapped by the vertex snapping procedure.

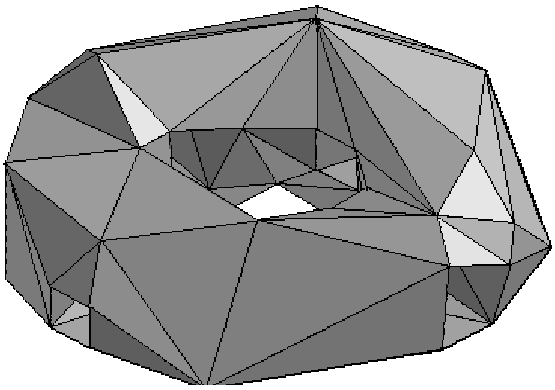
Table 5.1 lists the number of mesh vertices snapped by repositioning, local mesh modification, or local cavity triangulation in each adaptation step. It can be seen that, although most vertices are snapped simply by a repositioning, a substantial number of vertices in each step are snapped by performing local mesh modification operations. In addition, in most steps a few mesh vertices need to be addressed by the local cavity triangulation procedure. As indicated by the last column of Table 5.1, four new boundary vertices were created during the application of the twelve cavity meshing operations. When processed by the vertex snapping procedure, three of the new vertices were simply snapped by re-positioning and one was snapped after an edge collapse operation.

Table 5.2 indicates the local mesh modification operations performed during the vertex snapping process. Clearly, collapsing to a mesh vertex on the first problem plane is the most effective modification operation. Other edge collapse operations are the second most frequent modification followed by edge swap, region collapse and split plus collapse.

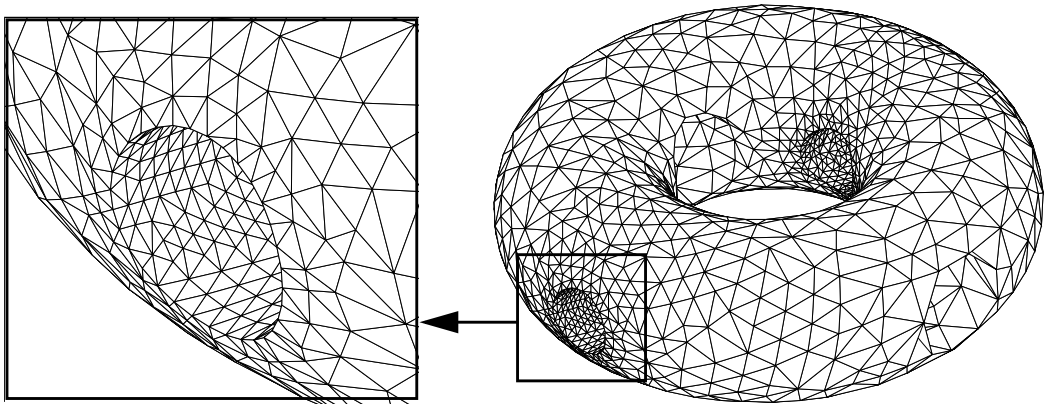
Timing statistics collected for this example indicate that the vertex snapping procedure including local mesh modifications as the first stage is 2.5 times faster than the one employing only cavity retriangulation.



(a) geometry model.



(b) initial mesh.



(c) final mesh.

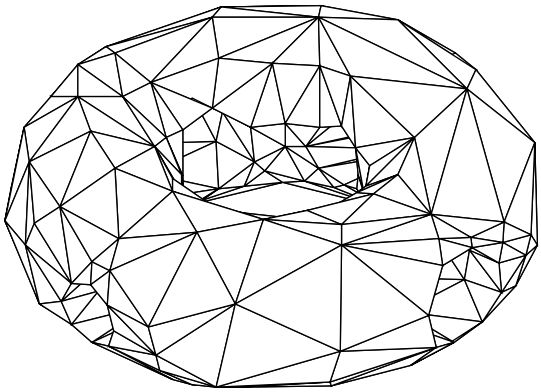
Figure 5.31 Torus with holes.

5.6.2 Sleeve of Ball Bearing

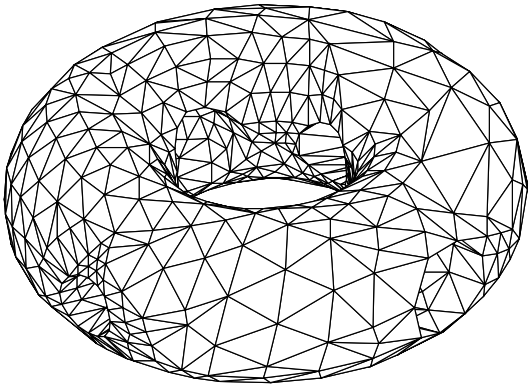
Figure 5.33 shows a mechanical part, an initial coarse mesh, and the mesh after the five steps of mesh adaptation as needed to match the specified mesh size field. Referring to the coordinate system as shown, the isotropic mesh size field to control adaptation is defined as:

$$h(x, y, z) = 0.002\left(4 - 3\cos\left(\frac{\pi}{L_0} \times z\right)\right) \qquad -L_0 \le z \le L_0 \tag{5.2}$$

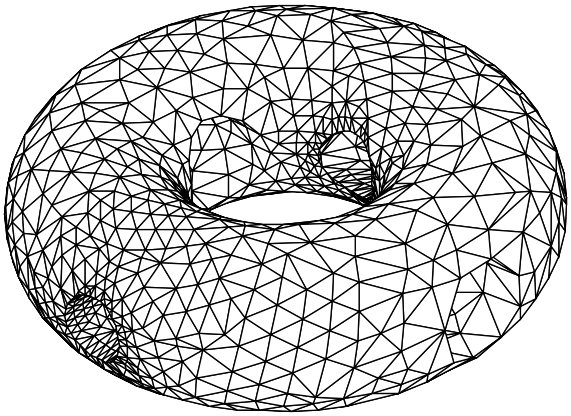
where L_0 is the half length of the part in z-axis. Figure 5.34 shows the intermediate meshes after the first three adaptation steps. From the picture of the close-up view, it is clear that the small



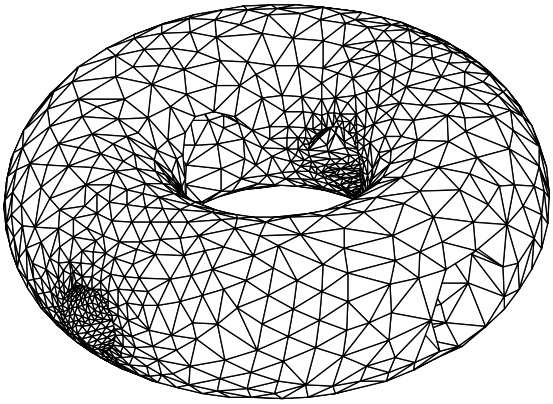
(a) after 1st iteration.



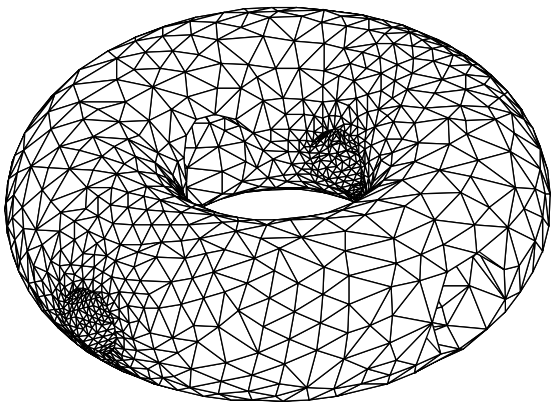
(b) after 2nd iteration.



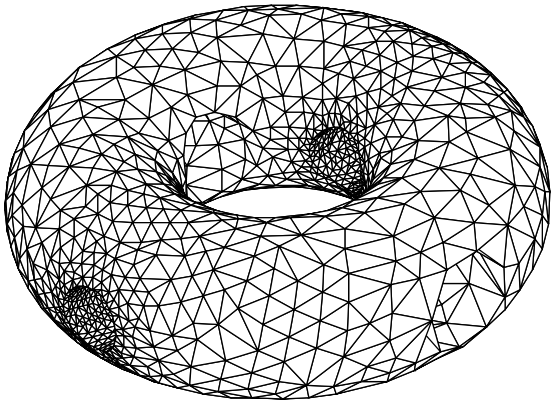
(c) after 3rd iteration.



(d) after 4th iteration.



(e) after 5th iteration.



(f) after 6th iteration.

Figure 5.32 Intermediate meshes of torus with holes.

Table 5.1 Vertices snapped by different modifications for torus model.

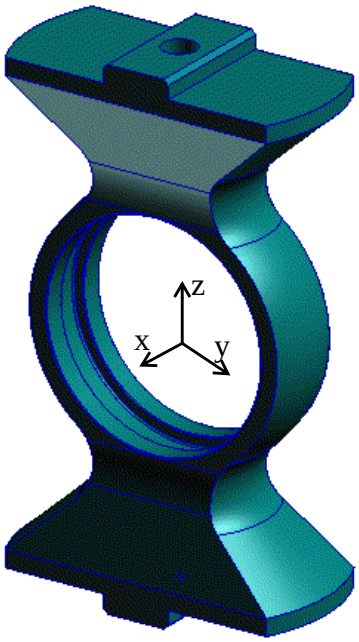
Iteration	Nbr of vertices to be snapped	Nbr of vertices snapped by re-positioning	Nbr of vertices snapped by local modifications	Nbr of vertices snapped by cavity triangulation	Nbr of new boundary vertices created in cavity triangulation
1	204	184	20	0	0
2	638	595	41	2	2
3	1187	1137	46	4	1
4	1449	1381	66	2	0
5	609	551	56	2	0
6	142	107	34	1	1
7	39	26	12	1	0

Table 5.2 Statistics of performed mesh modification operations for torus model.

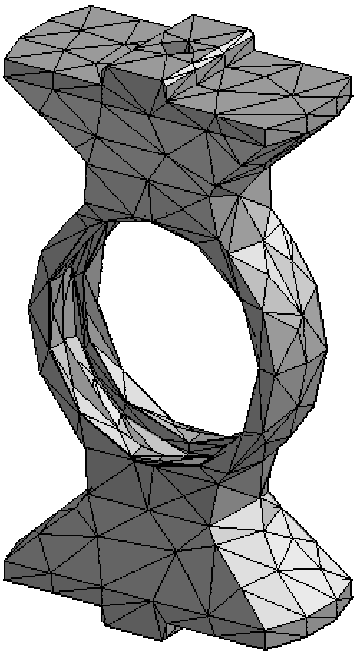
Iteration	Nbr of operations that collapse to a vertex on first problem face	Nbr of other mesh modification operations			
		edge collapse	edge swap	region collapse	split(s) + collapse
1	12	5	1	0	1
2	34	19	3	1	2
3	52	21	10	4	1
4	67	35	12	1	2
5	61	37	11	2	1
6	36	12	9	1	1
7	16	2	1	0	0

curved geometry features, such as small fillets, are properly represented by the snapping procedure as the mesh gets refined.

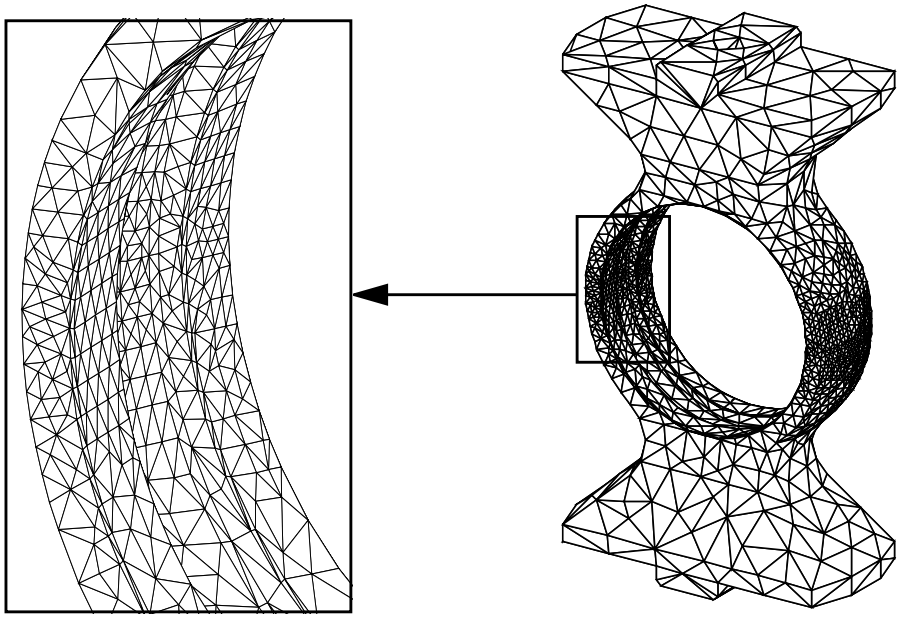
Table 5.3 lists the number of mesh vertices snapped by repositioning, local mesh modification operation or local cavity triangulation in each adaptation step, and Table 5.4 indicates the local mesh modification operations performed during the snapping of the mesh vertices. It can be seen that, in this example, more vertices need to be snapped by local mesh modification or local cavity remeshing, and all are snapped successfully. In this example three new boundary vertices are created when triangulating cavities. One of them is snapped by a re-positioning, one is snapped after performing a split and collapse operation, and the last one was dealt with by a cavity remeshing that did not introduce any new boundary vertices.



(a) geometry model.

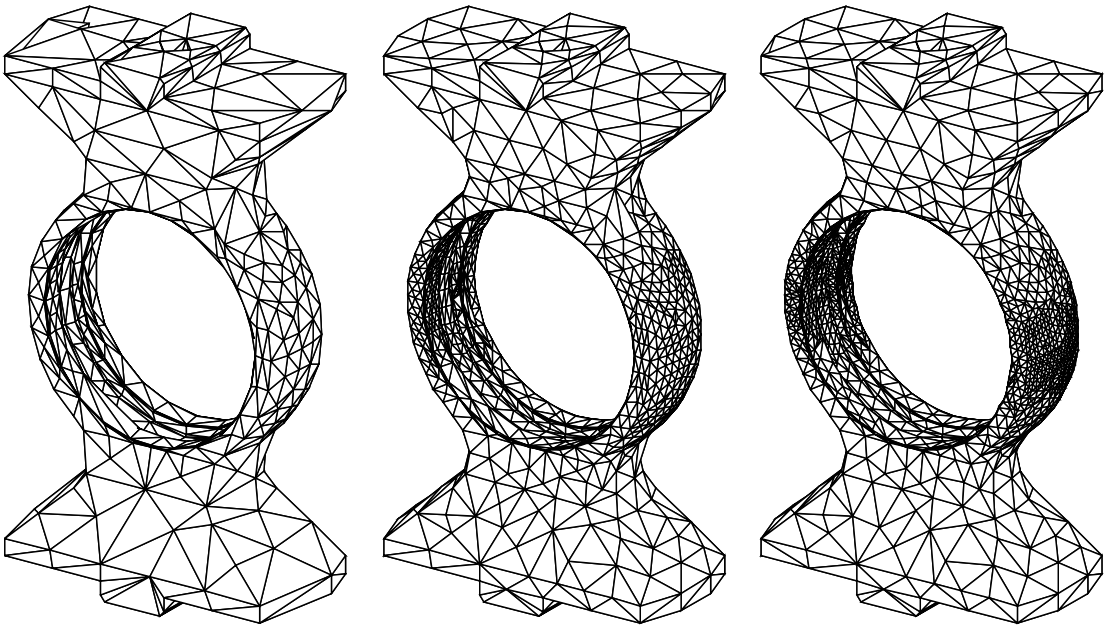


(b) initial mesh.



(c) final mesh (after 5th iteration).

Figure 5.33 Bearing Sleeve.



(a) after 1st iteration. (b) after 2nd iteration. (c) after 3rd iteration.

Figure 5.34 Intermediate meshes of bearing sleeve.

Table 5.3 Vertices snapped by different modifications for sleeve model.

Iteration	Nbr of vertices to be snapped	Nbr of vertices snapped by re-positioning	Nbr of vertices snapped by local modifications	Nbr of vertices snapped by cavity triangulation	Nbr of new boundary vertices created in cavity triangulation
1	561	391	158	12	1
2	888	741	129	18	0
3	993	876	105	12	2
4	211	158	52	1	0
5	40	31	9	0	0

Table 5.4 Statistics of performed mesh modifications for sleeve model.

Iteration	Nbr of operations that collapse to a vertex on first problem face	Nbr of other mesh modification operations			
		edge collapse	edge swap	region collapse	split(s) + collapse
1	120	45	20	4	3
2	90	39	17	4	3
3	77	43	15	1	0

Table 5.4 Statistics of performed mesh modifications for sleeve model.

Iteration	Nbr of operations that collapse to a vertex on first problem face	Nbr of other mesh modification operations			
		edge collapse	edge swap	region collapse	split(s) + collapse
4	44	17	7	0	1
5	9	2	2	0	0

5.6.3 Hub Section

Figure 5.35 shows another mechanical part, an initial mesh, and the mesh after the four iterations of mesh adaptation as needed to match the specified mesh size field. It can be noted that, in this example, only three parts of the initial mesh are refined and the initial mesh is so coarse that there are mesh edges spanning across cylindrical holes. As shown by the refined mesh and its close-up views, although geometric approximation remains the same where no refinement is requested, the geometry features are well represented where refinement is requested.

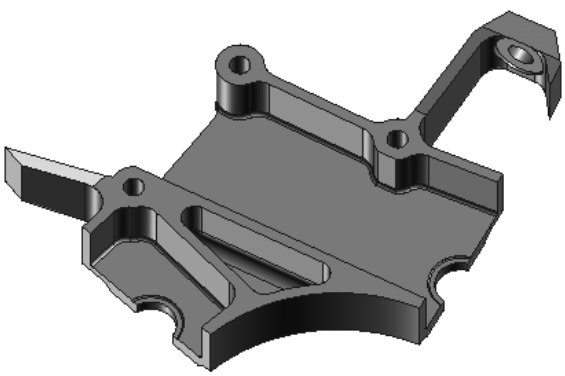
As before, all mesh vertices are snapped in this example. Table 5.5 lists the number of mesh vertices snapped by repositioning, local mesh modification operations or local cavity triangulation in each adaptation iteration, and Table 5.6 indicates what local mesh modifications are applied to snap all these vertices.

Table 5.5 Vertices snapped by different modifications for hub section model.

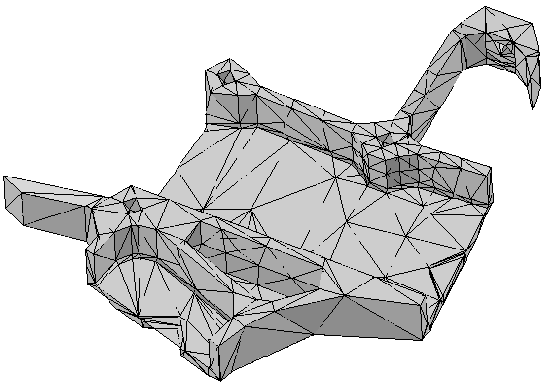
Iteration	Nbr of vertices to be snapped	Nbr of vertices snapped by repositioning	Nbr of vertices snapped by local remeshing	Nbr of vertices snapped by cavity triangulation	Nbr of new boundary vertices created in cavity triangulation
1	183	142	40	1	0
2	388	337	46	5	1
3	546	512	31	3	0
4	102	94	8	0	0

Table 5.6 Statistics of performed mesh modifications for hub section model

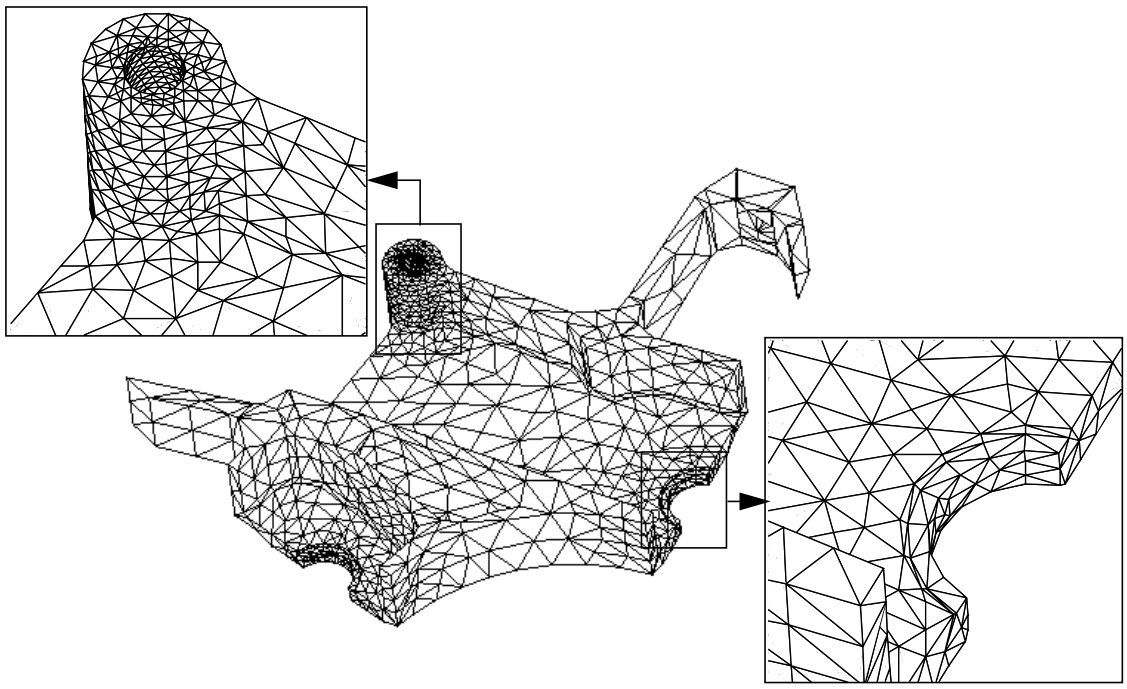
Iteration	Nbr of operations that collapse to a vertex on first problem face	Nbr of other mesh modification operations			
		edge collapse	edge swap	region collapse	split(s) + collapse
1	34	8	4	4	0
2	37	24	7	2	2
3	26	9	6	4	0
4	10	2	0	0	0



(a) geometry model



(b) initial mesh



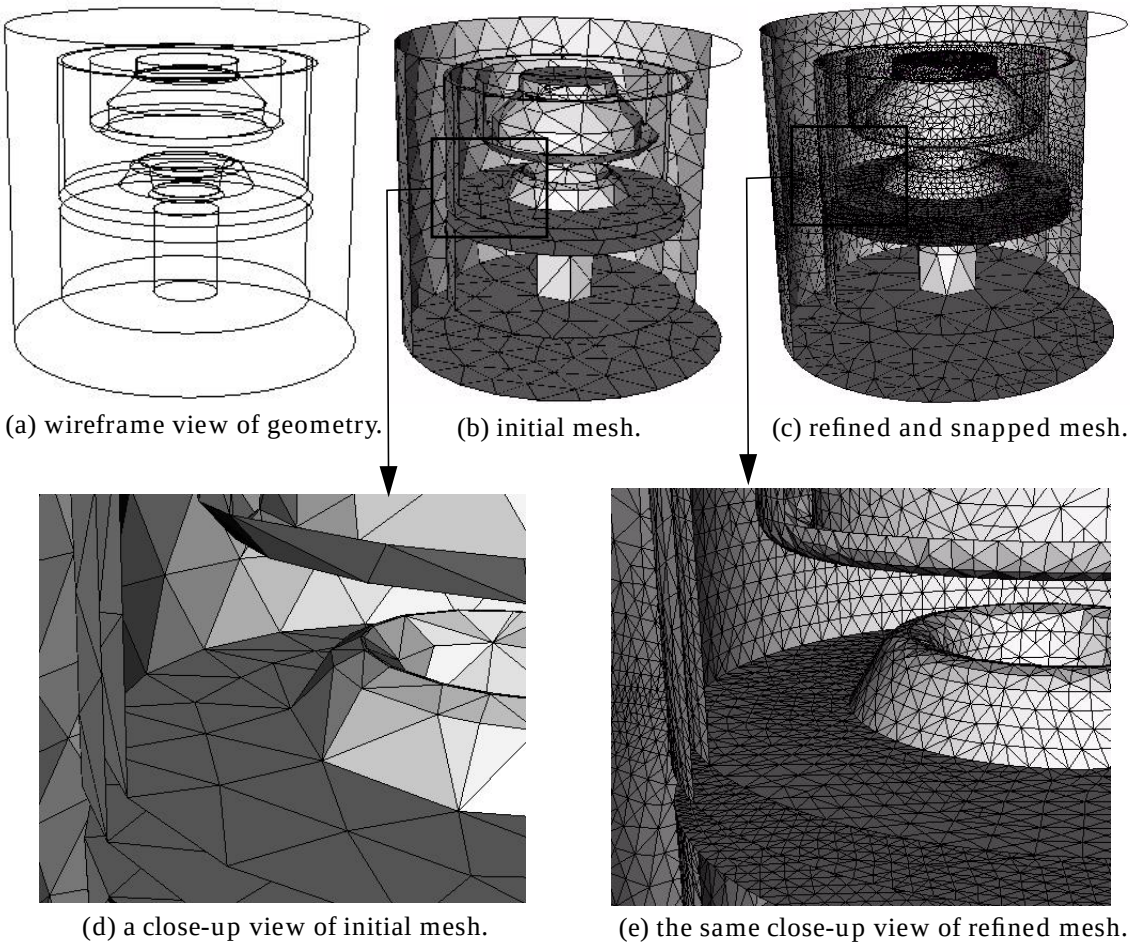
(c) adapted mesh (Close-up views shows two local meshes where refinement is requested)

Figure 5.35 Hub Section.

5.6.4 Gun

Figure 5.36 shows a non-manifold ACIS geometry, an initial coarse mesh and the mesh after sixteen steps of mesh adaptation. Figure 5.36(a) is a wireframe view of the non-manifold ACIS geometry, consisting of 5 solids, 51 faces, 98 edges and 59 vertices. Figure 5.36(b) shows the surface mesh of the initial mesh, where all volume elements and a part of the surface elements in front are hidden so that a view of the interior is exposed. Figure 5.36(c) shows the refined and

refined mesh. Figure 5.36(d) and (e) give a close-up view of a small portion of the domain. The desired mesh size field is adaptively specified in a heat transfer analysis.



(a) wireframe view of geometry. (b) initial mesh. (c) refined and snapped mesh.

(d) a close-up view of initial mesh. (e) the same close-up view of refined mesh.

Figure 5.36 Gun Model.

As before, all mesh vertices have been snapped. Table 5.7 lists the number of mesh vertices snapped by repositioning, local mesh modification operations or local cavity triangulation in each adaptation iteration, and Table 5.8 indicates what local mesh modifications are applied to snap all these vertices.

Table 5.7 Vertices snapped by different modifications for the gun model.

Iteration	Nbr of vertices to be snapped	Nbr of vertices snapped by re-positioning	Nbr of vertices snapped by local remeshing	Nbr of vertices snapped by cavity triangulation	Nbr of new boundary vertices created in cavity triangulation
1	8	1	6	1	1
2	9	0	15	0	0
3	4	1	3	0	0
4	82	66	16	0	0
5	78	64	14	0	0
6	308	277	29	2	0
7	155	148	7	0	0
8	493	470	21	2	0
9	392	378	14	0	0
10	1037	1014	23	0	0
11	1576	1541	33	2	0
12	1136	1114	22	0	0
13	2627	2594	32	1	0
14	2522	2499	23	0	0
15	2882	2854	28	0	0
16	1888	1875	13	0	0

Table 5.8 Statistics of performed mesh modifications for the gun model.

Iteration	Nbr of operations that collapse to a vertex on first problem face	Nbr of other mesh modification operations					
		edge collapse	edge swap	edge split	face swap	split + collapse	double split + collapse
1	3	2	0	0	0	0	0
2	2	2	10	1	0	0	0
3	0	0	2	2	0	1	0
4	2	4	13	0	0	0	0
5	2	2	14	1	0	0	0
6	8	17	16	2	0	0	0
7	1	2	3	0	0	2	0
8	3	16	10	0	0	1	1
9	1	12	5	0	0	1	0
10	1	15	6	0	0	0	0
11	6	16	14	0	1	1	0
12	2	9	12	0	0	1	0
13	0	19	12	0	1	0	0
14	2	15	0	0	0	0	0
15	2	23	10	0	0	0	0
16	0	10	5	0	0	0	0

CHAPTER 6

Mesh Adaptation Procedure

Chapter 2 presented a mesh metric field that can define desired element size and shape distribution. We also saw in Chapter 5 that curved domain can always be properly approximated by placing new boundary vertices onto their classified boundaries. Given any initial tetrahedral mesh and the mesh metric field defined over the domain, the goal of this chapter is to present a general *h*-version mesh adaptation procedure that applies local mesh modification operations to yield a mesh of the same quality as would be obtained by an anisotropic domain remeshing procedure but at less computational cost. The key to the effectiveness of the procedure is the ability to use information about the current local mesh and the mesh metric field to quickly determined the appropriate mesh modification to apply without the cost of examining all possible operations described in Chapter 4.

The adaptation procedure is an effective integration of four interacted processes: mesh refinement, mesh coarsening, vertex snapping and element shap correction. All are governed by the same mesh size field. The concept of transformed space (see Section 2.2.1) plays a critical role in the procedure such that all geometric computations that control the application of mesh modifications are performed in the transformed space. Therefore, in general the procedure generates an anisotropic mesh when the mesh size field is anisotropic.

To make any given mesh satisfy a given mesh metric field, the philosophy underlying the application of local mesh modification is to approach the desired local element size and shape configuration incrementally. In particular, it is:

- identify those mesh entities not satisfying the mesh metric field;
- perform appropriate mesh modification so that local mesh will better satisfy the metric field;
- repeat above steps until the mesh metric field is satisfied to an acceptable degree.

This chapter is structured as follows: the overall algorithm is presented in Section 6.1. Consideration is then given to the specific components of mesh coarsening (Section 6.2), mesh refinement (Section 6.3) and element shape correction (Section 6.4). The vertex snapping process used is that given in Chapter 5. In Section 6.5, three examples are given to show the effectiveness and to demonstrate particular technical aspects of the procedure.

6.1 Overall Procedure

Figure 6.1 gives the pseudo-code of the overall mesh adaptation procedure. Given any initial mesh, a mesh metric field as well as interval [L_{lower}, L_{upper}] of allowed mesh edge length in transformed space, and the domain the initial mesh is classified onto, the mesh adaptation procedure repeatedly modifies the mesh to satisfy the mesh metric field, in the meantime, maintaining appropriate approximation to the curved geometry domain. In particular, it consists of three stages: mesh coarsening and mesh refinement, and shape correction. The first two stages, coarsening and refinement, are controlled by mesh edge length analysis in transformed space. Generally speaking, It splits all mesh edges longer than the upper bound of the interval, places any new boundary vertices onto their classified boundaries, and, if possible, eliminates any mesh edge shorter than the lower bound. The third stage is dominated by both element quality and mesh edge length analysis in the transformed space. It re-aligns local mesh configurations through the determination and elimination of sliver tetrahedra using local mesh modifications.

```
1   initialize a mesh edge list;
    initialize L'max to be zero
    loop over mesh edges in the mesh
       compute L', the length of current mesh edge in transformed space
       if L' < Llower
          append the current into the mesh edge list
       else if L' > L'max
          set L'max to be L'
       end if
    end loop
11  process the edge list using mesh coarsening algorithm (refer to Section 6.2.1)

13  while L'max > Lupper, the upper bound of allowed edge length interval
       compute L'ref = max(α · L'max, Lupper), where α is a constant in (0.5,1]
       tag all edges longer than L'ref as refinement edges
       split the list of tagged edges and their adjacent elements by subdivision
       place all new boundary vertices onto their classified boundaries
       process all short edges element subdivision creates using coarsening algorithm
       update L'max
20  end while

22  determine all sliver tetrahedra in the transformed space in the mesh
23  eliminate these sliver tetrahedra by local mesh modifications if possible
```

Figure 6.1 Overall mesh adaptation procedure.

In the first stage (line1-11 in Figure 6.1), mesh coarsening is applied to eliminate short edges existing in the mesh. Here, a mesh edge is referred to as “short” if its length is smaller than L_{lower} in the transformed space. The reason for performing coarsening ahead of refinement is less peak memory usage. Let n_i be the number of elements in the initial mesh, n_r is the number of elements mesh refinement introduces, and n_c is the number of elements mesh coarsening removes. The peak element number is n_i - n_c + n_r if mesh coarsening is first applied, otherwise, it would be $n_i + n_r$.

The second stage (line 13-20 in Figure 6.1) properly splits mesh edges longer than L_{upper} in transformed space using the full set of edge-based refinement templates, places new boundary vertex onto their classified boundaries as well as coarsens any new short mesh edges the refinement templates create. It is applied multiple iterations to reduce the length of all mesh edges below L_{upper} in the transformed space associated with the given mesh metric field.

To achieve intelligent refinement patterns, a refinement criteria, refining a set of longest mesh edges in the transformed space in each iteration, is adopted. As a result, the maximal transformed length of the mesh is reduced in an incremental manner. The set of longest mesh edges are efficiently determined by maintaining variable L'_{max} , and tagging edges longer than

$$max(\alpha \cdot L'_{max}, L_{upper}) \tag{6.1}$$

as refinement edges, where α is a given constant that controls the length reduction rate. We suggest that $\alpha \leq 1$ and at least greater than $\sqrt{2}/2$ so that refinement edges can be selected in a way that the type of sliver tetrahedron depicted in Figure 2.9 can be eliminated during subdivision and coarsening iterations. Figure 6.2 illustrates that the “a set of longest edges criteria” can always select the desired refinement edges to split a triangle. In this example, three typical triangles are depicted: a triangle with an angle close 180 degrees but without short edge (left), a triangle with a relatively shorter edge (middle) and a triangle close to equilateral triangle (right). Solid black bullets indicate refinement edges when the criteria of (EQ-6.1) with α = 0.75 is applied. It can be seen that, in the case of left triangle, only edge M_a^1 can be the refinement edge; While, for the middle triangle, edge M_c^1 can not be the refinement edge; And all edges are refinement edges for the right triangle. Similarly, the desired refinement edges can be selected in case of three dimensional tetrahedral subdivision templates.

Since the edge-based refinement templates may create short mesh edges when splitting faces or regions, a process to eliminate these is applied after refinement to achieve a refined mesh in which new created vertices are not connected to any short mesh edge. Figure 6.3 gives three typical situations where short mesh edges may be created. Indicated by solid black bullets, when

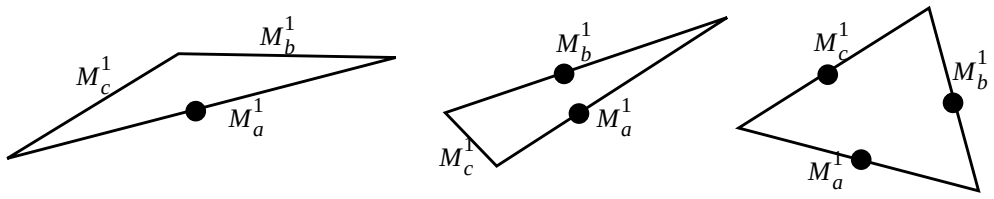


Figure 6.2 Possible patterns to select refinement edges.

subdividing the two triangles or the tetrahedron, the four mesh edges interior to the triangle/tetrahedron have to be created. Any can be a short edge if the simplex is in poor quality in the transformed space. Figure 6.4 shows the importance of eliminating these short edges after the element subdivision process. Figure 6.4(a) gives the initial mesh covering two dimensional domain $[0, 1] \times [0, 1]$ with origin at its left-down corner. The analytical expression specifies the target anisotropic mesh size (transformation matrix) field to catch discontinuity at *x=0.5*. Figure 6.4(b) gives the mesh generated using template refinement process without mesh coarsening. Clearly, the refined mesh is poor and not acceptable. Similar refined meshes have been reported by Lo in case of three dimensional tetrahedral meshes [56]. Lo also indicated that the poorly refined mesh can not be improved using edge/face swap and vertex relocation procedures. Figure 6.4(c) gives the adapted mesh using the algorithm in which subdivision process collects short mesh edges it creates, and the coarsening process that follows eliminates them. It can be seen that coarsening algorithm is effective in solving the problem.

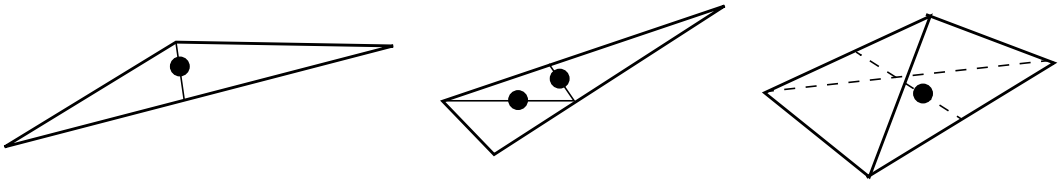


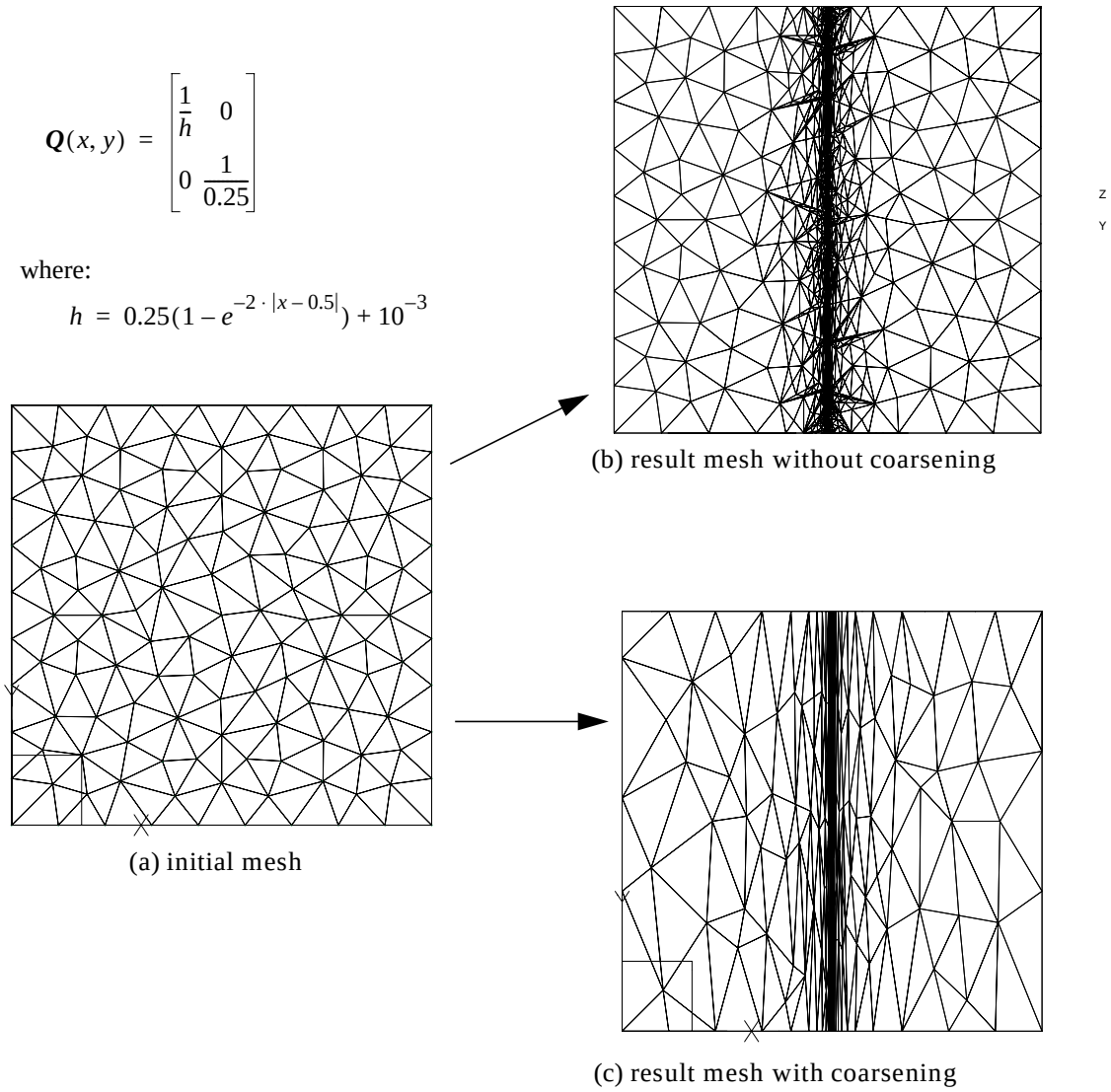
Figure 6.3 Possible short edges to be created in subdivision templates.

To account for curved geometry domains, the edge-based template refinement algorithm maintains a list of new vertices to be placed onto curved model boundary. Boundary vertices are identified in terms of the classification information between the mesh and the geometry model. When splitting a mesh edge classified onto a model face or a model edge, the new vertex inherits the classification of the parent mesh edge, but is temporally placed at a point on the parent mesh

$$\boldsymbol{Q}(x, y) = \begin{bmatrix} \frac{1}{h} & 0 \\ 0 & \frac{1}{0.25} \end{bmatrix}$$

where:

$$h = 0.25(1 - e^{-2 \cdot |x - 0.5|}) + 10^{-3}$$



(b) result mesh without coarsening

(a) initial mesh

(c) result mesh with coarsening

Figure 6.4 2D example to show the importance of eliminating short edges.

edge and inserted into the vertex list. After finishing the current iteration of refinement, the vertex list is processed by the snapping procedure presented in Chapter 5, which places those vertices onto their boundaries.

The short/long mesh edges or sliver tetrahedra snapping may created are addressed by the coarsening, refinement or element shape correction process that follows. Figure 6.5 depicts two 2D examples to demonstrate this. In Figure 6.5(a), placing vertex M_0^0 onto curved boundary G_0^1 generates three long edges (M_0^1, M_1^1 and M_2^1). These three edges are refined with new vertices

placed onto G_0^1 in the next iteration. While in Figure 6.5(b), projecting vertex M_1^0 onto G_1^1 generates a short edge M_3^1, which is collapsed in the following coarsening process.

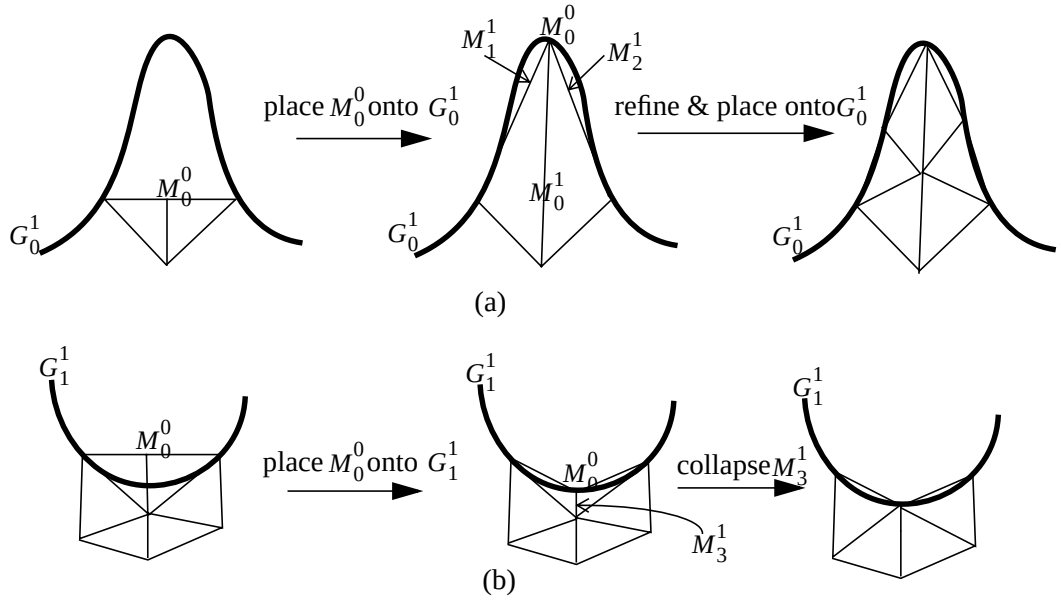


Figure 6.5 Short/long edges projecting vertices generates are addressed.

The second stage terminates when L'_{max} is less than L_{upper}. This termination condition is ensured since: (i) splitting is always performable; (ii) any collapse operation in coarsening and snapping process is denied if it creates a mesh edge longer than the current L'_{max} value in transformed space.

The third shape correction stage (the last two lines in Figure 6.1) aims at eliminating remaining sliver tetrahedra in the transformed space. It visits each tetrahedron of the mesh in turn to identify sliver tetrahedra in the transformed space, then effectively processes identified sliver tetrahedra using the shape correction algorithm described in Section 6.4. Note that the length criteria still controls the third stage in a sense that any local mesh modification is immediately denied if it would create a mesh edge longer than L_{upper}. Local mesh modifications to be evaluated could be: edge / face swap, double split collapse compound operator, split collapse compound operation, edge split and vertex repositioning.

6.2 Coarsening

To meet the mesh metric field, mesh coarsening must be general and not limited to simply reversing of refinement steps.

This section presents a mesh coarsening algorithm using local mesh modifications and governed by mesh metric field. Section 6.2.1 presents the pseudo-code of the algorithm. Focus is then given to explanations of technical aspects, including “always collapsing shortest edge” to avoid extensive evaluations (Section 6.2.2), “collapsing every other vertex” to maintain a good vertex distribution (Section 6.2.3), and the determination of appropriate local mesh modifications (Section 6.2.4).

6.2.1 The Coarsening Algorithm

The mesh coarsening process accepts a list of short mesh edges and a value of maximal mesh edge length allowed in transformed space, L'_{max}, as inputs. The goal is to eliminate all these short mesh edges if possible while not creating any mesh edge longer than L'_{max}.

Figure 6.6 describes the pseudo-code of the mesh coarsening algorithm. The first nine lines of the algorithm simply initialize a dynamic list by visiting each edge in the given short mesh edge list one by one, and inserting any end vertex of the mesh edge being visited into the dynamic list if it does not existed in the list yet. The tag process is used to effectively determine (in the complexity of *O(1)*) if a vertex is already in the dynamic list.

The coarsening process, starting at line 11 in Figure 6.6, repeatedly processes vertices in the dynamic list until the list is empty or all remaining vertices in the list have been tagged as processed. The relative order of vertices processed is important to maintain a good vertex distribution. Section 6.2.3 discusses this issue in detail. Since vertices in the dynamic list may not bound a short edge any more due to the mesh modifications carried out in previous steps, Also note that, in many cases, the vertex in the dynamic list may not bound a short edge any more due to mesh modifications carried out by previous steps, line 14-16 are needed to remove such vertices from the dynamic list.

Line 18-30 process a specific vertex, denoted as M_i^0, that bounds a short edge. Three local mesh modifications can be used: edge collapse, swap(s) collapse compound operations and vertex repositioning. The algorithm first evaluates an edge collapse operation, collapsing the shortest mesh edge in transformed space M_i^0 bounds. Other operations are selectively evaluated in terms of the results of the edge collapse evaluation. Once a local mesh modification is determined, execute it, update the dynamic list as well as tag all vertices connected to M_i^0 through a mesh edge as

```
1   Initialize a dynamic vertex list
    loop over the given short edge list
       for each vertex that bounds the current edge
          if the vertex is not yet tagged to be in the dynamic list
             append the vertex into the dynamic list
             tag the vertex to be in the dynamic list
          end if
       end for
9   end loop

11  while there are unprocessed mesh vertices in the dynamic vertex list
       get an unprocessed vertex M_i^0 from the list in a controlled order (see Section 6.2.3)
       get M_j^1, the shortest mesh edge in transformed space connected to M_i^0
14     if the transformed length of M_j^1 is greater than L_lower
          remove M_i^0 from the dynamic list
16        continue to the next unprocessed vertex in the list
       else
          evaluate collapsing M_j^1 with M_i^0 to be removed
          if the edge collapse would create an edge longer than L'_max in transformed space
             evaluate relocating M_i^0
          else if the edge collapse would lead to flat/inverted elements
             evaluate compound operations
          end if
          if any local mesh modification is determined in evaluation
             tag neighboring vertices of M_i^0 in the dynamic list as unprocessed
             apply the local mesh modification
             remove M_i^0 from the dynamic list if it is collapsed
          else
             tag M_i^0 as processed
30        end if
       end if
    endwhile
```

Figure 6.6 Pseudo-code of mesh coarsening algorithm.

unprocessed. If no local mesh modification can be applied (this can be the best choice in some cases), simply proceed to the next.

Since the success of eliminating a short mesh edge can not be ensured, a process that tags vertices in the dynamic list as processed has been incorporated to guarantee the termination of the algorithm.

6.2.2 Collapsing Shortest Edge

The collapse operation has been used as the major local mesh modification in the coarsening process. To be efficient, it is critical to realize that, given a mesh vertex connected short edge, only one edge collapse operation needs to be checked, i.e. collapsing the shortest edge connected

to the vertex. Here, the shortest edge of a vertex is referred to as the edge of smallest length in transformed space among all edges connected to the vertex.

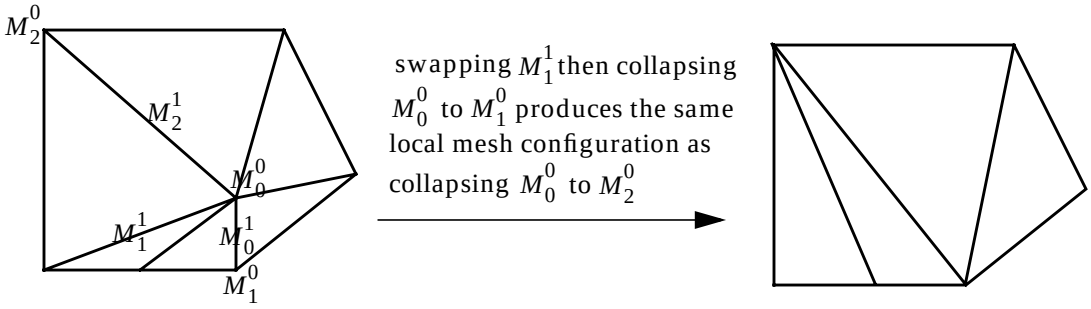


Figure 6.7 2D example to justify "collapsing shortest edge" criteria.

The reason for applying this "collapsing shortest edge" criteria can be explained from two aspects. First, since collapsing a shorter edge causes less changes to the local mesh, collapsing the shortest edge has a higher success rate than collapsing a longer one. The second justification can be illustrated using the simple 2D example depicted in Figure 6.7. In this example, collapsing the shortest edge M_0^1 with vertex M_0^0 removed is not possible while collapsing a longer one (for example M_2^1) can succeed. However, it can be seen that the compound operation, swapping M_1^1 then collapsing vertex M_0^0 to M_1^0, succeeds and yields the same local mesh configuration as directly collapsing vertex M_0^0 to M_2^0.

6.2.3 Eliminating Vertices Topologically Every Other One

By processing vertices in the dynamic list in an order that is topologically every other one, a good vertex distribution during mesh coarsening can be maintained. Consider Figure 6.8 for an example to demonstrate this idea. Figure 6.8(a) shows a two dimensional mesh to be coarsened, where M_0^0 represents the current vertex being processing and black bullets indicate its topologically neighboring (edge connected) vertices. Figure 6.8(b) shows the locally coarsened mesh with M_0^0 collapsed. In terms of the "every other vertex" criteria, the next vertex to be processed should be a vertex indicated by a circle in Figure 6.8(b). Figure 6.8(c) shows such a result mesh (vertex M_2^0 is collapsed). It can be seen that both vertex distribution and local mesh quality are maintained. Figure 6.8(d) shows the result mesh where vertex M_1^0 were allowed to be collapsed next. In this case, coarsening would be too concentrated on the left-bottom corner, while other parts of the mesh are untouched. Furthermore, even additional swap operation(s) is applied to regain a good local mesh connectivity, vertices are not in good distribution yet.

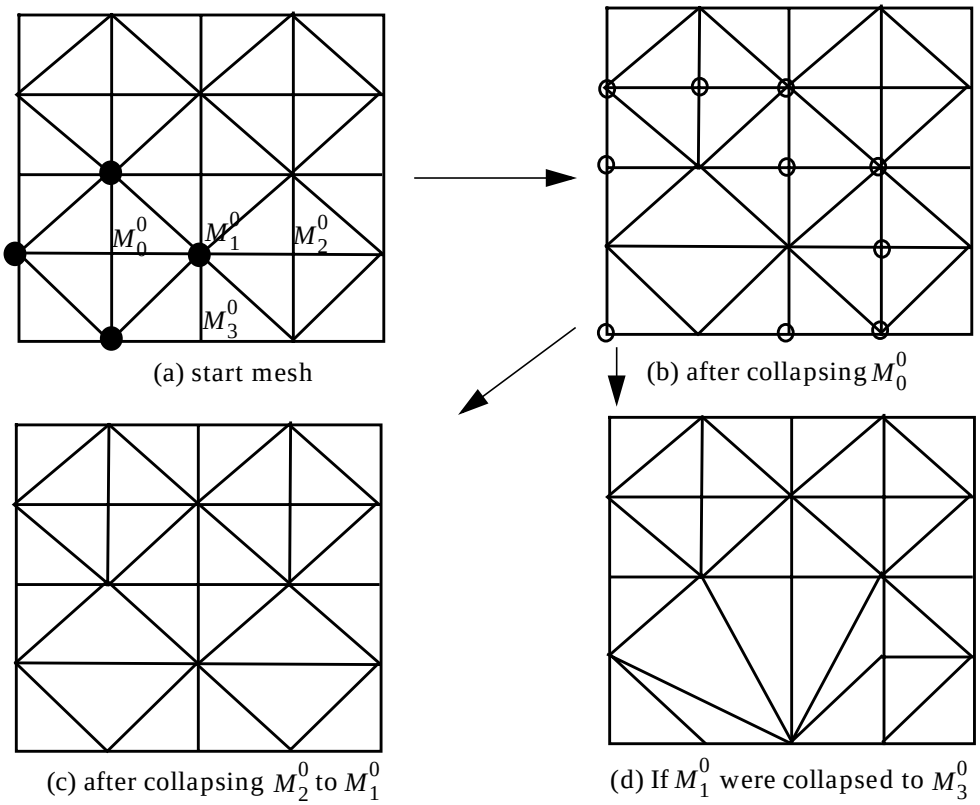


Figure 6.8 2D example to show the need of processing vertex “every other one”.

De Cougny has pointed out the needs of maintaining vertex distribution control during mesh coarsening and proposed a method in terms of an independent set paradigm [23]. Two deficiencies exist in his method. First, although coarsening is a local activity in adaptive analysis, the independent set of vertices is determined in a global sense, and it has to be determined more than once in case where where more than one level of coarsening is required. Second, constraining vertices in the independent set as not collapsible is unnecessary and may preclude desired edge collapse operations. An algorithm that overcomes these two deficiencies focuses on changing the order of processing vertices in the dynamic list to follow the “topologically every other vertex” rule.

Figure 6.9 and Figure 6.10 gives the pseudo-code that changes the order of processing vertices in the dynamic list by properly tagging them as "not collapsible" and cleaning up these tags. Figure 6.11 depicts a part of a 2D mesh to be coarsened to demonstrate this algorithm, where integers (1,2,3,.....,13) indicates mesh vertices in the dynamic (an abbreviation of mesh vertex M_i^0),

and the bottom line indicates the order of these vertices. The first vertex to be processed is M_1^0 since it is the first vertex in the dynamic list and no vertex has been tagged as "not collapsible" yet. Figure 6.11(b) shows the resulting mesh and the updated dynamic list after M_1^0 is collapsed, where black bullets indicate the vertices tagged as "not collapsible" (connected to M_1^0 by mesh edges). The next two vertices, M_4^0 and M_6^0, in the dynamic list can not be processed since M_4^0 has been tagged as "not collapsible" and M_6^0 is not connected to any tagged vertex by a mesh edge. M_{11}^0 is processed next. Figure 6.11(c) shows the resulting mesh after M_{11}^0 is collapsed and vertex M_7^0, M_8^0, M_9^0 and M_{13}^0 are further tagged. Following the same logic, in the first traversing over the dynamic list, the next two vertices to be processed in turn are M_{12}^0 and M_5^0. Then, the "not collapsible" tags on remaining vertices of the dynamic list are cleared and the list is traversed from the start again until all vertices in the dynamic list are processed.

```
while there are vertices not tagged "processed" in the dynamic vertex list
   loop over vertices not tagged "processed" in the dynamic list
      if no vertex in dynamic list is tagged as "not collapsible" yet
         process M_i^0 and update tags (see )
      else if M_i^0 is not tagged but connected to a vertex tagged as "not collapsible"
         process M_i^0 and update tags (see )
      end if
   end loop
   clear "not collapsible" tags on vertices in the dynamic list
end while
```

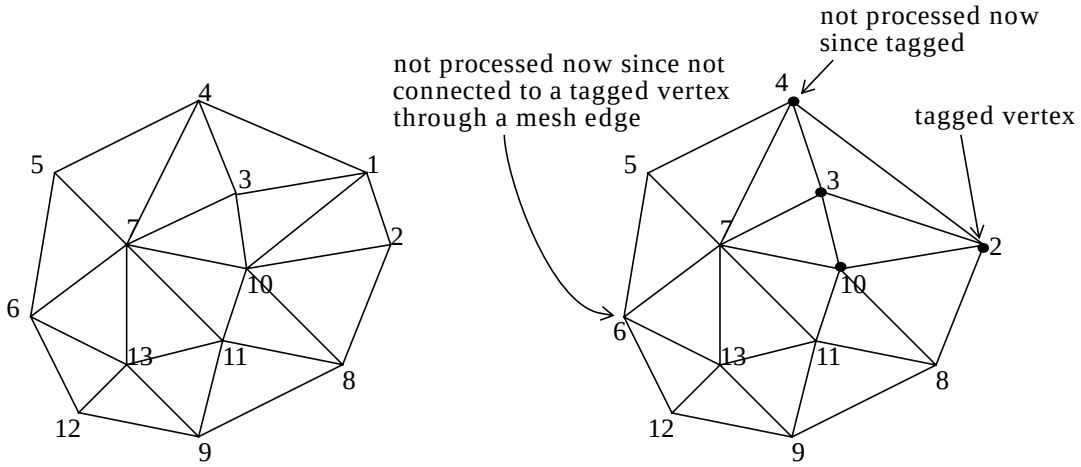
Figure 6.9 Pseudo-code to process vertices "topologically every other one".

```
if M_i^0 can be removed by edge collapsing or compound operation
   tag neighboring vertices of M_i^0 as "unprocessed" and "not collapsible"
   apply the edge collapsing or compound operation
   remove M_i^0 from the dynamic list
else
   tag M_i^0 as "processed"
end if
```

Figure 6.10 Pseudo-code to process vertex M_i^0 and update tags.

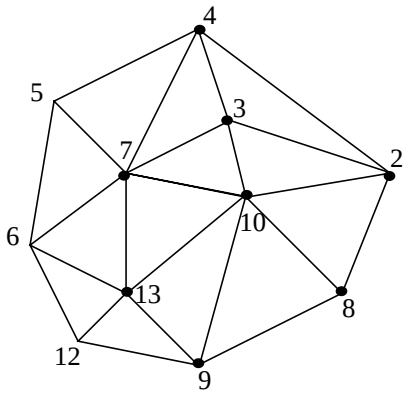
6.2.4 Coarsening Using Local Mesh Modifications

Local mesh modifications used includes: edge collapsing, vertex repositioning, swap(s) plus collapse compound operation.



dynamic list: {**1**,4,6,11,3,10,7,12,5,2,13,8,9}
(a) a portion of a 2D mesh to be coarsened

dynamic list: {**4**,6,11,3,10,7,12,5,2,13,8,9}
(b) after vertex 1 collapsed



dynamic list: {4,6,**3**,10,7,12,5,2,13,8,9}
(c) after vertex 11 collapsed

Figure 6.11 2D example for the algorithm of maintaining "every other vertex".

Collapsing the shortest edge is first evaluated. Without actually creating the local result mesh, the evaluation process can qualify edge collapsing by virtually computing the information below: (i) topological validity; (ii) geometric validity (refer to Section 4.2.1), and (iii) the maximal length of all new edges to be created in the transformed space (refer to Section 4.2.2). Edge collapse operation is denied if the computed maximal length is greater than L'_{max} (the given allowed length in transformed space), or it violates any topology / geometry restriction.

The evaluation of other two local mesh modifications depends on the evaluation result of edge collapsing. In case edge collapsing is denied because the computed maximal edge length would be greater than L'_{max}, repositioning an end vertex of the shortest edge is further evaluated

since the fact that collapsing the shortest mesh edges leads to a long mesh edge implies that the local mesh is in good density but bad vertex distribution. The target location of repositioning is computed using the method described in Section 4.1.4.

In case edge collapsing is denied because it would lead to flat/inverted elements, swap(s) plus collapse compound operations are further evaluated. The motivation of the compound operation is to collapse the shortest short edge, however, pre-swaps are required to make the collapsing successful. To identify pre-swap operations, first those tetrahedra that would be flat or inverted after the application of edge collapsing are determined, possible swaps are then determined using the same tetrahedron analysis techniques described in Section 5.4.2.

Once a local mesh modification is determined, execute it, update the dynamic list as well as the associated tags. However, it is also possible that no local mesh modification can be applied (note that in some cases this can be the best choice), then simply proceed to the next.

6.3 Refinement

A wide variety of algorithms have been developed for adaptive mesh refinement. In general, they can be decomposed into three main groups, depending on which techniques they are based:

- Element subdivision methods [23,3,7,60,54,76,88,9,11,12,41,92],
- insertion using Delaunay criteria [17,31,61], and
- Edge splitting [25,56,65].

This section presents a refinement engine using the full set of element subdivision templates, similar to De Cougny's method [23], however, governed by a given mesh metric field. Methods in terms of Delaunay vertex insertion or edge splitting are not considered since they are relatively expensive compared with element subdivision methods, and the full set of element subdivision templates are used to minimize over-refinement. Given a mesh size field and a refinement threshold L'_{max} (the maximal allowed edge length in transformed space), the refinement engine can determine a set of refinement edges (mesh edges longer than L'_{max} in transformed space) and effectively split these edges and their adjacent elements without traversing over the whole mesh and generate anisotropic meshes.

The structure of the following subsections is: the algorithm of the refinement engine is first given (Section 6.3.1), then discussions are focused on technical aspects, including where to

introduce new vertices (Section 6.3.2), diagonal mesh edge selection in case of ambiguity (Section 6.3.3) and the use of lookup table for efficiency (Section 6.3.4).

6.3.1 The Refinement Algorithm

Figure 6.12 presents the refinement algorithm that uses a full set of subdivision templates. It performs *create* and *delete* operations to modify the mesh. The create operations are applied from bottom-up, i.e., first create mesh entities to replace refinement edges, then create mesh entities to replace faces refinement edges bound, finally create mesh entities to replace regions refinement edges bound. The reason for using such a bottom-up strategy is to make this algorithm appliable to non-manifold geometry models for instance the example as depicted in Figure 6.13, which has a model edge that does not bound any model face, and a model face that does not bound any model region. In the process of subdividing, the algorithm creates child mesh entities for each parent mesh entity (the mesh entity being split) one by one and does not immediately delete the parent mesh entity but properly attaches creation information onto the parent mesh entity if it still bounds any higher dimension mesh entities. The attached information is retrieved and used when the higher dimension mesh entities the parent entity bounds are processed. For example, in case of subdividing face M_j^2 that bounds two mesh regions, the algorithm first retrieves all creation information attached to edges in $\left\{M_j^2\{M^1\}\right\}$, then creates a triangulation of M_j^2 using retrieved information to replace M_j^2, finally attaches the information of the triangulation onto M_j^2. When any mesh region M_j^2 bounds is processed, the attached information on M_j^2 will be retrieved and used. After the creation process, a deletion process deletes any subdivided mesh entity and its downward mesh entities that do not bound any higher dimension mesh entities. The final mesh is always conforming after all subdivided mesh entities are deleted.

Figure 6.14 illustrates this subdivision algorithm using a simple two dimensional example. In this example, the initial mesh consists of only four triangles as depicted in Figure 6.14(a), where refinement edges are indicated using four black bullets. Figure 6.14(b) shows the non-conforming mesh after processing the refinement edges, where eight new mesh edges (indicated by the eight segments) are created to replace the four refinement edges and attached onto the four refinement edges, and the circle indicates the collected mesh faces to be processed later. Figure 6.14(c) gives the non-conforming mesh after the shaded face is replaced by three child faces (the three shrunk triangles) and deleted. Also note that the deletion process has deleted the top edge that bounds the shaded face since it does not bound any mesh face any more. Figure 6.14(d) gives the refined mesh after replacing another face to be split with four child triangles and deleting its related mesh entities.

initialize a list of faces to be split
Loop over mesh edges in the mesh
  let M_i^1 be the current edge
  compute L', the length of M_i^1 in the transformed space
  if M_i^1 is a refinement edge, i.e., $L' > L'_{max}$, then
    create new mesh entities to replace M_i^1 (M_i^1 is not deleted)
    if M_i^1 bounds any mesh faces, so can not be immediately deleted, then
      attach a pointer to all new mesh entities to replace M_i^1 onto M_i^1
      for each mesh face M_i^1 bounds
        insert it into the face list and tag it as inserted if it is not tagged as inserted yet
      end for
    else
      delete M_i^1
    end if
  end if
end loop

initialize a list of regions to be split
Loop over the mesh face list to be split
  let M_j^2 be the current mesh face to be split
  determine the subdivision template of M_j^2 based on refinement edges that bound M_j^2
  retrieve all new mesh entities attached to edges in $\left\{ M_j^2\{M^1\} \right\}$
  create a new triangulation to replace M_j^2 using retrieved information
  if M_j^2 bounds any mesh region, therefore can not be immediately deleted
    attach a pointer to the new triangulation onto M_j^2
    for each region M_j^2 bounds
      insert it into the region list and tag it as inserted if it is not tagged as inserted yet
    end for
  else
    delete M_j^2
    delete any edge in $\left\{ M_j^2\{M^1\} \right\}$ and free attached pointer if not bound any face
  end if
end loop

Loop over the mesh region list to be split
  let M_k^3 be the current mesh region to be split
  determine subdivision template of M_k^3 in terms of refinement edges that bound M_k^3
  retrieve all new triangulations attached to the faces $\left\{ M_k^3\{M^2\} \right\}$
  create a new tetrahedralization to replace M_k^3 using retrieved information
  delete M_k^3
  delete any face in $\left\{ \begin{matrix} M_k^3\{M^2\} \\ M_k^3\{M^1\} \end{matrix} \right\}$ and free attached pointer if not bound any mesh region
  delete any edge in $\left\{ M_k^3\{M^1\} \right\}$ and free attached pointer if not bound any mesh face
end loop

Figure 6.12 Pseudo-code for element subdivision.

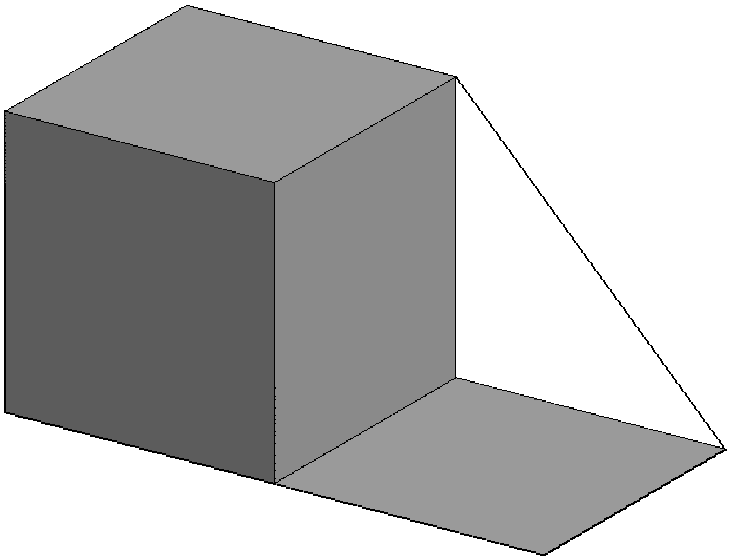


Figure 6.13 Non-manifold model.

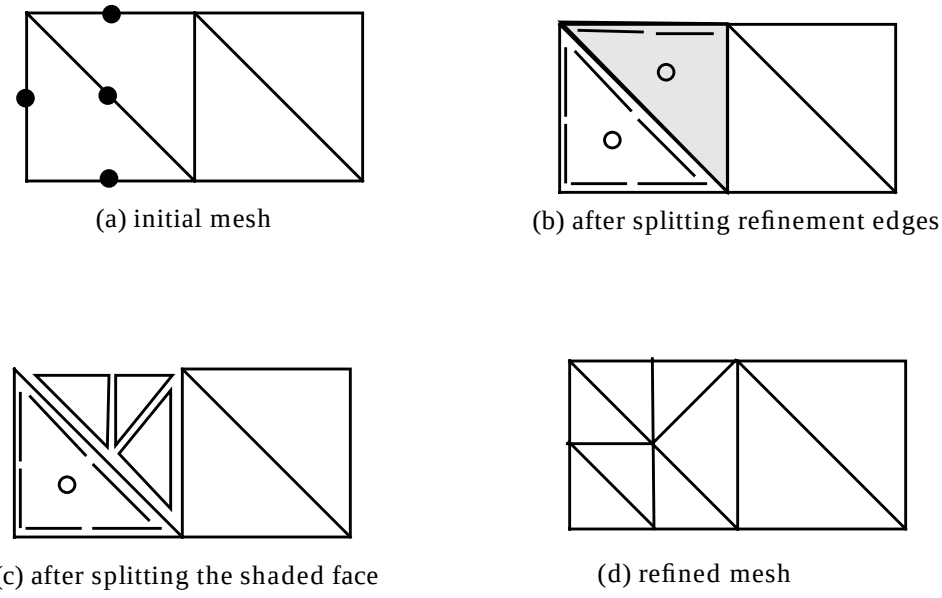


Figure 6.14 Illustration of element subdivision algorithm.

6.3.2 Selection of Vertex Insertion Point

An important issue in mesh refinement is where to insert new vertices. The most commonly used point is the middle point of an edge in physical space when used for isotropic refinement. Considering the transformation interpretation of mesh metric field and the desire for an anisotropic mesh, a better option is the middle point in the transformed space.

Figure 6.15 illustrates the concept of “middle point in transformed space”, where an edge is depicted in both physical and transformed space and ellipses indicate the desired directional mesh size distribution at the two end vertices of the edge. In general, the physical position of middle point in transformed space can be computed using (EQ-3.6). It can be seen that, when mapped back to physical space, the middle point of transformed space usually is not the middle point in physical space, but moves towards the end vertex of smaller desired mesh size. The bigger the desired sizes at two end vertices differ, the more it moves closer to the end vertex of smaller size.

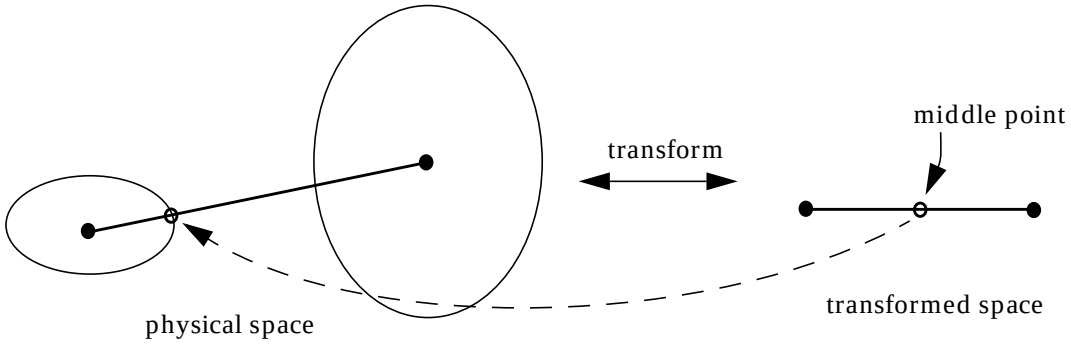


Figure 6.15 Middle point of transformed space.

To compare the two strategies, both have been implemented and several three dimensional examples are investigated. From the practical point of view, it is also observed that the middle point of transformed space is a better insertion point in a sense that: (i) in general, it takes less (or equal) time to reduce the edge length to the level specified by the mesh size field; (ii) it generates less than or equal to the number of elements as geometric midpoint splitting. Table 6.1 and Figure 6.16 provide one of the investigated examples, where Figure 6.16 shows an initial tetrahedral mesh of 3748 elements and an anisotropically refined mesh to capture a cylindrical discontinuity and Table 6.1 gives collected statistics using the two vertex insertion strategies. It can be seen that the middle point of transformed space strategy produced less number of elements and took less time.

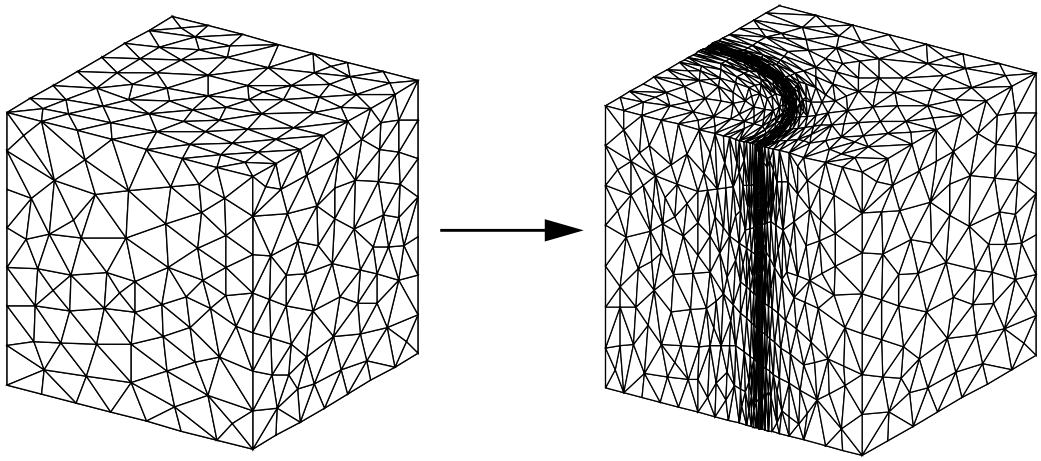


Figure 6.16 Example used to compare the two vertex insertion strategies.

Table 6.1 Comparison between the two vertex insertion strategies.

vertex insertion strategy	statistics of refined mesh		
	# of vertices	# of elements	time (seconds)
middle point of physical space	6760	73598	62.88
middle point of transformed space	6325	68521	40.37

6.3.3 Selection of Diagonals

Another issue in template-based refinement is the appropriate selection of diagonal mesh edges. Figure 6.17 depicts the three ambiguous situations met in applying subdivision templates and indicates their corresponding diagonal edges. In Figure 6.17(a), two edges of a triangle are marked as refinement edges, the triangulation of the quadrilateral area (shaded) is not uniquely determined, two options of creating the diagonal edge are possible. In Figure 6.17(b), all edges of a tetrahedron are marked as refinement edges, and three options of creating the interior edge exist. And two options to create the interior edge in case of Figure 6.17(c).

The rule of creating the shortest diagonal can be used in case of isotropic refinement. Figure 6.18 demonstrates this idea in a simple 3D construct. In this example, the initial mesh is a cube consisting of six tetrahedra (Figure 6.18(a) is a wireframe view of the mesh). Figure 6.18(b) shows a refined mesh where the creation of diagonal were not controlled (three tetrahedra at top-

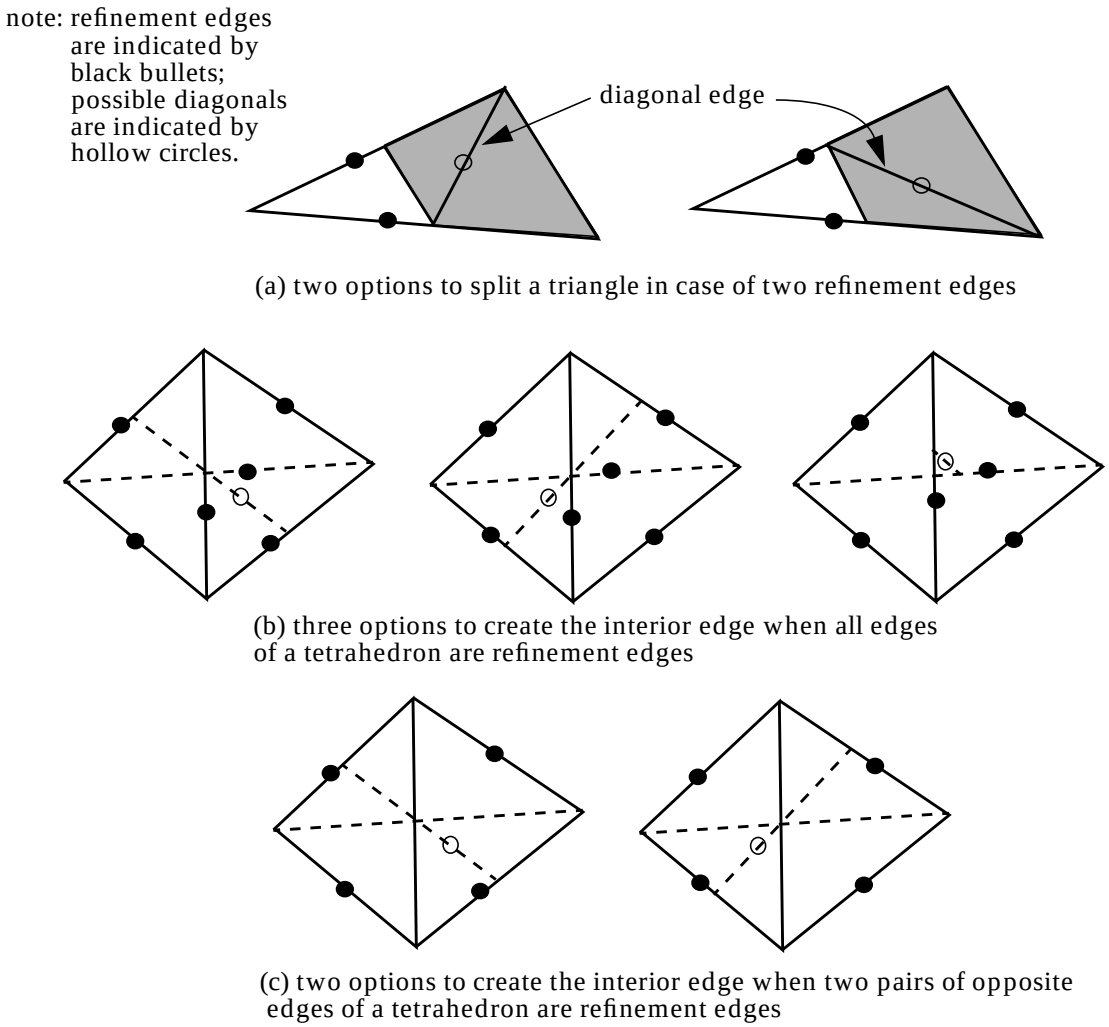


Figure 6.17 Ambiguity in creating diagonal edges.

left are hidden to view the interior). It can be seen that a long edge cross the top and bottom face is created, which makes the refinement unable to reduce the maximal mesh edge length and, in worst case, can lead to the divergence of the refinement iterations. Figure 6.18(c) shows the result mesh after three uniform refinements where the shortest diagonal edge is always created in case of ambiguity. It can be seen that all new tetrahedra are similar to the six initial tetrahedra in the initial mesh, and the maximal mesh edge length decreases to $1/8$.

Considering the anisotropic mesh metric field, a better option is to select the diagonal aligning with anisotropy. Figure 6.19 depicts a 2D example to illustrate this idea, where the ellipse

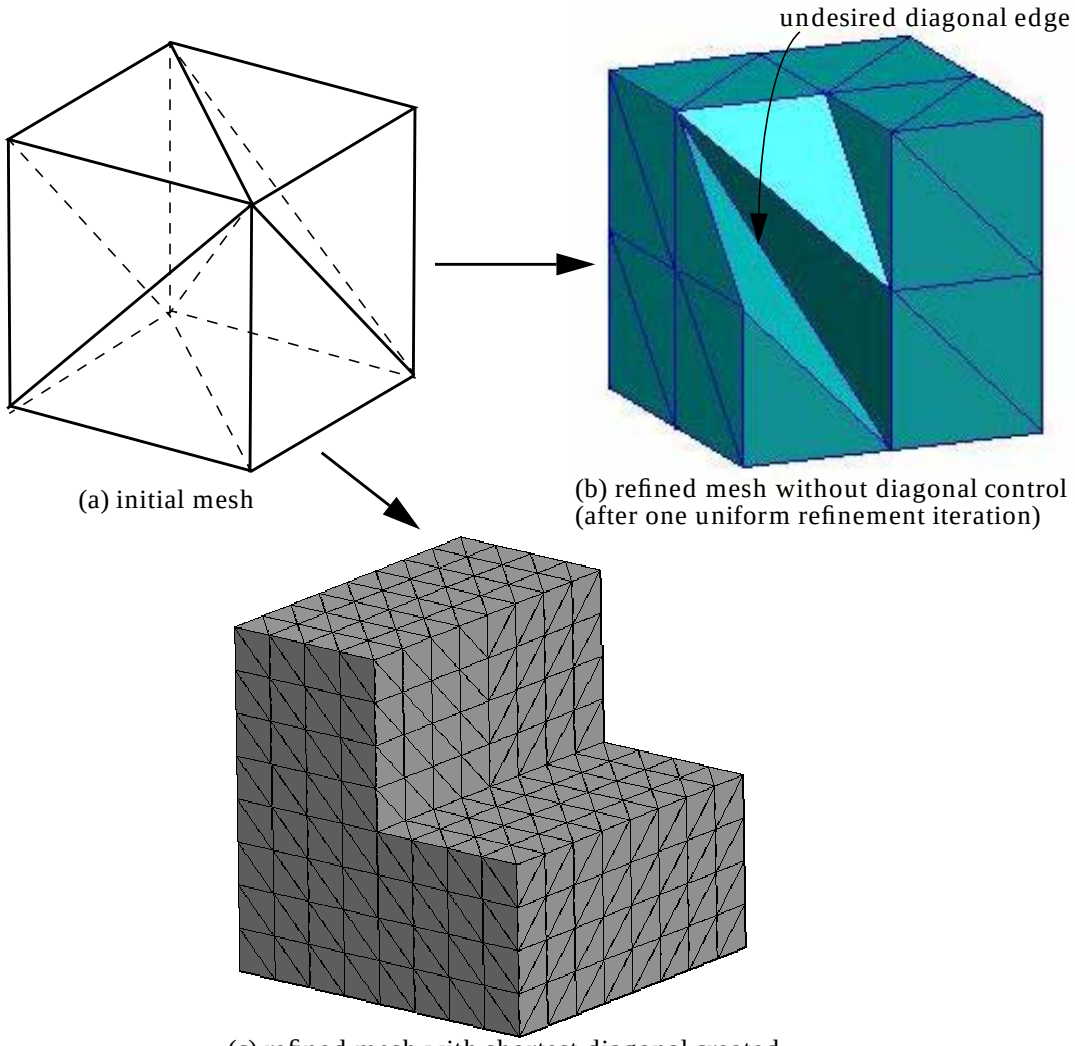


Figure 6.18 Example to show the need of creating short diagonal edge

indicates the given mesh metric over the shaded region with vector $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ be its principle directions. In the shaded quadrilateral area, either diagonal M_0^1 or M_1^1 can be created. The option of creating M_0^1 aligns to anisotropic better from two aspects: (i) it is shorter than M_1^1 in the transformed space; (ii) the minimal angle between M_0^1 and the two principle directions is smaller than that of M_1^1.

In current implementation, the diagonal length in transformed space is used to guide the selection of diagonals. Figure 6.20 shows the pseudo-code of the diagonal selection function, which is called by all ambiguous subdivision templates. The function accepts the mesh size field and two (or three) point pairs as input, compute the shortest distance between the two points in a

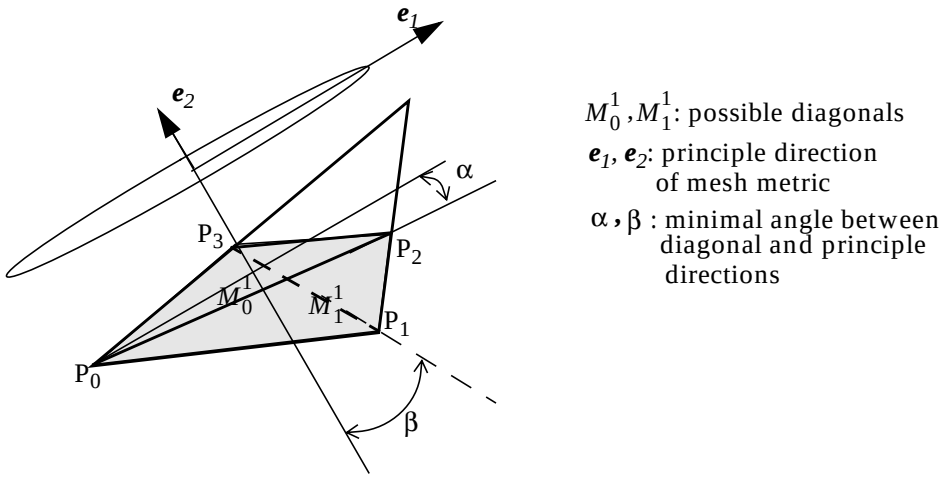


Figure 6.19 Create the diagonal aligning to anisotropy.

pair in transformed space and returns an integer indicating which point pair will creating the shortest edge in transformed space. Note that, since the diagonal edge or even the two end vertices of possible diagonals were not created when calling the function, here, a point pair is used to represent the two end points of a diagonal edge. For example in the setting of Figure 6.19, there are two point pairs, (P_0,P_2) and (P_1, P_3).

```
let (Pi, Qi) with i=1,n (n=2 or 3) be the given point pairs and Pi, Qi be points in i-th pair
initialize integer j to be 1
initialize variable dsq to be the distance square between P1 and Q1 in transformed space
for each i from 2 to n
    compute the distance square between Pi and Qi in transformed space
    if the computed distance square is less than dsq
       set dsq to be the computed distance square
       set j to be i
    end if
end for
return j
```

Figure 6.20 Pseudo-code of the diagonal selection function.

6.3.4 Lookup Table

Lookup tables and using binary integers as key to these tables have been used to improve the efficiency of subdividing elements.

Lookup tables are a collection of pre-defined tables that are closely related to subdivision templates and can be indexed using two keys. Figure 6.21 illustrates the usage of lookup tables in current implementation. Given a tetrahedron with refinement edges in arbitrary number and arbitrary order, two keys can be generated using bitwise operations: one is a six bit binary number (000000~111111), indicating the number and distribution of refinement edges; another is a four bit binary number (0000~1111), indicating the diagonal selection at the four bounding faces in case of ambiguity. These two keys uniquely determine a specific template and the needed lookup tables. Using the information in the lookup tables, the templates then have been implemented only using “create” and “delete” operations, without any relative expensive operations such as searching and adjacency interrogation.

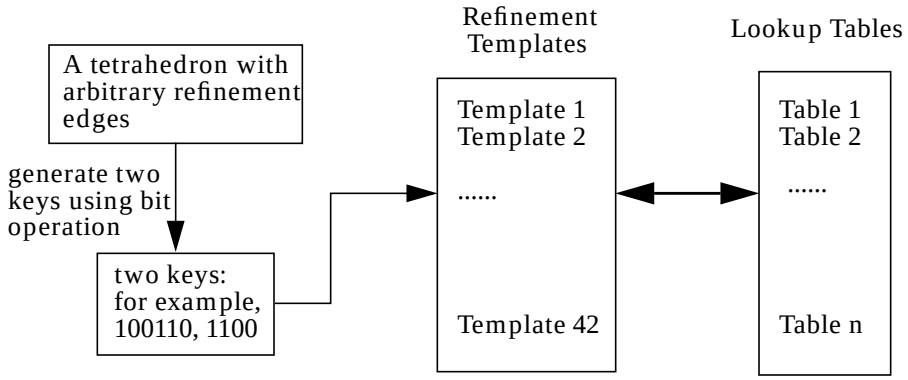


Figure 6.21 Using a lookup table to gain efficiency

6.4 Shape Correction

Because element quality may degrade during coarsening or placing vertices onto boundaries, local operations are included to allow the mesh to be modified to better match the desired mesh size field.

Local mesh modifications, particularly edge swap operators, can be used to improve the mesh by replacing poor elements with better ones. In case swap operations can not be applied, additional local mesh modifications, e.g. split, relocation and collapse operations have shown to be useful [23,25,65]. The key issue in applying these local mesh modifications lies in the effective determination of the most appropriate local mesh modification without the cost of evaluating a large number of possible operations.

6.4.1 Outline of Shape Correction Algorithm

Figure 6.22 gives the outline of the algorithm in pseudo-code form. Given a quality threshold, the algorithm first initializes a dynamic list and collects all elements with quality below the threshold into the dynamic list. Then process each element in this dynamic list using tetrahedron analysis techniques described in Section 6.4.2 and Section 6.4.3 until the list is empty. Whenever a tetrahedron in this list is processed, the dynamic list is updated to remove the tetrahedron from the list. If a local mesh modification is executed, update the dynamic list to remove the deleted elements and insert any newly created elements with quality below the threshold.

Although the dynamic list is kept updating in the application of local mesh modifications, the termination is guaranteed since, no matter successful or not, any processed element is removed from the list and local mesh modification is applied only if the local mesh quality improves.

```
initialize a dynamic list
insert all elements with quality below a given threshold into the dynamic list
while the dynamic list is not empty
    pop out an element from the list
    analyze the popped element (see Section 6.4.2)
    determine the best mesh modification to eliminate the popped element
    if the best mesh modification exists and can improve element quality
       apply the best mesh modification
       update the dynamic list
    end if
end while
```

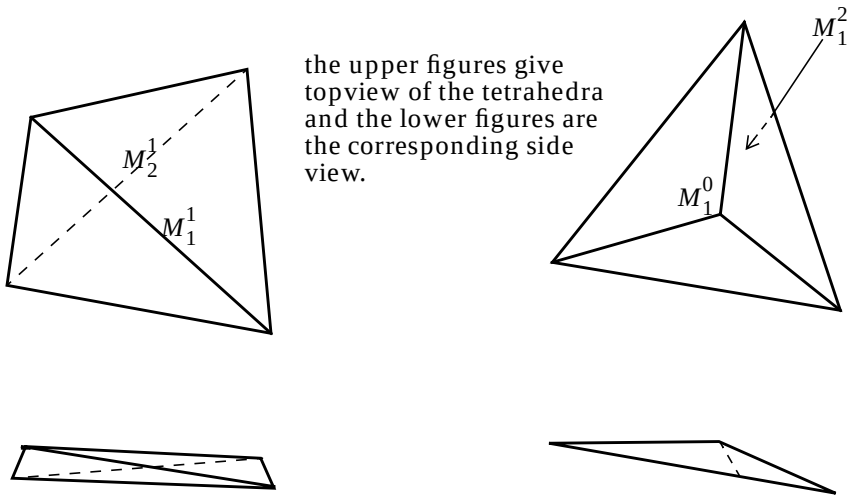
Figure 6.22 Pseudo-code of the shape correction algorithm.

6.4.2 Analysis of Sliver Tetrahedra

6.4.2.1 Classification

Sliver tetrahedron is characterized by very small volume without any relatively short edge that bounds the tetrahedron in the transformed space. The possible types of sliver tetrahedra are limited. To support the intelligent selection of local mesh modifications, it is useful to classify tetrahedra into two types[1] (refer to Figure 6.23). A tetrahedron is classified as type I sliver if two opposite edges of the tetrahedron almost intersect; A poor quality element is classified as type II if one vertex of the tetrahedron is close to the centroid of its opposite face.

1. Similar classification can also be found in references [25,52].



(a) Type I: M_1^1 and M_2^1 almost intersect (b) Type II: M_1^0 is close to the centroid of M_1^2

Figure 6.23 Classification of sliver tetrahedra.

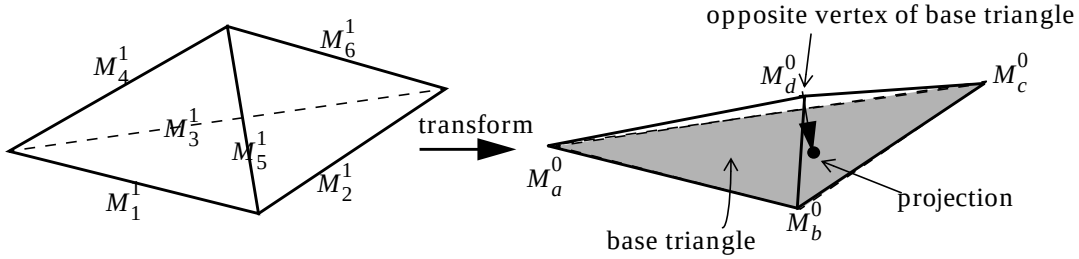
Such a classification is useful to the determination of local mesh modification since it clearly indicates the reason why the tetrahedron is of poor quality, therefore, pointing to the possible operations to eliminate the sliver tetrahedron. In particular, for type I, the reason is the small distance between a pair of opposite edges of the tetrahedron, so one (or both) of the edges must be eliminated (the key entities are two edges); For type II, the key reason is that a vertex is too close to its opposite face, so either the vertex or the face must be eliminated (key entities are a vertex and a face).

6.4.2.2 Determination of Classification and Key Entities

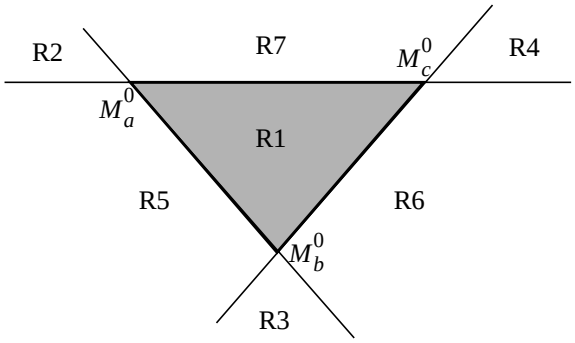
There are at least two methods to identify the two sliver types. One method is to compute the dihedral angles of the poor quality tetrahedron in the transformed space. From Figure 6.23, it can be seen that two dihedral angles associated with a pair of opposite edges should be close to 180 degrees in case of type I; and three dihedral angles associated with three converged edges should be close to 180 degrees in case of type II. Another relatively inexpensive method used in this context is based on projection in transformed space. Figure 6.24(a) shows a simple construct used to explain the projection method. To identify between the two types, first a base triangle (shaded) is determined, which can be any mesh face of the tetrahedron; then the vertex of the tetrahedron opposite to the base face is projected onto the base face in the transformed space. The type and key mesh entities of the tetrahedron can be determined in terms of the projection on the base triangle plane in the transformed space. Figure 6.24(b) shows the top view of the base triangle

plane, which is subdivided into seven subareas defined by extending the mesh edges of the base triangle. There is one subarea associated with each edge, one associated with each vertex, and the base triangle itself. Table 6.2 indicates the tetrahedron type and key mesh entities in terms of where the projection locates. The projection must not be on any mesh vertices of the base triangle since all mesh edge are in a length level close to one and volume of the sliver is almost zero in the transformed space.

Note that, as long as the volume of the tetrahedron is almost zero, the selection of base triangle has no effect on the analysis results given in Table 6.2. For instance in the example depicted in Figure 6.24, if the face bounded by vertex M_a^0, M_b^0 and M_d^0 were selected as the base triangle and vertex M_c^0 were projected onto it, the projection would be in a subarea associated with vertex M_d^0, and the key entities would still be M_d^0 and its opposite face.



(a) tetrahedron in two space: physical space (left), transformed space (right)



(b) Top view of the base triangle

Figure 6.24 Determination of tetrahedron type in terms of projection.

6.4.3 Intelligent Selection of Local Mesh Modifications

Given a sliver tetrahedron, the process of determining an appropriate local mesh modification to eliminate the sliver consists of three priority levels:

Table 6.2 Type and key mesh entities in terms of projection location.

Projection location	Type	Key mesh entities
R1	I	M_d^0 and its opposite face
R2		M_a^0 and its opposite face
R3		M_b^0 and its opposite face
R4		M_c^0 and its opposite face
R5	II	M_1^1 and M_6^1
R6		M_2^1 and M_4^1
R7		M_3^1 and M_5^1

- check a small number of swap operations. In case of type I sliver, this is to evaluate swapping the two key mesh edges of the sliver; In case of type II sliver, this is to evaluate swapping the key face and the three edges that bound the key face (note that swapping any edge that bounds the key face also can eliminate the face).
- if swap operation is unable to improve local mesh quality, further check compound operations. For type I sliver, this includes: splitting both key mesh edges and collapsing the new interior mesh edge (double split collapse operator); splitting either of the key edge, followed by relocating or collapsing the new vertex (split collapse and split reposition operator); For type II tetrahedron, it includes: splitting the key face followed by collapsing or relocating the new vertex (split collapse and split reposition operator).
- further check vertex repositioning operation in case compound operation can not be determined. For type II sliver, it is to relocate the key vertex; For type I sliver, relocating any vertex that bounds the tetrahedron is tested (refer to Section 4.1.4 for the computation of target location).

The reason of evaluating local mesh modifications in three levels is to make the effective local mesh modifications to be evaluated first, therefore avoid extensive evaluations and gain efficiency. Since swap operation is the most effective local mesh modification to eliminate slivers (see results of the first two examples in Section 6.5), and the possible swap operations are limited (only two possibilities in case of type I and four possibilities in case of type II), swap operation is set as the first to be evaluated. The second level evaluates specific sets of compound local mesh modifications to increase the chance of success. It involves the evaluation of five compound operations in case of type I, or two evaluations in case of type II. The third level evaluates vertex motions.

Although, in practice, these three levels of local mesh modification usually are enough to improve mesh quality to a reasonable level, the elimination of the sliver can not be ensured yet due to the heuristic nature of local mesh modifications.

The determined local mesh modification is applied only if it improves local mesh quality and does not violate the length criteria. Also, after the application of swap or compound operations, the length of new mesh edges are examined in transformed space. Attempts are made to collapse them in case of new short mesh edges.

6.5 Examples

The mesh adaptation procedure presented in this chapter has been tested against a wide range of models under either analytical or piecewise linear mesh size field definition(s). Three of them are given in this section to demonstrate effectiveness and technical aspects of this mesh adaptation procedure. In all these examples, the default parameter values of the mesh adaptation procedure are used. In particular, this includes: (i) parameter α that control length reduction rate (see (EQ-6.1)) is 0.75, (ii) the allowed mesh edge length interval in transformed space is [0.5,1.4], and (iii) using cubic mean ratio as element quality measure (see (EQ-2.15)) with quality threshold to be 0.008.

6.5.1 Expanding Spherical Shock

Figure 6.25 shows a $1 \times 1 \times 1$ cubic domain and a uniform initial mesh of the domain. The coordinate system and the vectors in Figure 6.25(a) depict the setting of specifying anisotropic mesh size field. For a point P of coordinate *(x,y,z)*, the anisotropic mesh size in transformation form is

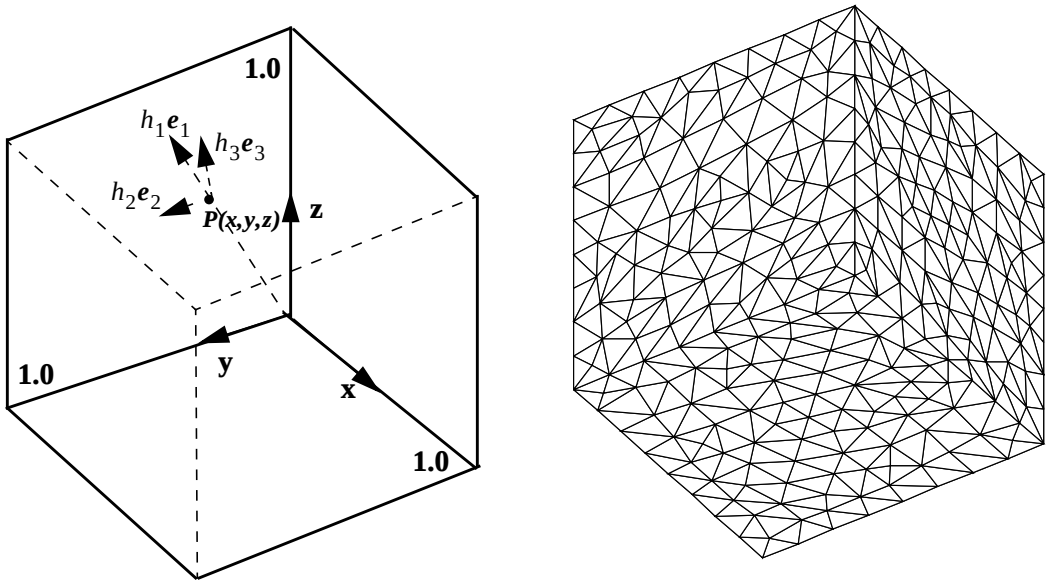
$$\boldsymbol{Q}(x, y, z) = \begin{bmatrix} \boldsymbol{e}_1 \\ \boldsymbol{e}_2 \\ \boldsymbol{e}_3 \end{bmatrix} \begin{bmatrix} 1/h_1 & 0 & 0 \\ 0 & 1/h_2 & 0 \\ 0 & 0 & 1/h_3 \end{bmatrix}$$

where $\boldsymbol{e}_i$ ($i=1,2,3$) are three orthonormal vectors at point P and h_i ($i=1,2,3$) are the specified mesh edge length in these three directions. To specify a spherical anisotropic mesh size field, the expressions of $\boldsymbol{e}_i$ and h_i ($i=1,2,3$) can be:

$$\begin{cases} h_1 = 0.125(1 - e^{-3|r^2 - t^2|}) + 0.00125 \\ h_2 = h_3 = 0.125 \end{cases}$$

$$\boldsymbol{e}_1 = \left[\frac{x}{r}, \frac{y}{r}, \frac{z}{r}\right]$$

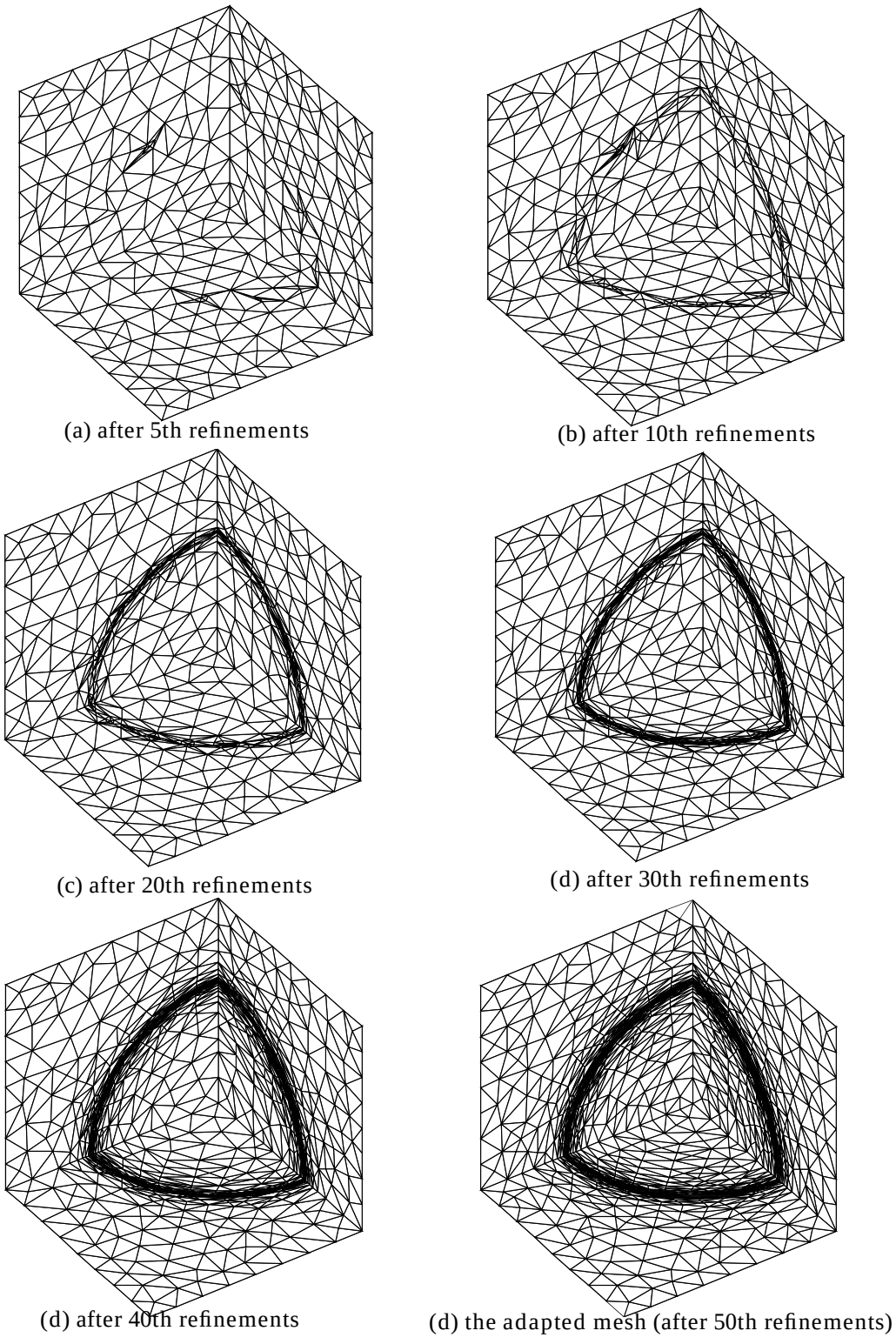
with $\boldsymbol{e}_2$ be any unit vector orthogonal to $\boldsymbol{e}_1$, and $\boldsymbol{e}_3 = \boldsymbol{e}_1 \times \boldsymbol{e}_2$. Here $r = \sqrt{x^2 + y^2 + z^2}$ and t is a given parameter that can control the move of the anisotropy (Note that $\boldsymbol{e}_1$ would be a zero vector at origin, where an isotropic mesh size of 0.125 can be used instead since $h_1 = h_2 = h_3$).



(a) domain (b) initial mesh

Figure 6.25 Domain and initial mesh of exploding spherical shock example.

Figure 6.26 shows a series of intermediate meshes towards satisfying the spherical anisotropic mesh size field of t=0.6 and the final adapted mesh. Since the uniform initial mesh is coarse, no mesh modification is applied in the first coarsening stage. Figure 6.26(a)-(e) give the intermediate meshes after 5th, 10th, 20th, 30th and 40th refinement iterations, respectively. Figure 6.26(f) gives the final adapted mesh. The template-based refinement algorithm is totally applied fifty times to obtain the adapted mesh. Figure 6.27 indicates the history of maximum transformed mesh edge length in the mesh during these refinement iterations. It can be seen that the maximum transformed length of the mesh is reduced from 16.18 to 1.4 in a fluctuated fashion. The fluctuation is reasonable since, in this example, the anisotropy to be captured is in a narrow range and the length in transformed space is computed numerically, so edges across the narrow range of anisotropy may be missed in the beginning. Figure 6.28 indicates the number of refinement edges at each refinement. At maximum 3215 mesh edges have been split in one refinement iteration, while



(a) after 5th refinements

(b) after 10th refinements

(c) after 20th refinements

(d) after 30th refinements

(d) after 40th refinements

(d) the adapted mesh (after 50th refinements)

Figure 6.26 Intermediate and final meshes to spherical mesh size field of t=0.6.

at minimum only two mesh edges. It should be pointed out that the number of refinement edges increases slowly in the beginning since the anisotropy to be captured is in a narrow range and the edges of the initial mesh crossing the range are not aligned to the anisotropy. It totally takes 27.7 seconds to refine the uniform initial mesh of 6860 tetrahedra into the adapted mesh of 40250 tetrahedra on a Sun Fire 280R machine (CPU: UltraSparc-III+; System clock frequency: 150MHz; CPU clock frequency: 900MHz; Memory: 10240M).

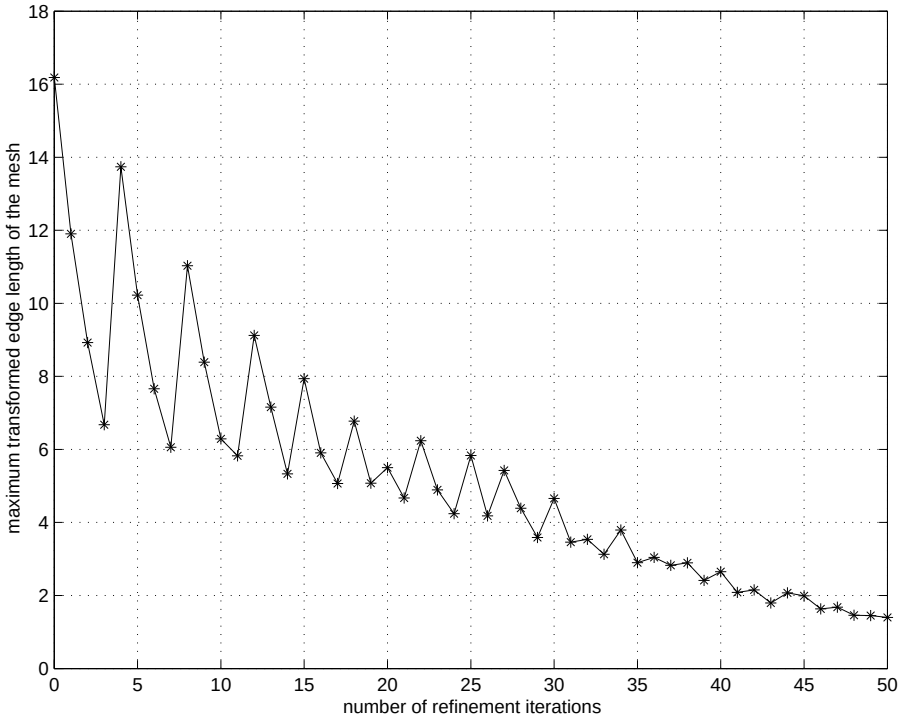


Figure 6.27 History of maximal transformed edge length in refinements.

Table 6.3 gives statistics of performed local mesh modifications in the shape correction stage. Edge swap is the most effective operation, followed by splitting plus repositioning operation, double split plus collapsing operation, single split plus collapsing operation and face swap. The total cost of the shape correction stage is 9.77 seconds on the same Sun Fire 280R machine.

Table 6.3 Statistics of performed mesh modifications in shape correction stage.

	Edge Swap	Split + reposition	Double Split + Collapse	Split + Collapse	Face swap
No. of mesh modifications	2283	63	19	11	10

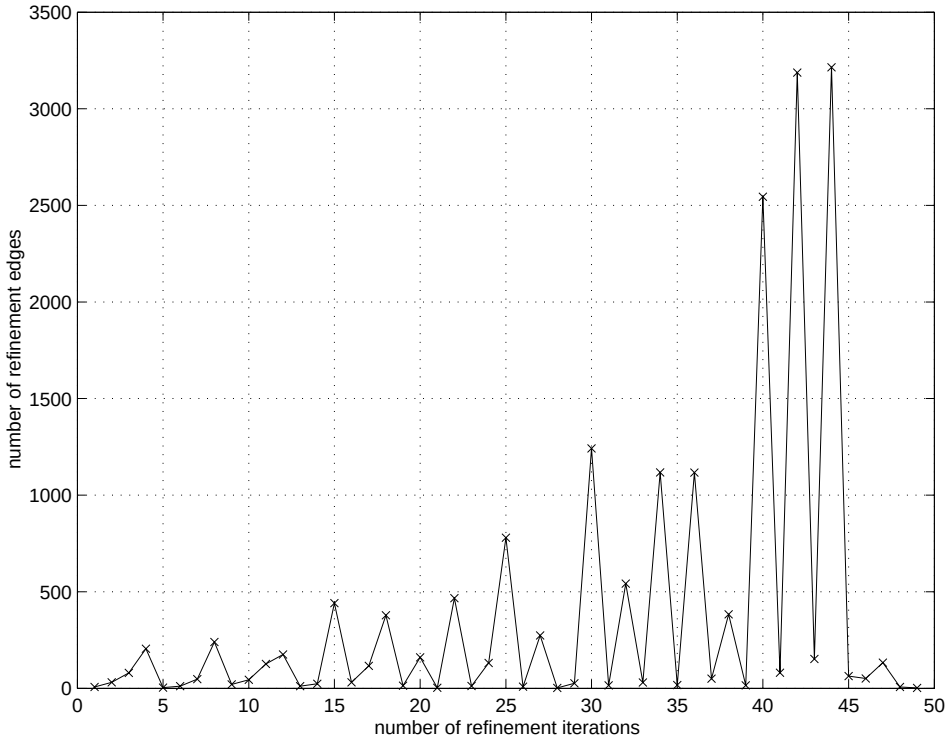
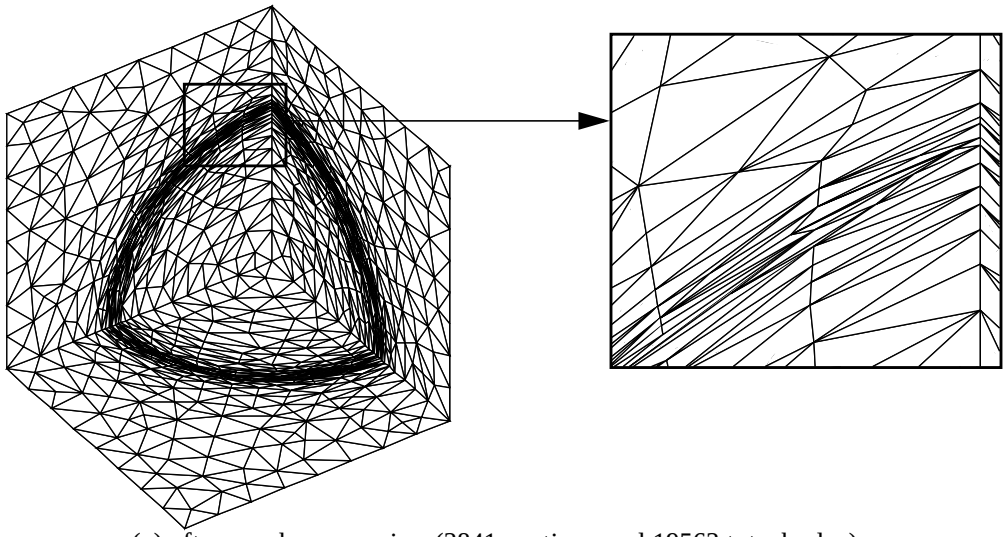


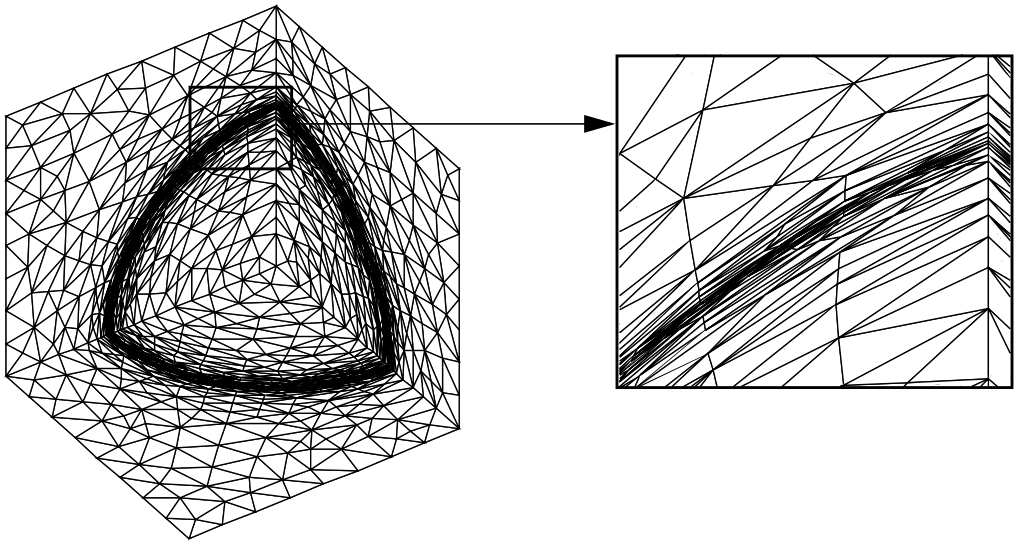
Figure 6.28 Number of refinement edges at each refinement iteration.

Figure 6.29 shows a new intermediate mesh, a new adapted mesh and their close-up views when the parameter *t* of the spherical mesh size field is changed from *0.6* to *0.62*. Since the radial desired mesh edge length changes from *0.00125* to *0.011* at $r = 0.6$, mesh coarsening is applied nearby $r = 0.6$ in the first stage of mesh adaptation algorithm. Totally 3758 edge collapsing operations are applied, decreasing the number of tetrahedra in the mesh from 40250 to 19563. Figure 6.29(a) shows the mesh after mesh coarsening. From the close-up view it can be seen that the spherical anisotropy is respected during coarsening. Figure 6.29(b) shows the adapted mesh after the 20th application of the refinement iteration. It refines tetrahedra nearby $r = 0.62$ and increases the number of tetrahedra in the mesh to 36839.

Table 6.4 summarizes the timing statistics of this example obtained on the same Sun Fire 280R machine. It can be seen that moving anisotropy from a previous anisotropic mesh is more efficient than catching the anisotropy from a coarse uniform mesh.



(a) after mesh coarsening (3841 vertices and 19563 tetrahedra)



(b) adapted mesh after the 20th refinement (7033 vertices and 36839 tetrahedra)

Figure 6.29 Intermediate and adapted meshes for spherical size field of t=0.62.

Table 6.4 Timing statistics for exploding spherical shock example[a].

t parameter of the spherical size field	time (seconds)			total time (seconds)
	coarsening	refinement	improving	
0.6	0	27.77	9.77	37.54
0.62	4.90	13.01	5.97	23.88

a. Data were collected on a Sun Fire 280R machine (CPU: UltraSparc-III+; System clock frequency: 150MHz; CPU clock frequency: 900MHz; Memory: 10240M).

6.5.2 Meshing of Triple Parachute Domain

Figure 6.30 shows the surface meshes of three parachutes and a box domain containing these parachutes[1]. Positions of the box are indicated by their coordinates and the front face of the box is hidden to show the interior parachutes. Figure 6.31 shows a not smoothly graded initial volume mesh of the domain, consisting of 375,782 tetrahedra and 62,183 vertices. To better visualize the tetrahedral mesh, Figure 6.31 only shows the surface meshes of three outer boundary faces and all interior mesh faces interacting with plane *y=0* (Here, a mesh face is defined as interacting with a plane if it does not intersect the plane but a tetrahedron it bounds intersects. Note that these mesh faces interacting with the plane are not coplanar and this 3D visualization techniques available from Simmetrix Inc. will be used a lot hereafter).

The role of the mesh adaptation procedure is to create a smoothly graded tetrahedral mesh by adapting the initial mesh to a smooth mesh size field. Since the desired isotropic mesh size on the outer boundaries and on canopies can be obtained from the given surface mesh, such a smooth desired isotropic mesh size field between outer boundaries and canopies can be defined by properly interpolating the known mesh sizes on surface meshes (the appropriate definition of the smooth mesh size field is not the focus of this example).

A smoothly graded tetrahedral mesh consisting of 1,332,330 tetrahedra and 235,796 mesh vertices has been generated by the mesh adaptation procedure. Figure 6.32 visualizes this adapted mesh by showing interior mesh faces of tetrahedra interacting with two planes. Figure 6.32(a) provides a side view by showing the mesh faces interacting with a vertical plane that cuts through two parachutes (the plane of *x=0.6*), while Figure 6.32(b) provides a top view showing the mesh faces interacting with horizontal plane *z=0*. Clearly, Mesh size grades smoothly from the outer box boundaries to canopies in all directions. Figure 6.33 shows the history of maximum transformed mesh edge length vs. refinement iterations. It can be seen that only seven refinement iterations are needed to achieve desired mesh size and the maximum transformed mesh edge length decreases monotonically from 8.1 to 1.4. Table 6.5 indicates the number of refinement mesh edges at each refinement iteration. The seven refinement iterations totally cost 99.11 seconds with 58.71 seconds on template-based refinement and 40.40 seconds on collapsing new short edges the refinement templates create (timing statistics are obtained on the same Sun Fire 280R platform).

---

1. Courtesy of Sunil Sathe and Prof. Tayfun E. Tezduyar of T*AFSM, Rice University.

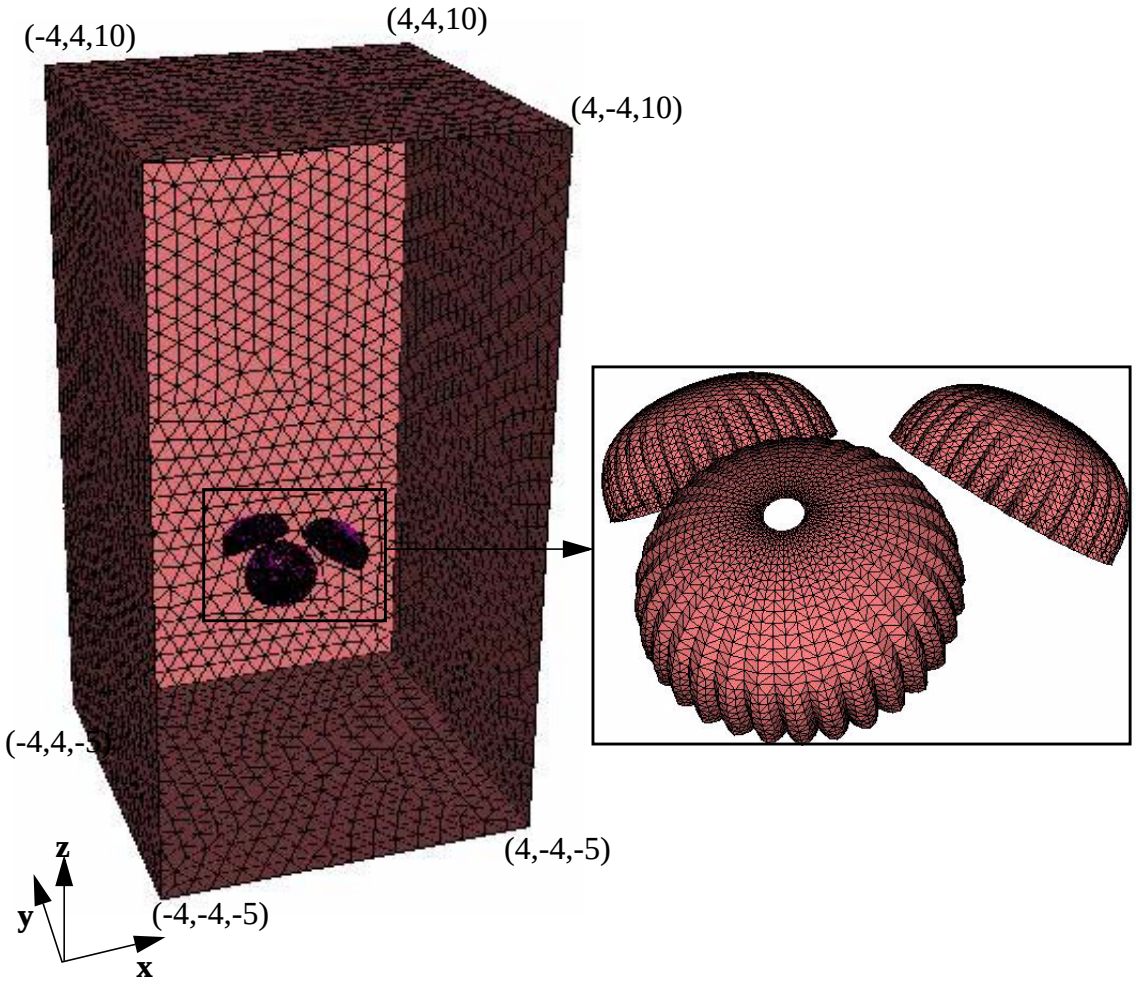


Figure 6.30 A box domain with three parachutes.

Table 6.5 Number of refinement mesh edges at each refinement iteration.

refinement iteration	1	2	3	4	5	6	7
refinement edges	240	1239	4118	14015	23433	79918	75849

After refinement iterations, only twenty-four tetrahedra is below the given quality threshold of 0.008 in terms of cubic mean ratio element quality measure, and all are eliminated by edge swap operations in the shape correction algorithm. The total cost of the shape correction stage is 26.54 seconds on the Sun Fire 280R machine, most of which are spent on computing the cubic mean ratio of 1.33 million tetrahedra.

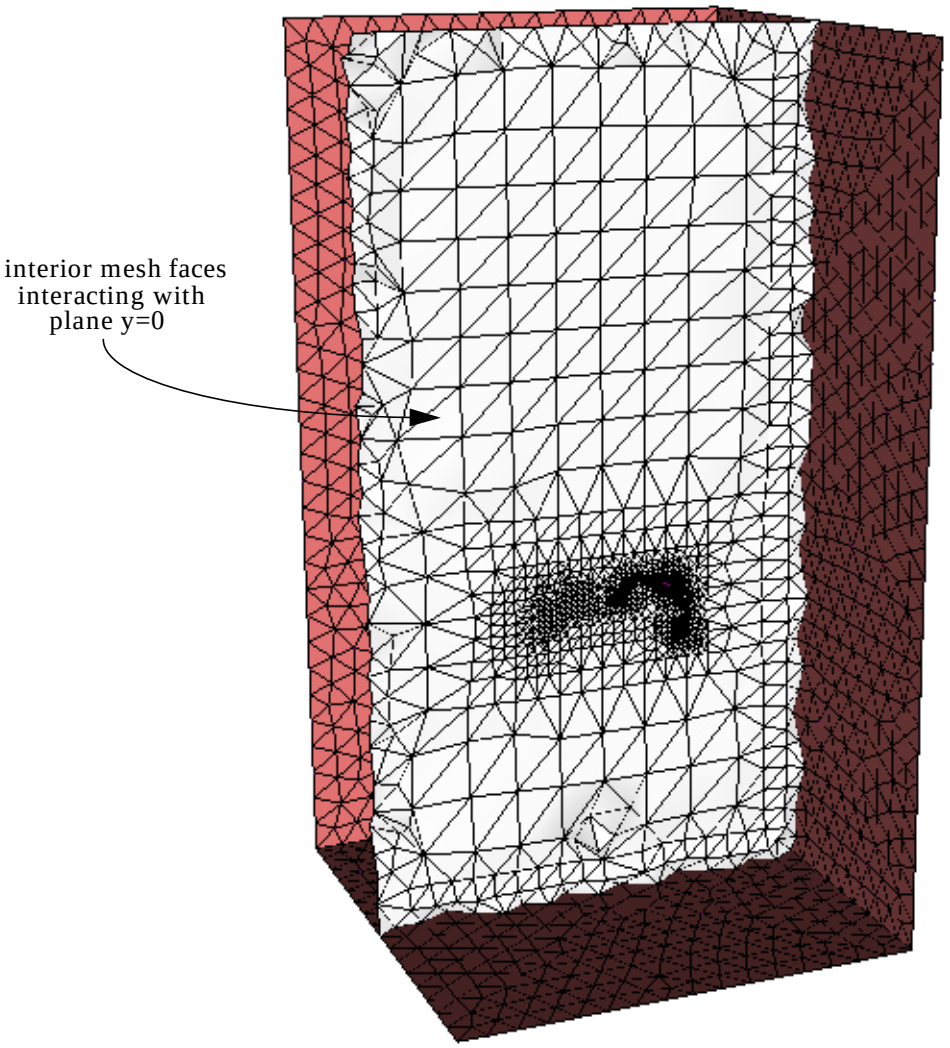


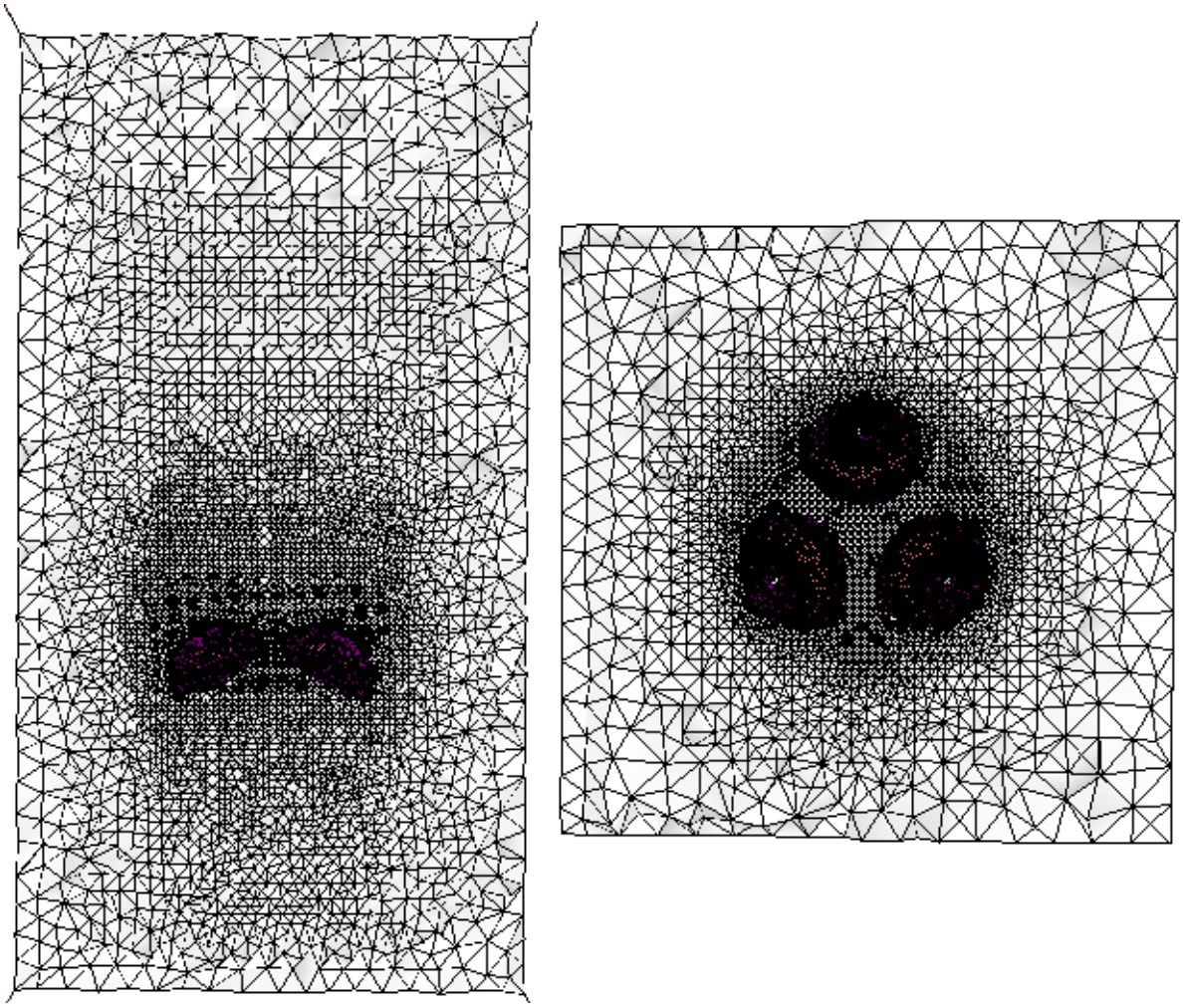
Figure 6.31 The initial tetrahedral mesh of box domain with three parachutes.

6.5.3 Circular Pipe

Figure 6.34 shows the model of a circular straight pipe and an initial mesh of the pipe. Following the coordinate system in Figure 6.34(a), the desired anisotropic mesh size field is (refer to Section 3.2.1),

$$\boldsymbol{Q}(x, y, z) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/h_r & 0 & 0 \\ 0 & 1/h_\theta & 0 \\ 0 & 0 & 1/h_z \end{bmatrix}$$

with



(a) mesh faces interacting with plane x=0.6 (b) mesh faces interacting with plane z=0

Figure 6.32 The adapted mesh of the box domain with three parachutes.

$$\begin{cases} h_r = 0.2(1 - e^{2(x^2 + y^2 - 1)}) + 0.003 \\ h_\theta = h_z = 0.2 \end{cases}$$

where $\cos\theta = x/r$, $\sin\theta = y/r$ and $r = \sqrt{x^2 + y^2}$.

Figure 6.35 shows the result mesh that conforms to the anisotropic mesh size field as well as respects the cylindrical surface. Figure 6.35(a) shows the mesh faces on the plane of *z=0* and on the cylindrical surface, while Figure 6.35(b) shows the interior mesh faces interacting with the

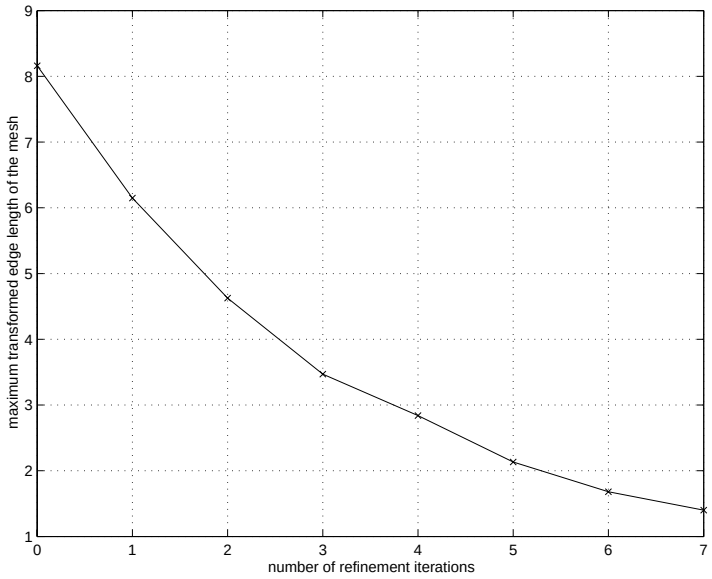
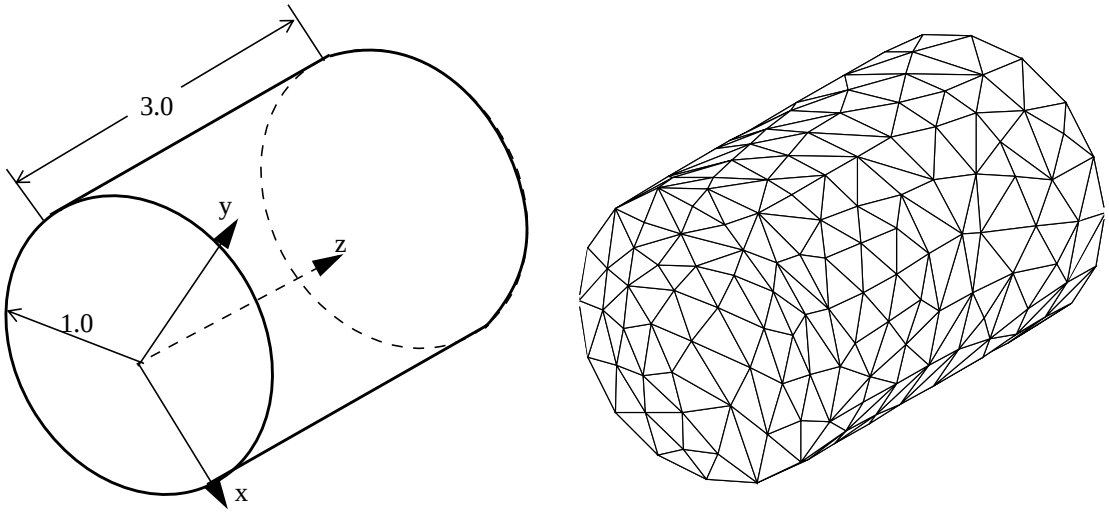


Figure 6.33 History of maximal transformed edge length of parachute example.



(a) cylindrical domain (b) Initial mesh

Figure 6.34 Model and initial mesh of circular pipe example.

plane of *x=0*. It can be seen that all vertices are placed onto their classified boundary and the mesh is anisotropically refined nearby the cylindrical surface. This result mesh is obtained after twenty-two refinement iterations. Figure 6.36 indicates the history of the maximum transformed mesh edge length during these refinement iterations. Again, the maximum transformed edge length is monotonically decreased from 51.3 to 1.4 since no edge can across the anisotropy. Table 6.6 indicates the number of refinement edges and the number of boundary mesh vertices to be placed

onto the cylindrical surface in the first twenty refinement iterations. Similar to results given in Section 5.6, most of the vertices are placed onto the cylindrical surface simply by a repositioning operation, while a small number of the vertices are incrementally placed onto the cylindrical surface after applying local mesh modifications. Particularly in this example, 121 edge collapsing and 19 edge swap operations are performed. Only one vertex needed to be placed onto boundary using the cavity remeshing algorithm.

Table 6.6 Number of refinement edges and snapping vertices vs. iteration.

iteration	1	2	3	4	5	6	7	8	9	10
N_e [a]	9	17	112	73	402	545	478	565	86	1130
N_v [b]	9	17	90	14	69	43	45	56	9	230
iteration	11	12	13	14	15	16	17	18	19	20
N_e	1570	510	2769	2758	652	3653	3726	10511	31	8
N_v	113	39	217	159	63	246	372	738	0	0

a. N_e **:** number of edges to be refined.
b. N_v **:** number of mesh vertices to be placed onto curved boundaries.

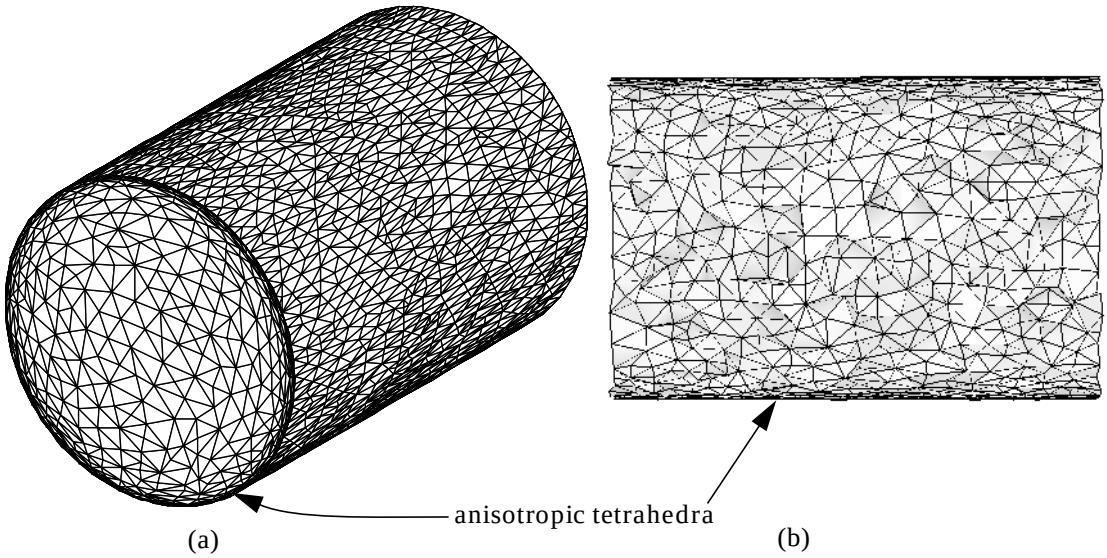


Figure 6.35 Adapted mesh with all boundary vertices snapped.

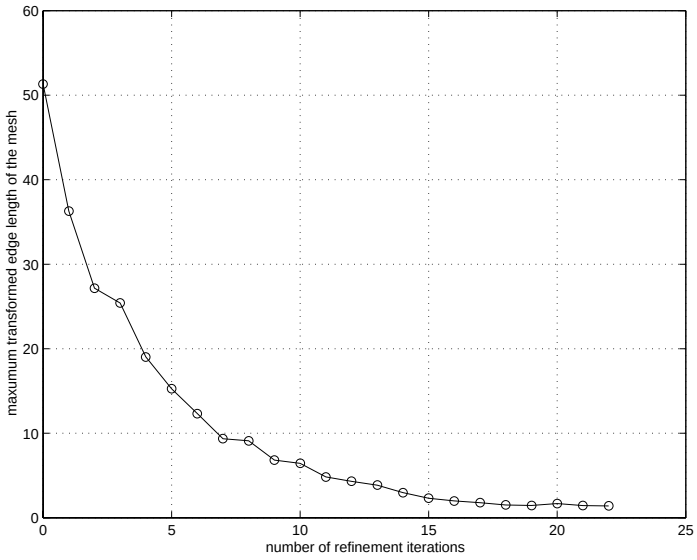


Figure 6.36 History of maximal transformed length of circular pipe example.

CHAPTER 7

Application in Adaptive Simulation

The mesh adaptation procedure has been embedded into three analysis codes[1] to perform *h*-version adaptive simulation for various applications. Those applications have, in many ways, driven the development of the mesh adaptation procedure. This chapter provides four examples to show its applications in anisotropic adaptive flow simulation.

The first two examples, steady flow in bifurcation blood vessel (Section 7.1) and unsteady surrounding flow near a parachute (Section 7.2), use stabilized finite element formulation developed by Jansen [90] and Hughes [18] to solve flow problem. Since linear tetrahedral elements are used, second order derivatives can be used to adaptively construct mesh metric field. In particular, the method in Section 20.8.1 of Reference [35] is adopted to compute second derivatives in a sense of distribution for classic C^0 finite elements, and new mesh metric field is constructed by decomposing and scaling the matrix of second derivatives using method described in Appendix A.

The last three examples, four contact Riemann problem (Section 7.3), cannon blast problem (Section 7.4) and backward facing step problem (Section 7.5), simulate transient flows with solution discontinuity (shock, expansion waves and etc.) using discontinues Galerkin method. In addition to defining mesh metric field using the same second derivative strategy in the portion of the domains without solution discontinuities, a two step procedure is used to construct the mesh metric field near the discontinuities: first, the location of elements crossing discontinuity are determined using a solution smoothness indicator; then the mesh metric aligning to the discontinuity are defined. Detailed descriptions of the adaptive definition of anisotropic mesh size field are given in references [70,71].

In all examples, the mesh metric fields are represented by the piecewise definition over the current mesh (see Section 3.2.2) and the anisotropic mesh size field smoothing procedure given in Section 3.5 of Reference [71] is applied to control the gradation of adaptive meshes. Furthermore, solutions on the previous mesh is locally transferred during the application of local mesh modifications, therefore they are maintained during mesh adaptation and the simulation can continue from the previous step using the adapted mesh.

1. PHASTA, DG and heat transfer application of Trellis.

7.1 Bifurcation Blood Vessel

The first example demonstrates the controlled anisotropic meshes appropriate for the cardiovascular system models. Figure 7.1 illustrates the geometric model domain approximating a portion of blood vessel with a symmetric bifurcation. At the inlet boundary, the inlet velocity profile indicated below is specified in the *z* direction and all other velocity components are set to zero.

$$u_3 = min(25(1-r), (1-r)^{1/7}) \tag{7.1}$$

where u_3 is the velocity in *z* direction and $r = \sqrt{x^2 + y^2}$ ($r \le 1$). At the two outlet boundaries, zero pressure boundary conditions are applied. At all other boundaries, no slip wall boundary conditions are enforced.

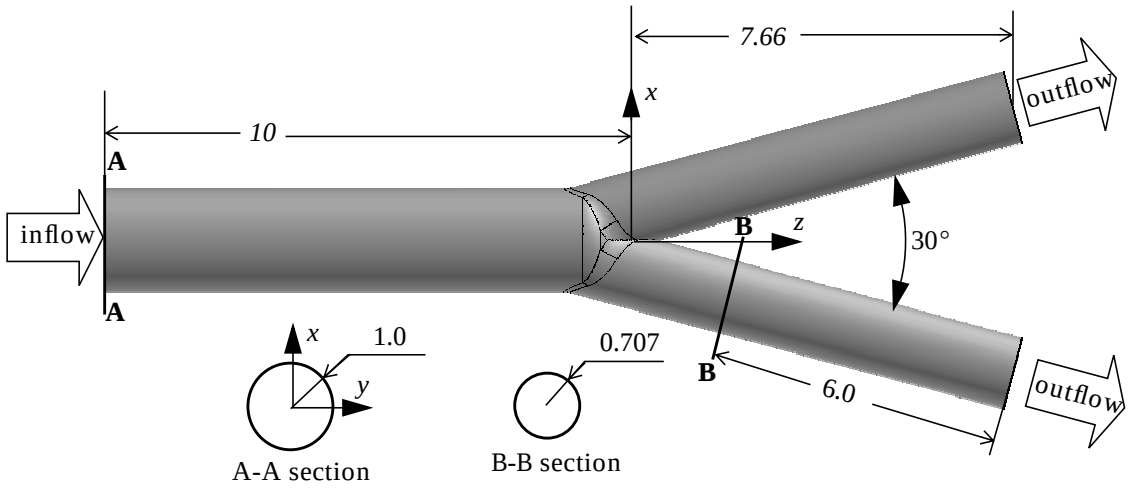


Figure 7.1 Schematic diagram of bifurcation blood vessel.

The advective form of time-dependent, incompressible Navier-Stokes equations is [90]:

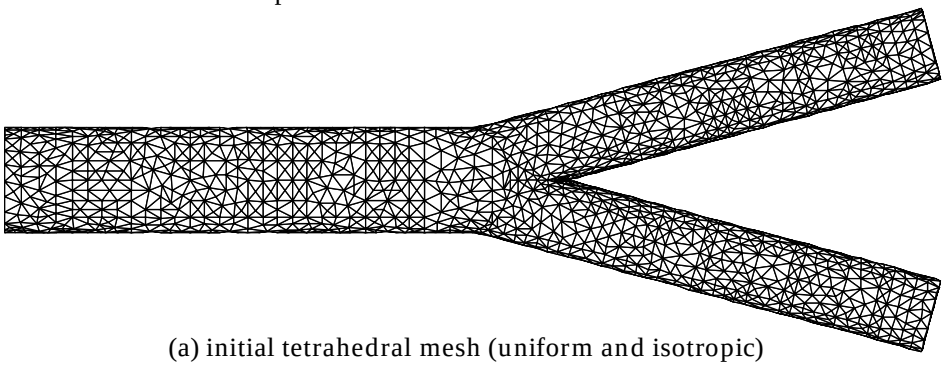
$$\begin{cases} u_{i,i} = 0 \\ \dot{u}_i + u_i u_{i,j} = (-p_{,i} + \tau_{ij,j} + f_i)/\rho \end{cases} \tag{7.2}$$

where *p* is the pressure, ρ is the density (assumed constant), u_i is the *i*th component of velocity, f_i is the prescribed body force, and τ_{ij} is the viscous stress tensor given by:

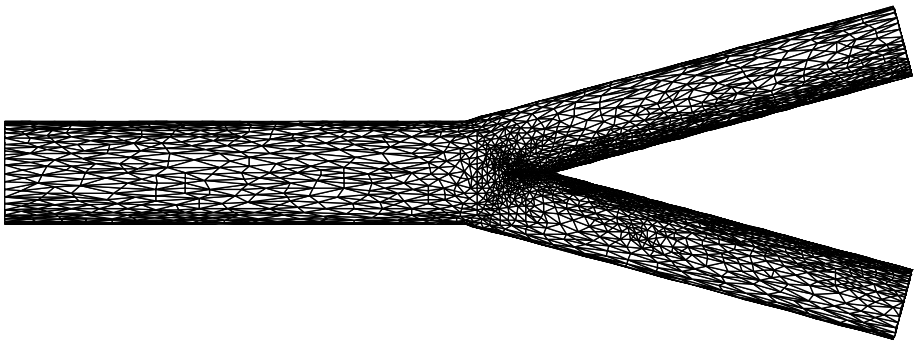
$$\tau_{ij} = \mu(u_{i,j} + u_{j,i}) \tag{7.3}$$

where μ is the kinematic viscosity. The blood vessel flow is simulated by solving this PDE using stabilized finite element formulations [18,90].

Figure 7.2 gives the top view of initial mesh and the anisotropically refined mesh after the fourth application of mesh adaptations. The initial mesh is uniform and isotropic. It is used to solve the flow problem in solution steps from 0 to 50 (0.5 seconds for each solution step). The refined mesh is achieved after the application of four anisotropic mesh adaptations at solution step 50, 80, 110, 140 respectively to equilibrate the spatial and directional distribution of leading interpolation error (see Appendix A), and it is used to solve the flow problem from step 140 to step 170. The number of elements is increased from the initial 38,903 to 47,257 to 71,082 to 129,990 to 270,753 in these four mesh adaptations.



(a) initial tetrahedral mesh (uniform and isotropic)



(b) anisotropically refined mesh after the 4th mesh adaptations

Figure 7.2 Coarse initial mesh and anisotropically refined mesh.

Figure 7.3 shows the refined mesh and its associated flow speed contours on two sections of the blood vessel model (see Figure 7.1 for the definition of the two sections). Figure 7.3(a) shows the surface mesh and the flow speed contour at inlet while Figure 7.3(b) shows the interior mesh faces and speed contour related to B-B section. Figure 7.4 shows the interior mesh faces interacting with the plane of $y = 0$ and the speed contour on the plane, including close-up views to the area

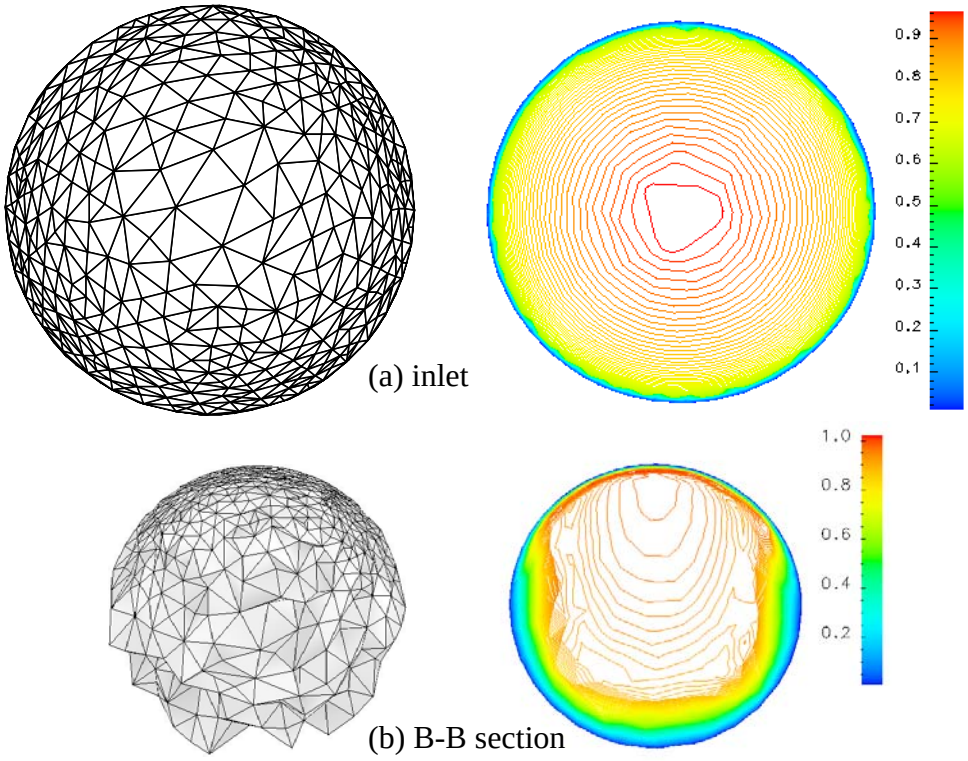
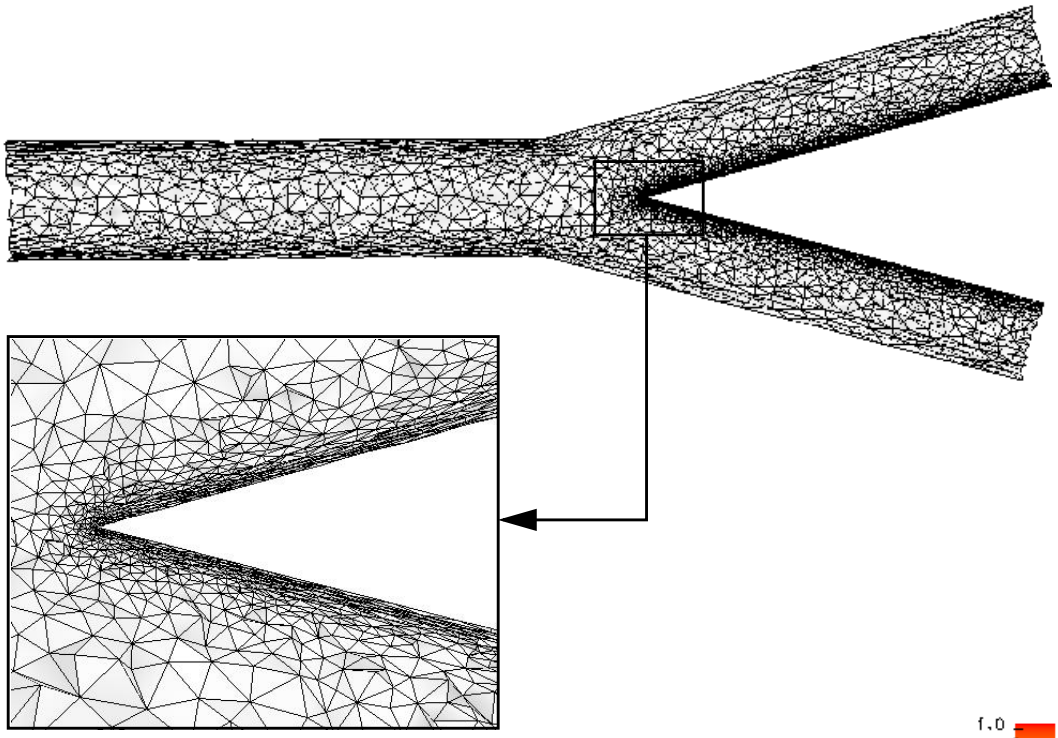


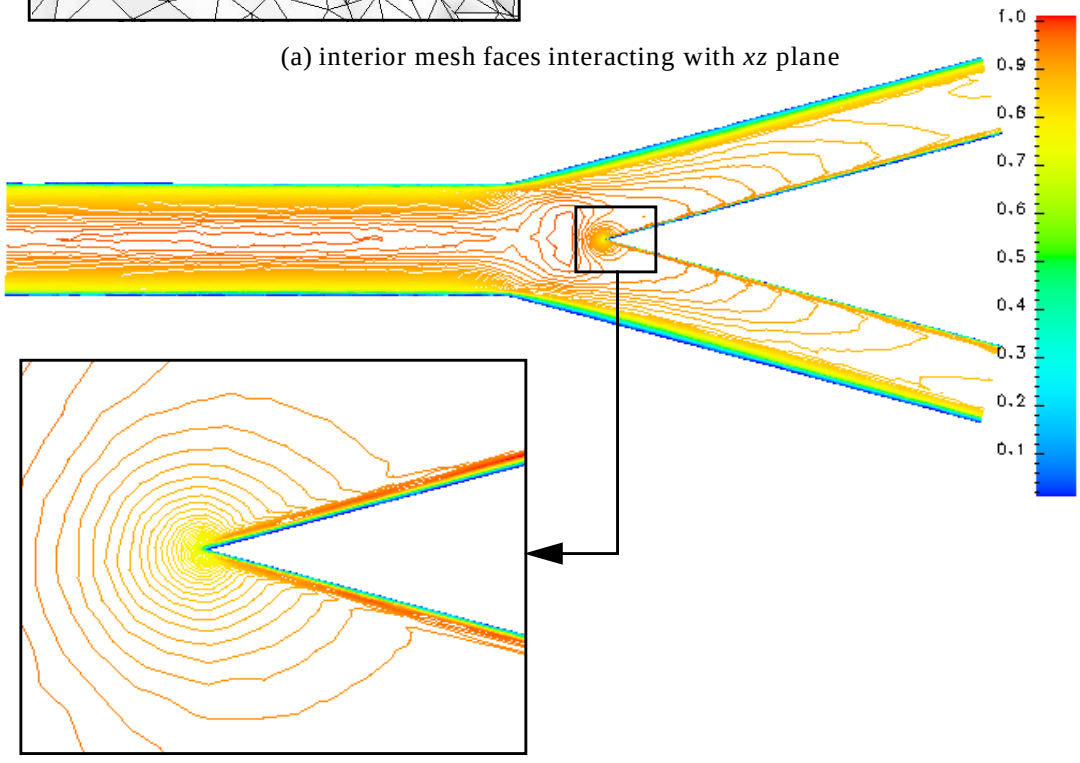
Figure 7.3 Mesh and flow speed contour on two blood vessel sections.

near the bifurcation point. It can be seen that mesh adaptation has created anisotropic elements in the boundary layer of wall surfaces (particularly, elements of a very small radial spacing, somewhat larger azimuthal spacing and larger still axial spacing), small isotropic elements at bifurcation and quickly developing anisotropy in the boundary layers downstream of the bifurcation, which catches both singularity and boundary layers as indicated by the speed contours.

Figure 7.5 shows the surface meshes on wall boundaries with the front cylindrical wall boundary in the downstream of the bifurcation hidden. Figure 7.5(a) shows the initial mesh faces on wall boundary. It can be seen that the wall boundary is rough (there are zigzags as indicated) and the geometry at bifurcation is not well approximated. Figure 7.5(b) shows that of the refined and snapped mesh faces. It can be seen that geometric discretization error has been improved when boundary edges in azimuthal direction are refined and modified to better align with the direction of zero curvature; and also the boundary mesh near bifurcation.



(a) interior mesh faces interacting with *xz* plane



(b) velocity contour on *xz* plane

Figure 7.4 Interior mesh faces and flow speed contour on the *xz* plane.

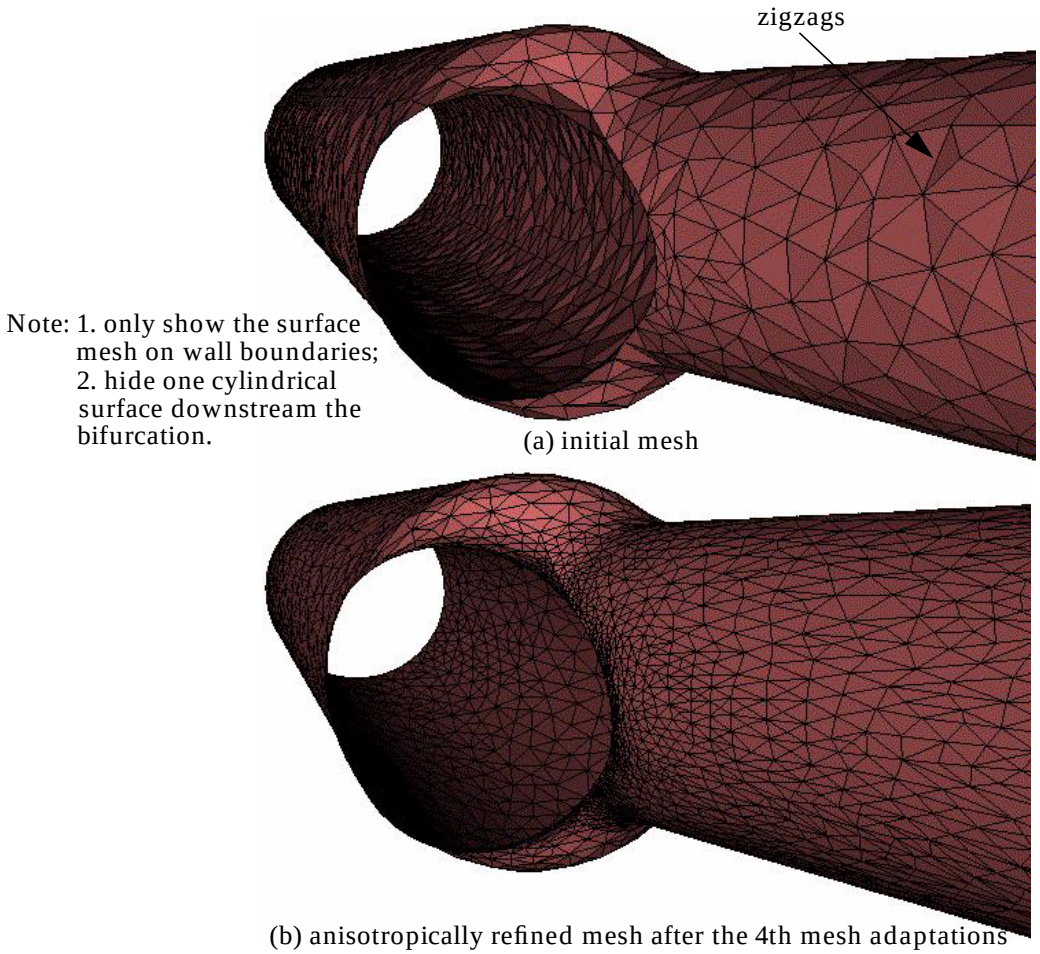


Figure 7.5 The refined and snapped mesh.

7.2 Flow around a parachute

The interaction between parachute system and the surrounding fluid field are dominant in parachute operations. 3D flow simulation to predict the surrounding flow using stabilized finite element formulations is of interest and feasible [42,84] using a carefully defined tetrahedral mesh that was not changed during simulation (see for example [84] and Section 6.5.2). Experience is needed to generate such a mesh and, to be conservative, the mesh is usually over-refined. This example presents alternative for the mesh control of the fluid domain near a parachute using the mesh adaptation procedure.

Consider a parachute with four cut-off at corners in a box domain as illustrated in Figure 7.6[1]. Let u_i (i=1,2,3) be the ith component of velocity, τ represent stress tensor, $(\boldsymbol{n} \cdot \tau)$ be the stress acting on the surface with normal $\boldsymbol{n}$, and $(\boldsymbol{n} \cdot \tau)_i$ indicate the ith component. The initial velocity field is simply $u_1 = u_2 = 0$, $u_3 = 1$ everywhere. The boundary conditions are applied:

- On parachute, no slip wall, i.e., $u_1 = u_2 = u_3 = 0$.
- At the inlet boundary, $u_1 = u_2 = 0$, $u_3 = 1$.
- At the outflow boundary, traction free, i.e., $(\boldsymbol{n} \cdot \tau) = 0$.
- At the two side box faces normal to x-axis, no slip in x direction, i.e., $u_1 = 0$, $(\boldsymbol{n} \cdot \tau)_2 = 0$ and $(\boldsymbol{n} \cdot \tau)_3 = 0$.
- At the two side box faces normal to y-axis, no slip in y direction, i.e., $u_2 = 0$, $(\boldsymbol{n} \cdot \tau)_1 = 0$ and $(\boldsymbol{n} \cdot \tau)_3 = 0$.

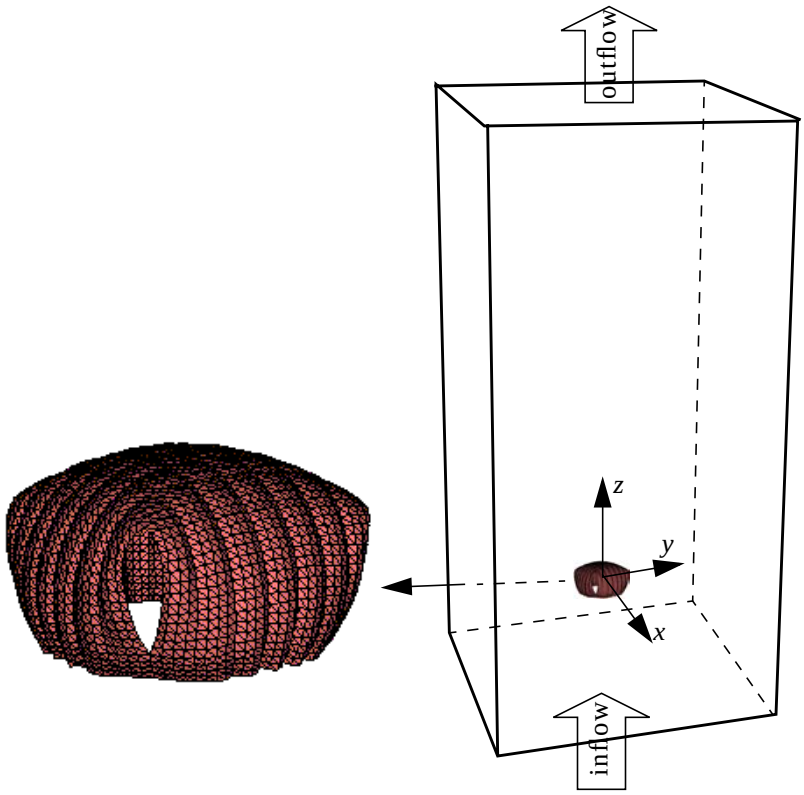


Figure 7.6 Schematic diagram for the parachute problem.

1. Courtesy of Sunil Sathe and Prof. Tayfun E. Tezduyar of T*AFSM, Rice University.

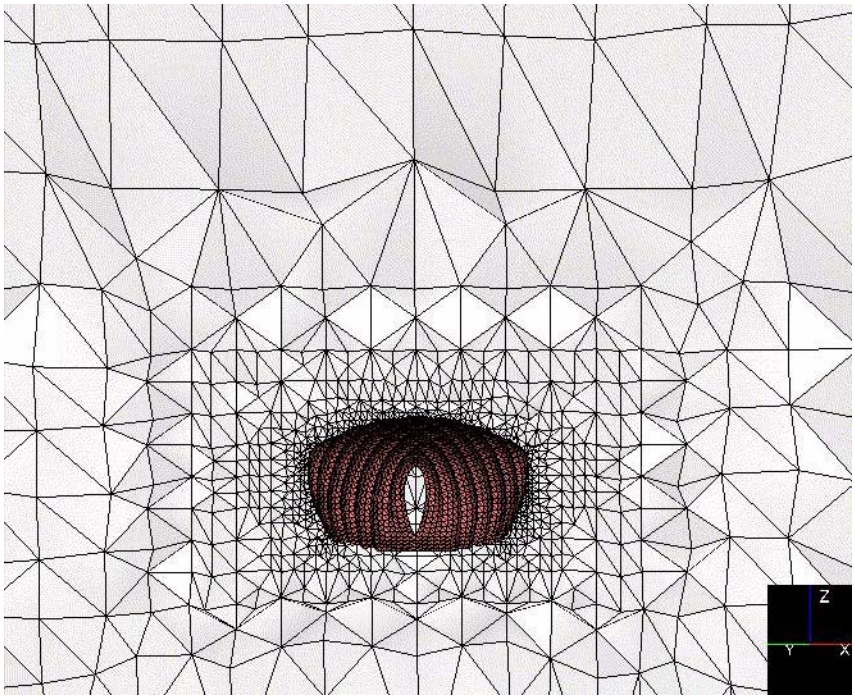
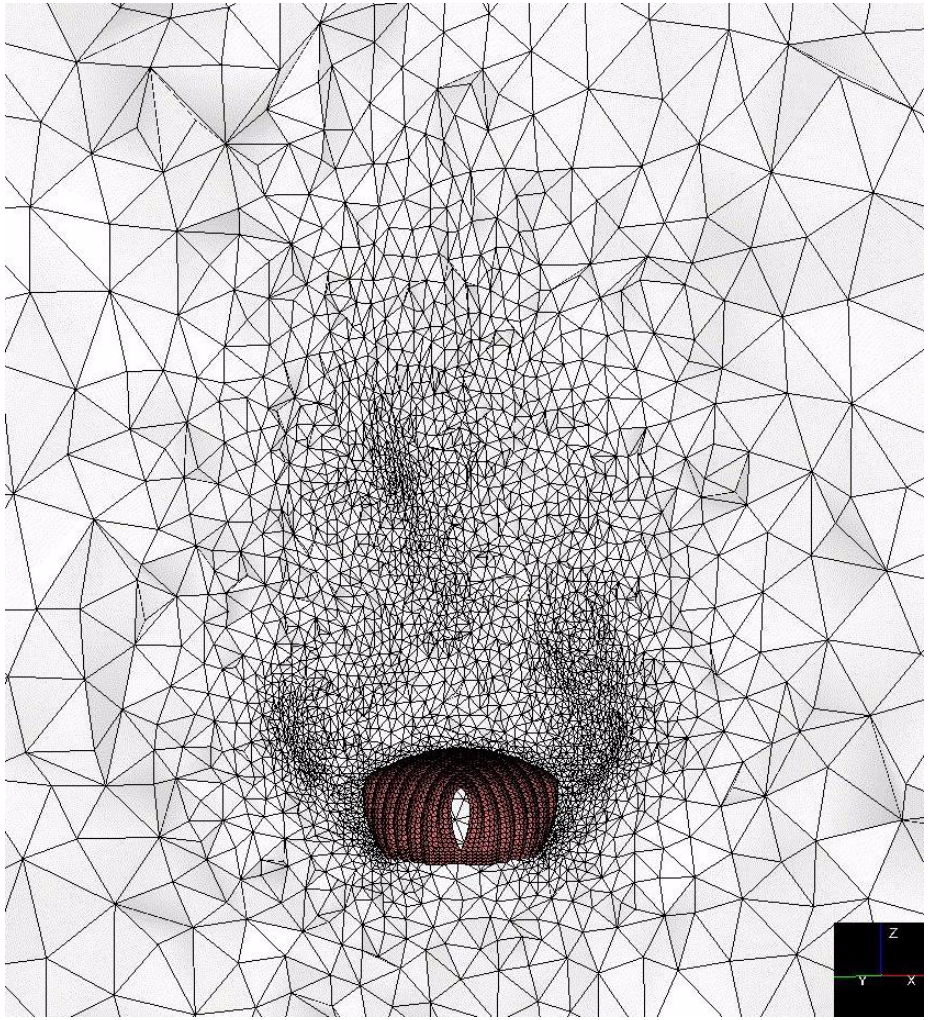


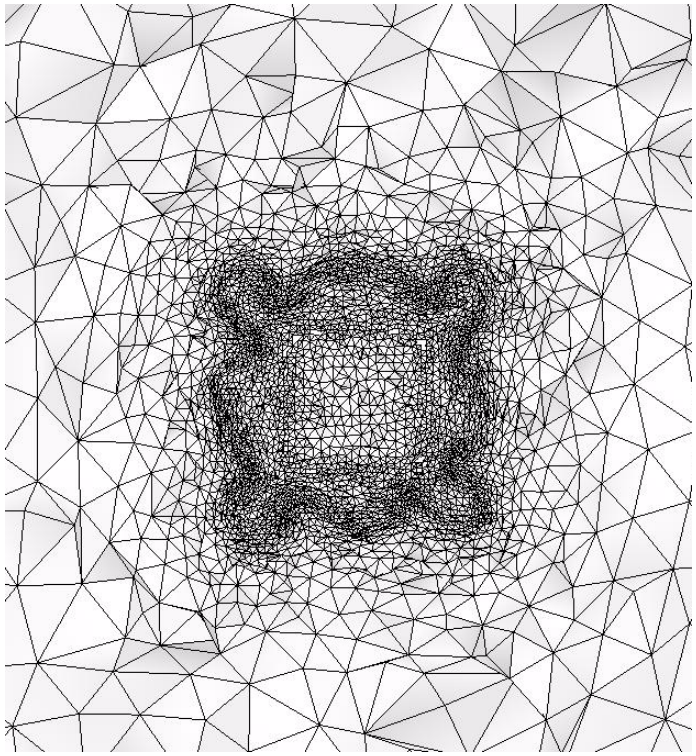
Figure 7.7 Initial mesh interacting with plane $x = y$ for the parachute problem.

Figure 7.7 visualizes the initial mesh by showing a slice of the 3D mesh interacting with the plane of $x = y$. It consists of 123,948 tetrahedra. To allow pressure discontinuity on the two sides of the parachute, the volume mesh has been split by the canopy surface such that two duplicated pieces of surface meshes exist, one is classified on the upper side of the canopy surface and another is classified on the lower side.

This unsteady solution has been obtained by solving the same time-dependent, incompressible Navier-Stokes equations given in Section 7.1. The initial mesh is used in the first 60 solution steps, then the mesh is adapted based on second order directive information every 30 steps. Figure 7.8 shows the mesh after the 8th mesh adaptation, which is used to solve the flow from step 270 to step 300. The total number of tetrahedra has reached 563,487. Figure 7.8(a) shows the side view of the evolving mesh (the slice of the 3D interior mesh faces interacting with plane $x = y$), while (b) shows the top view (the slice of interior mesh faces interacting with plane $z = 0.6$). Figure 7.9 gives the speed contour at solution step 300 on these two planes. It can be seen that mesh adaptation has created small elements near the parachute and in the downstream out of the cut-off, and refined the region on top of the parachute in a way that can catch unsteady flow around step 300. Also the result mesh is smoothly graded mesh in all directions. Figure 7.10 shows

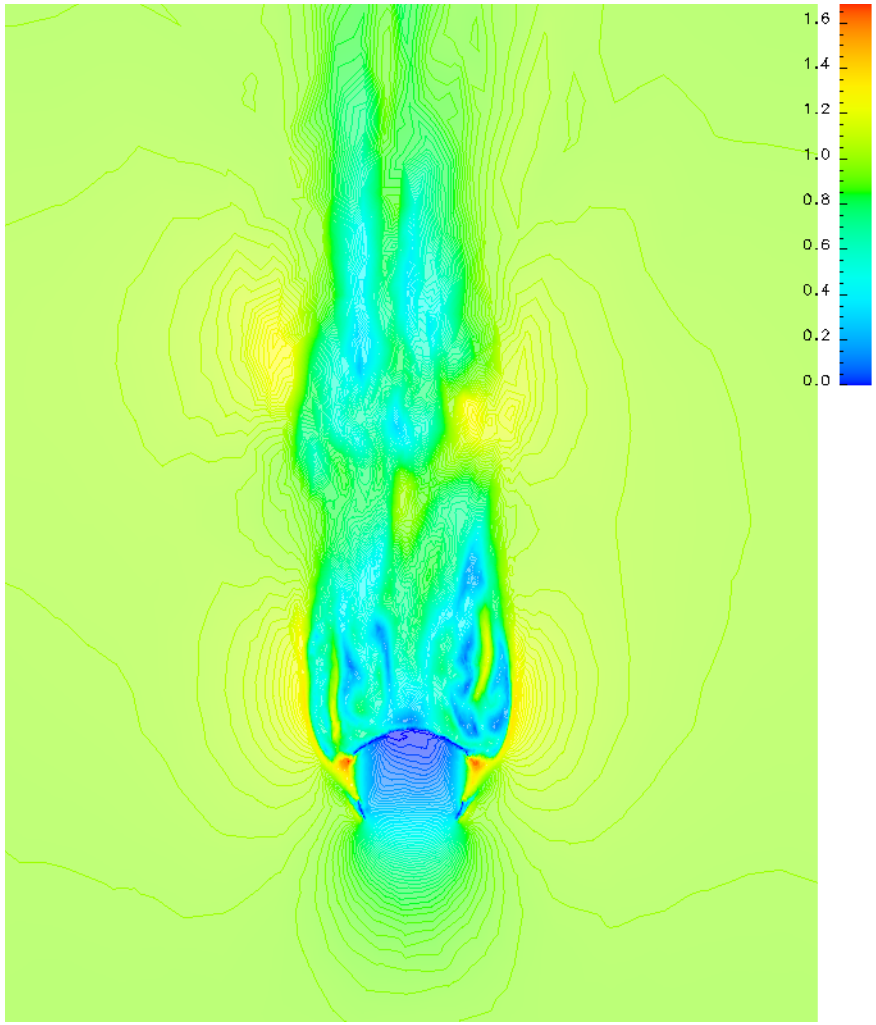


(a) side view: the parachute and interior mesh faces interacting with plane $x = y$

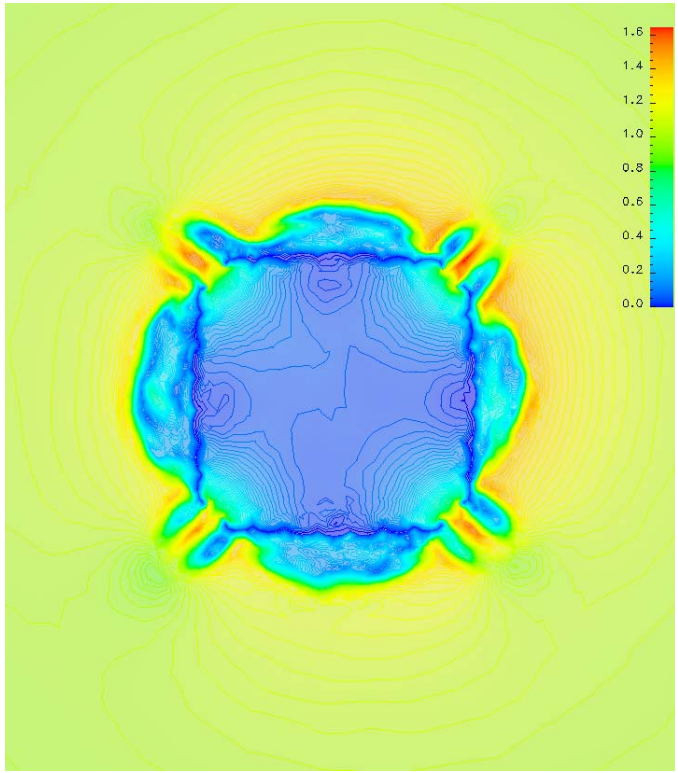


(b) top view: interior mesh faces interacting with plane $z=0.6$

Figure 7.8 Evolving mesh after the 8th mesh adaptation for parachute problem.



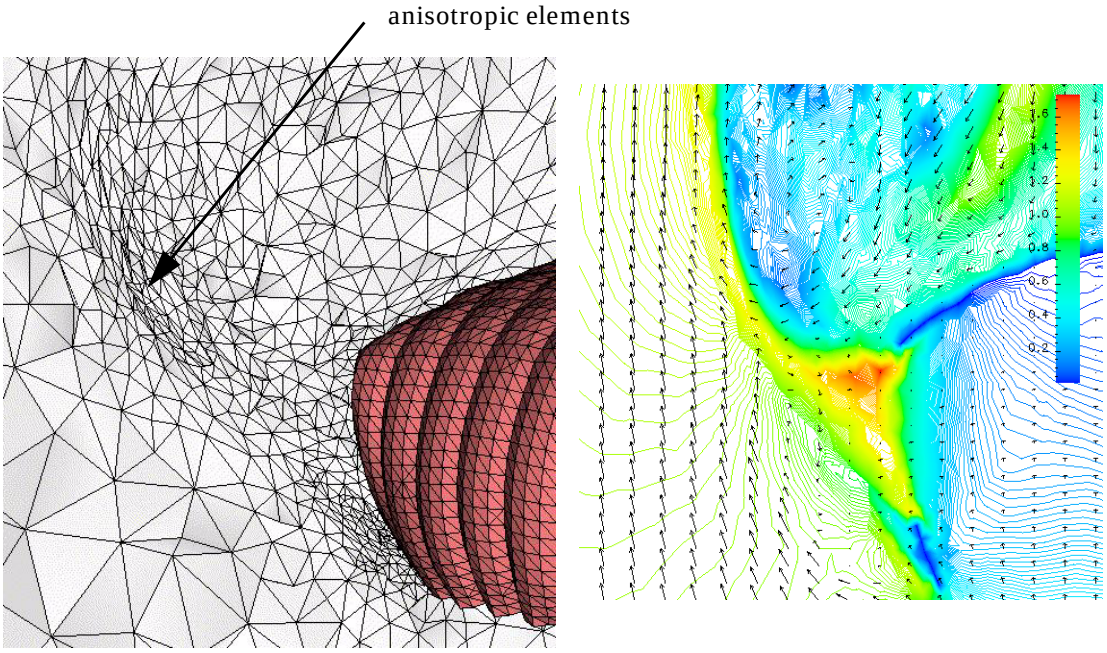
(a) side view: velocity contour on plane $x = y$



(b) top view: velocity contour on plane $z = 0.6$

Figure 7.9 Velocity contour at step 300 on two planes for parachute problem.

a closed-up view to the downstream mesh and velocity near one of the cut-off corner of the parachute. One can see anisotropic elements aligned to the flow.



(a) mesh (b) speed contour and vector field of velocity

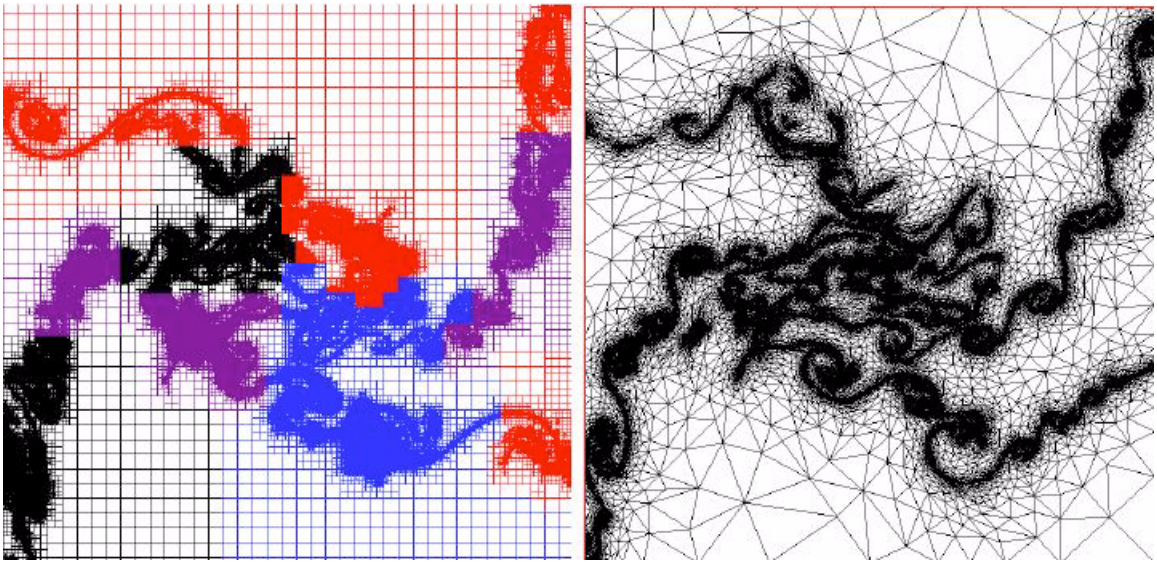
Figure 7.10 Zoom near one cut-off corner.

7.3 Four Contact Riemann Problem

Figure 7.11 gives two meshes adapted to the same smallest mesh dimension in solving a four contact Riemann problem [71]. The quadtree-like quadrilateral mesh in Figure 7.11(a) is the result of a non-conforming refinement procedure, while the result of conforming anisotropic refinement is given in Figure 7.11(b). The conforming anisotropic refinement only took 1/4 the time of the non-conforming procedure.

7.4 Cannon Blast Problem

The third example shows the controlled anisotropic elements capable of tracking expansion waves and their interactions in a 2D cannon blast problem. Consider the problem of a 2D perforated tube of diameter *155 mm* (a cannon) as shown in Figure 7.12(a). The diameter of the perforated holes inside the barrel (the muzzle break) is *d=28.6 mm*.



(a) non-conforming refinement of quadtree-like mesh

(b) conforming anisotropic refinement

Figure 7.11 Four contact Riemann problem.

The initial conditions for the problem are the one of a shock tube. Consider a virtual interface inside the barrel (see Figure 7.12(a)). The initial pressure for the gas in the left of the virtual interface and inside the tube are $P = 57{,}273{,}627.96\ Pa$, i.e. more than *500* times the external atmospheric pressure of $P_{atm} = 101{,}300\ Pa$, the initial temperature of the air inside the tube is $T = 2{,}111.5$ K, and its initial velocity along x direction is *0*.

The problem is governed by Euler's equation and solved using discontinuous Galerkin method. The final time of the computation was $t_{end} = 5e\text{-}4$. Starting time steps were about $5 \sim 10^{-8}$ seconds and the final time steps were about $1.5 \sim 10^{-8}$ seconds. The mesh was refined every 10^{-6} seconds so that the total number of mesh refinements is *501*, including the initial refinement that enables the correct capture of the initial discontinuous state (see Figure 7.12(a)). The total number of time steps was *45,438*. The total number of degrees of freedom in the initial mesh was *5,556* which corresponds to *463* triangles. After the *501* adaptations, the number of degrees of freedom reached *778,488* which corresponds to *64,874* triangles. Figure 7.12 shows the evolution of the mesh for the cannon blast problem. Figure 7.13 plots the density contours corresponding to the adapted meshes of Figure 7.12. One can clearly see that the density contours do not have any pre and post shock noise due to the alignment of anisotropic elements with shock waves, and the simultaneous development between anisotropic elements and the density contours. Figure 7.14

and Figure 7.15 gives two close-up views to further demonstrate the captured solution by the aligned anisotropic elements. In Figure 7.14, the complex shock-shock interactions happening above the perforated holes are captured by anisotropic elements distributed in the direction and position the density contours indicate. In Figure 7.15, the zoom near the front shock shows the alignment between the anisotropic elements and the front shock.

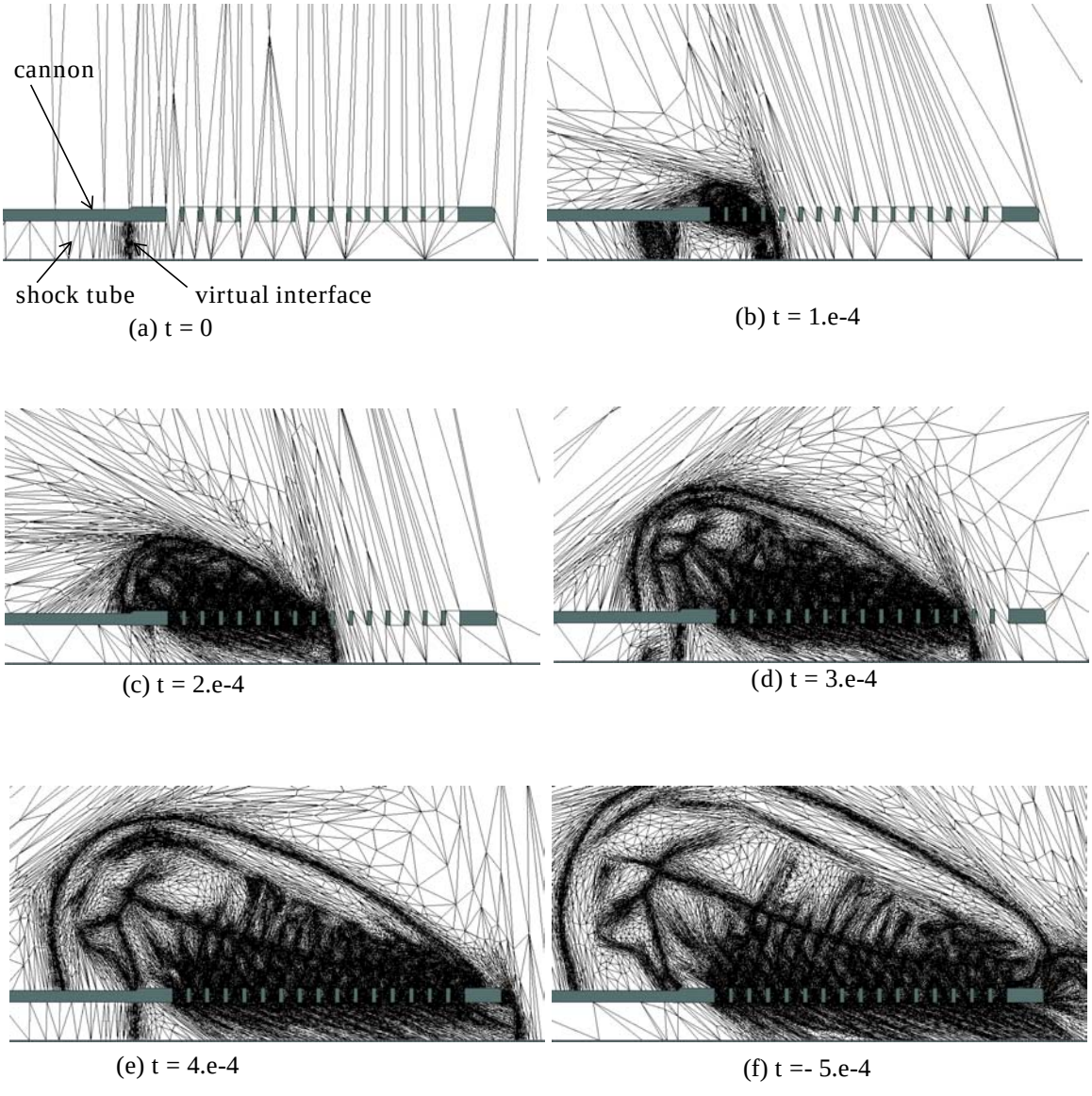


Figure 7.12 Evolution of the adapted meshes for the cannon problem.

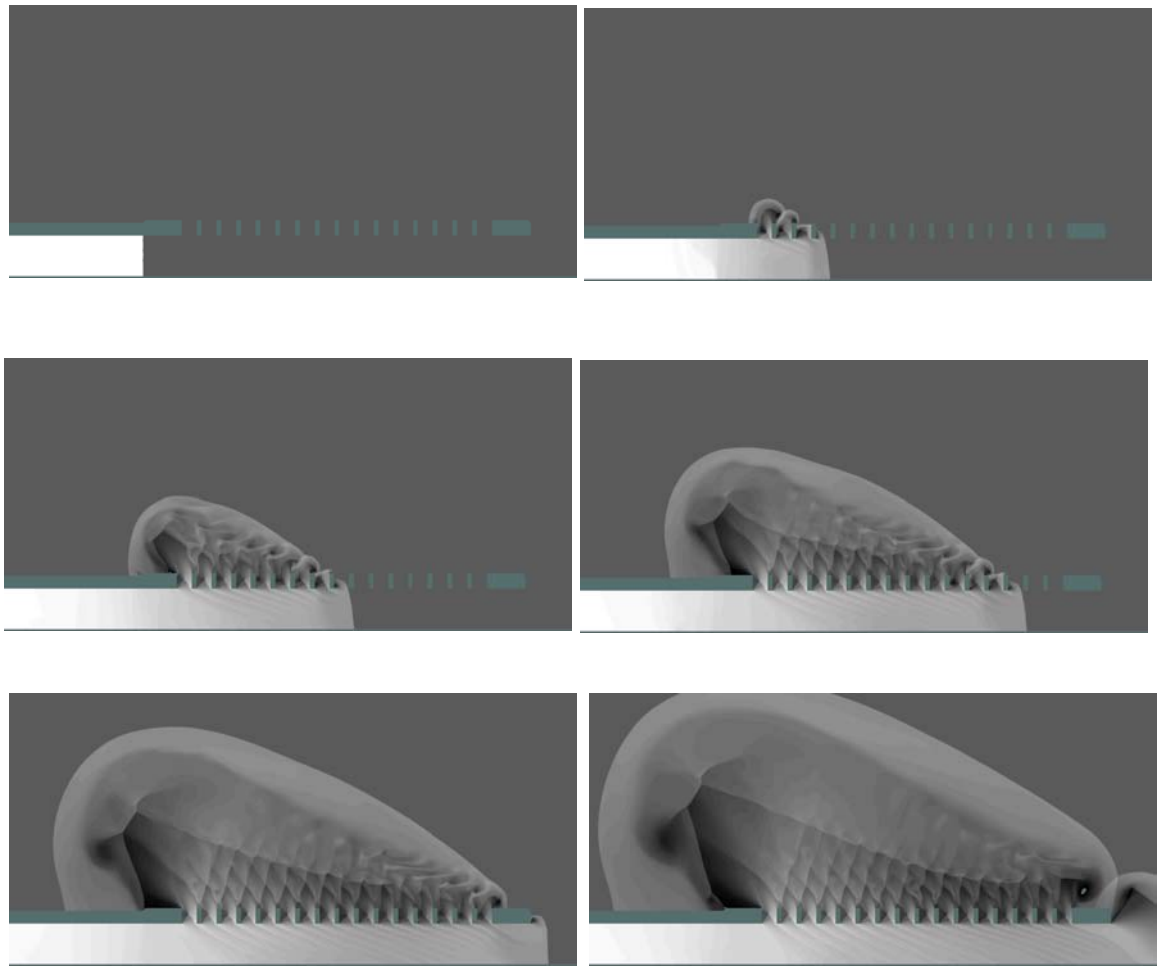


Figure 7.13 Evolution of density contours in log scale for the cannon problem.

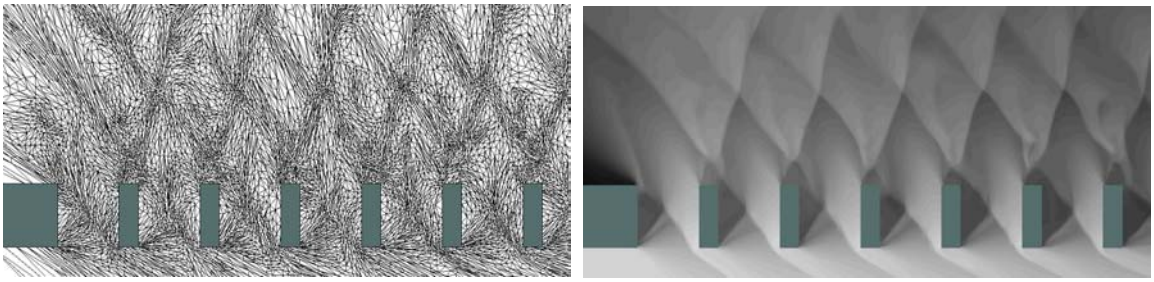


Figure 7.14 Complex shock-shock interaction structure near the muzzle.

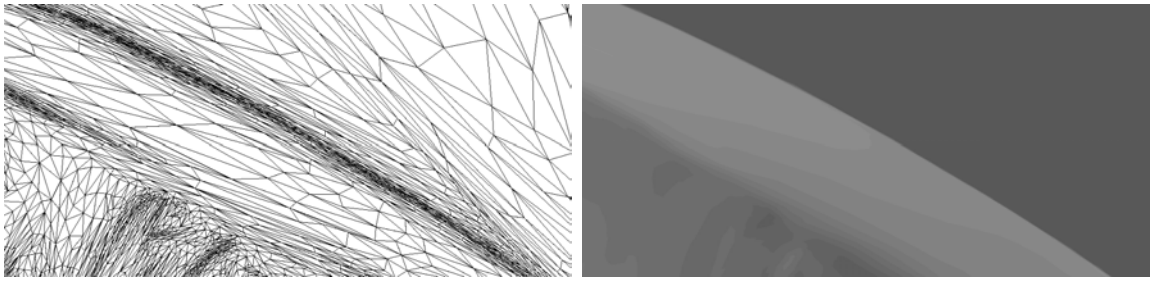


Figure 7.15 Zoom near the front shock at *t=5.e-4*.

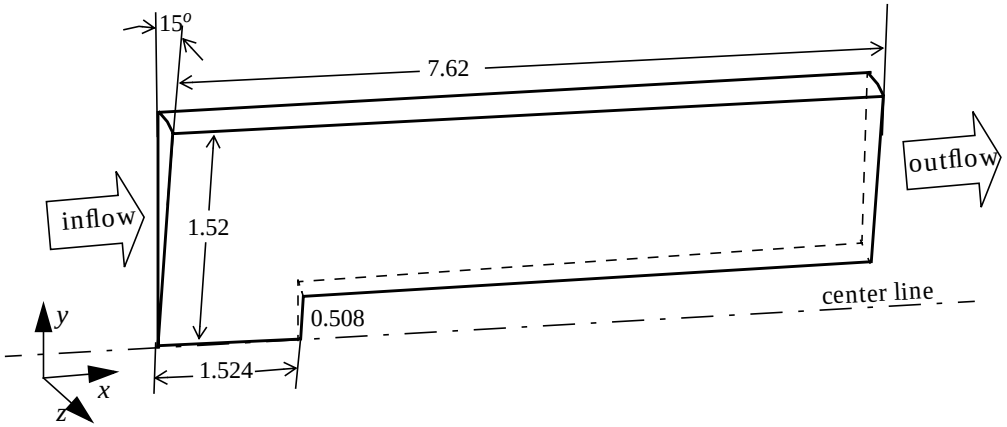


Figure 7.16 Schematic diagram of backward step.

7.5 Backward Step

This example demonstrates the controlled anisotropic tetrahedral elements in the simulation of shock development when a backward facing step is impulsively inserted into a Mach 3 gas flow in a straight pipe. Figure 7.16 shows the analysis domain. Since axisymmetric, only a small section (15 degree) of a cylinder is used. The cylinder is of length 7.62 and of radius 1.52, and the step is situated at x = 1.524 and of height 0.508. The initial condition is a constant Mach 3 flow field in the x-axis, in particular,

$$p = 1$$

$$\rho = 1$$

$$u_1 = M_s\sqrt{\Upsilon} = 3\sqrt{1.4} = 3.55$$

where p denotes pressure, ρ denotes density, u_1 is the velocity in x direction, M_s is Mach number and Υ is gas parameter. The boundary conditions are as follows:

- At inlet and outlet, the velocity, density and pressure are that of the initial Mach 3 flow;
- At the two cut planes parallel to the center line of the cylinder, symmetry boundary condition is applied;
- For all other surfaces, slip wall boundary condition is applied.

The problem is solved using discontinuous Galerkin method, starting from a uniform isotropic initial mesh of size 0.5. A steady flow pattern with shock is reached in about 4 seconds. The mesh is updated every 5×10^{-3} seconds, therefore, a total of 800 mesh adaptations are performed. The total number of degrees of freedom in the initial mesh is 14,960 which corresponds to 748 tetrahedra. After 800 mesh adaptations, the number of degrees of freedom reaches 96,020 which corresponds to 5081 tetrahedra. Figure 7.17 shows the evolution of the mesh for the backward step problem while Figure 7.18 shows the evolution of corresponding density contour surface. It can be seen that the mesh aligns to the discontinuity of density with anisotropic tetrahedra and develops as the discontinuity develops. Figure 7.19 shows a close-up view of the mesh and density contour near the top surface where the shock reflects. It can be seen that elements become needle-like where the shock strikes the top surface.

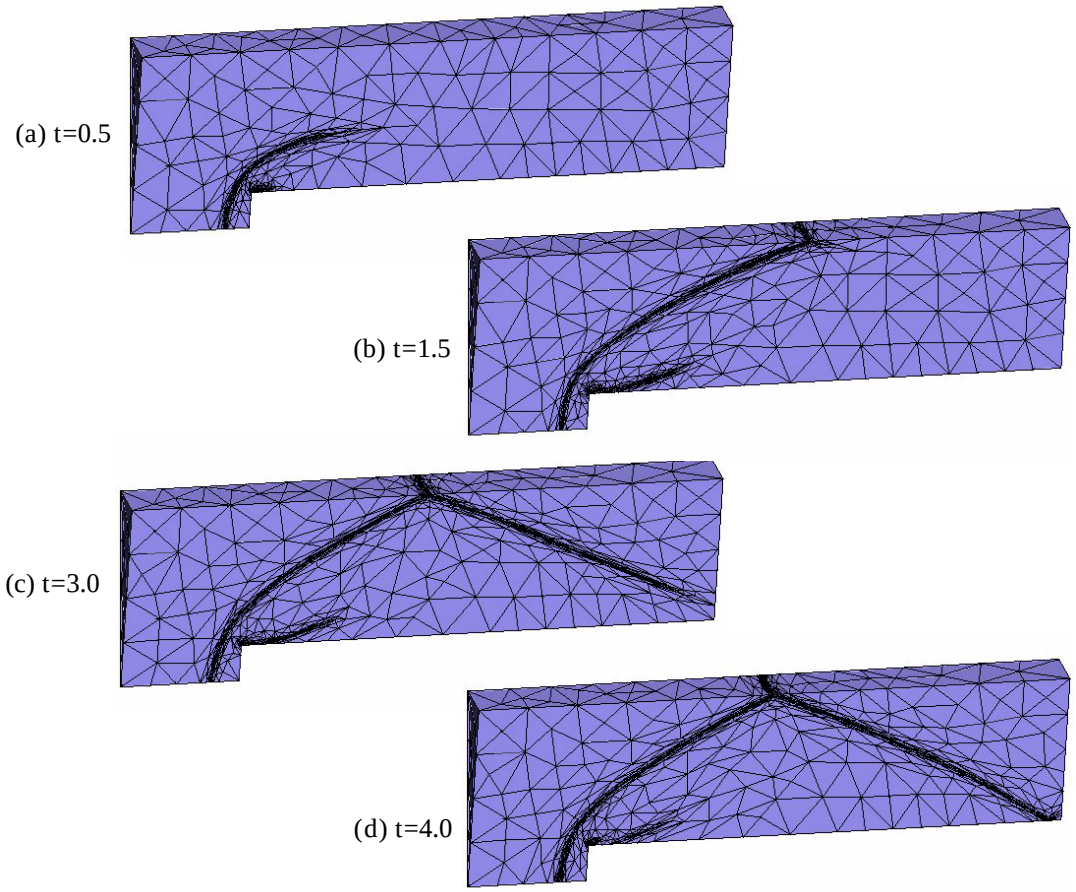


Figure 7.17 Evolution of adapted mesh for backward problem.

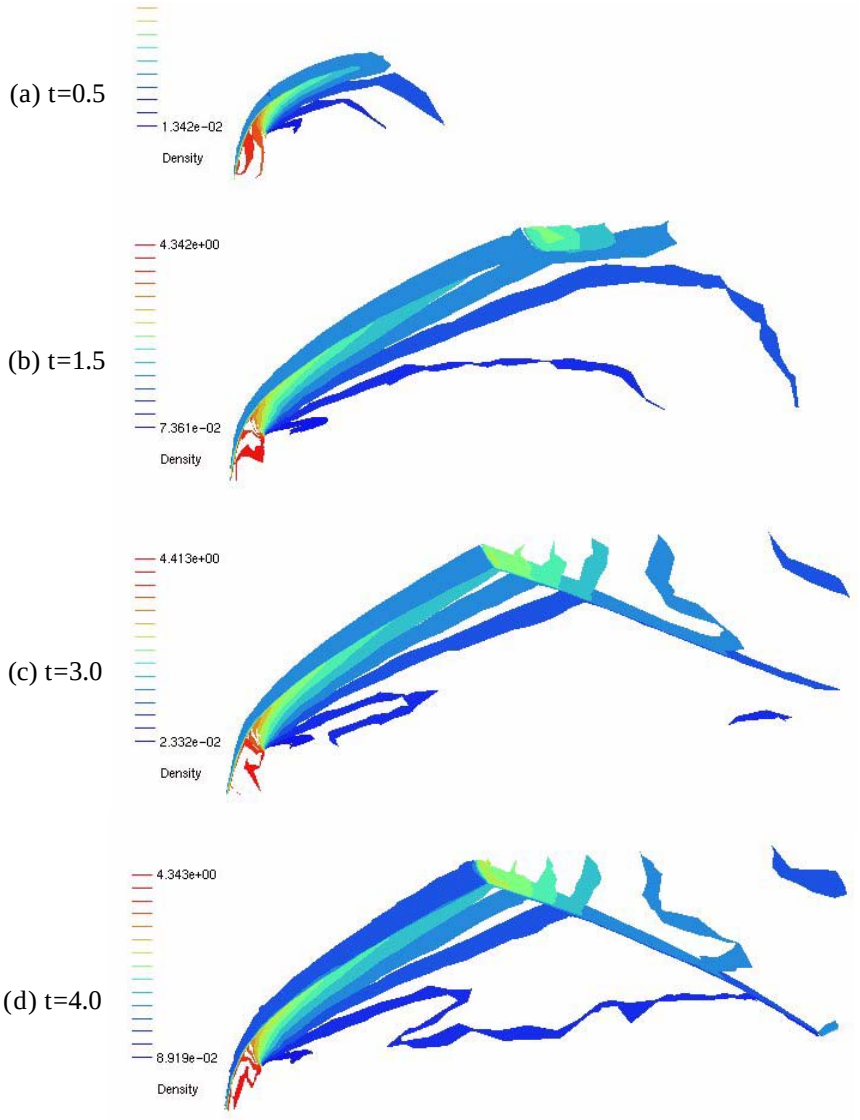
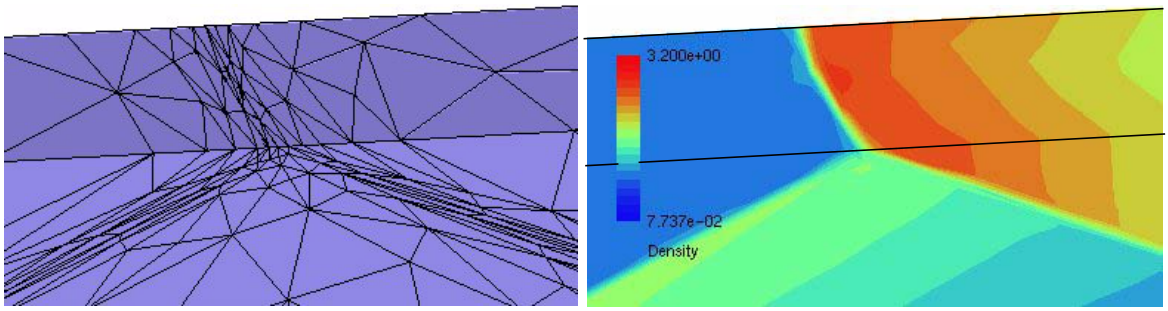


Figure 7.18 Evolution of density contour surfaces for backward step problem.



(a) mesh (b) density contour

Figure 7.19 Zoom near the shock reflection at t = 4 seconds.

CHAPTER 8

Conclusion

The aim of this thesis was to develop effective tools and algorithms for general anisotropic mesh adaptation on complex three dimensional domains. This chapter summarizes the contributions made in achieving this goal, and suggests direction for future work.

8.1 Research Contributions

8.1.1 Representation and Implementation of Anisotropic Mesh Size Field

An essential requirement for anisotropic mesh adaptation procedure is an appropriate representation of desired element size and shape distribution. Contributions made for this purpose are:

- A definition of mesh metric tensor field was given.
- Introduced a transformed space where the mesh metric field is isotropic and normalized; Length and volume computation in the transformed space were derived;
- Ideal conformity criteria between mesh metric field and the mesh was introduced and demonstrated in one dimension. Practical conformity criteria feasible for mesh modification procedures was presented.
- An abstract interface for mesh metric field was developed. Two specific mesh metric field definition options were implemented: analytical definition over geometry domain and a piecewise definition option using the current mesh.

8.1.2 Mesh Modification Operators

To support anisotropic mesh adaptation, three contributions at mesh modification operator level were made on the basis of existing mesh modification operators:

- For all mesh modification operators, supported the evaluation of maximal edge length and worst element quality of the result mesh configuration in transformed space;
- Enriched the existing mesh modification operators with four additional mesh modification operators;
- Defined a base local mesh modification object that wraps around all common data and functionalities of local mesh modification operators.

8.1.3 Mesh Modification Procedures

An effective anisotropic mesh modification procedure for three dimensional geometry domains capable of handling anisotropic elements has been developed and tested. This procedure consists of four related components: mesh refinement, vertex snapping, mesh coarsening and mesh re-alignment. Key contributions are:

- A vertex snapping procedure that addressed the geometry approximation issue.
- Generating and maintaining control of anisotropic elements by respecting mesh metric field.
- Effective application of local mesh modification in adaptive analysis.

An efficient incremental refinement algorithm governed by mesh metric field was developed. The algorithm can reduce maximal edge length in the transformed space to acceptable level incrementally by applying element subdivision templates in an iterative way. Several techniques were introduced to quickly align the mesh to the mesh metric field, including: (1) simultaneously split “a set of longest mesh edges” in transformed space; (2) collapse new short edges refinement creates; (3) create new vertices in the middle point of an edge in the transformed space; (4) in case ambiguous, always create the shortest diagonal in transformed space; and (5) use lookup tables.

A procedure of placing vertices onto curved geometry boundary was developed. It first applies local mesh modification to effectively snap as many vertices as possible. The key that improves efficiency and success rate is always ensuring that a vertex moves at least to its first problem plane or model boundary after the application of a mesh modification step. Since it is not desirable to attempt to develop a provably exhaustive set of local mesh modification operations, the snapping procedure includes a generalized cavity triangulation procedure that is applied to the small numbers of vertices that remain unsnapped after mesh modification. The cavity triangulation procedure ensures the vertex under consideration is dealt with. Although, the re-meshing procedure can introduce new vertices to be snapped, the procedures have properly placed these vertices on the boundary for all examples tested due to the fact that they are on a triangulation with improved local geometric approximation.

A mesh coarsening algorithm was presented with two key contributions: (1) effective determination of mesh modifications by using “collapsing shortest edge” as criteria. In terms of this criteria, only one edge collapsing evaluation is needed, and, more importantly, in case the edge collapsing fails, the next local mesh modification can be effectively determined by analyzing the available information computed in evaluating the edge collapsing; and (2) an algorithm that changes vertex processing order to follow “topological every other vertex” was developed to maintain good vertex distribution during coarsening.

The contribution in re-alignment algorithm is the sliver tetrahedron analysis technique and the effective selection of local mesh modifications based on the analysis result.

8.1.4 Adaptive Anisotropic Simulation for general 3-D Geometry

To apply the anisotropic mesh adaptation procedure for general adaptive simulations, additional technical issues addressed are:

- Adaptive mesh metric field specification using second derivative and gradient information.
- Mesh metric field smooth algorithm capable of controlling the anisotropic gradation of adaptive meshes.
- Local solution mapping in the execution of mesh modifications.

These algorithms were integrated into the first anisotropic adaptive procedure for 3-D general geometries, and have been applied to solve flow problems. Results of adaptivity were presented to demonstrate the effectiveness and the alignment of adapted meshes to both geometry model and mesh metric field.

8.2 Future Work

The research presented in this thesis can be extended in following aspects:

- More general and reliable anisotropic error estimation and mesh correction indication (defining mesh metric field) is critical to complete the anisotropic adaptivity procedure.
- Tightly integrate a parallel version with parallel solvers for application with large scale transient problems.
- Extend this work into *p*-version and *hp*-version adaptive analysis.
- Extend this work into *hr*-version adaptive analysis.
- Some near term extensions and possible enhancements are:

  1. Use principle directions of mesh metric tensor to improve alignment. Although Section 2.2.2 has indicated that it is desired to generate mesh edges aligning to the three principle directions of the metric tensor, this information was not fully used in this thesis. Several possibilities of using this information are worth investigating. For example, adjust the position of new vertices created in splitting.

2. Connectivity information[1] is another piece of information that provides the potential of producing better adapted meshes. It is useful to understand the relationship between the optimal connectivity and local mesh size distribution. For example, in a 2D construct, the optimal number of edges connected to an interior vertex is six. However, this number may increase or decrease depending on the relative mesh size surrounding the vertex.
3. Although a callback mechanism was developed to support the solution mapping in the execution of mesh modification, the adjacency interrogation related to the boundary mesh entities of the cavity associated with the mesh modification is invalid since, at this point, a valid local mesh topology is not constructed yet. This issue may be addressed by a bit modification to the mesh database.
4. Since effective, considered curved domains and capable of creating smooth mesh gradation, it may be desirable to extend the capabilities in this thesis for anisotropic mesh generation purpose.

1. Number of edges connected to a vertex, number of faces connected to an edge and etc.

REFERENCES

1. Ainsworth, M. and Oden, J. T., A Posteriori Error Estimation in Finite Element Analysis, John Wiley & Sons, Inc., 2000.

2. Almeida, R. C., Feijoo, R. A., Galeao, A. C., Padra, A. and Silva, R. S., “Adaptive finite element computational fluid dynamics using an anisotropic error estimator”, *Comput. Meth. Appl. Mech. Engrg.* v182, 379-400, 2000.

3. Arnold, D. N., Mukherjee, A. and Pouly L., “Locally adapted tetrahedral meshes using bisection”, *SIAM J. Sci. Comput.* v22, 431-448, 2000.

4. Babuska, I., and Szabo, B., “On the rates of convergence of the finite element method”, *Int. J. Num. Meth. Engrg.* v18, 323-341, 1982.

5. Baehmann, P. L. and Shephard, M. S., "Adaptive multiple-level h-refinement in automated finite element analysis", *Engineering wirh computers* v5, 235-248, 1989.

6. Bank, R. E. and Smith, R. K., "Mesh smoothing using a posteriori error estimators", *SIAM J. Numer. Anal.* v34, 979-997, 1997.

7. Bansch, E., “Refinement in 2 and 3 dimensions”, *Impact of Computers in Science and Engineering* v3, 181-191, 1991.

8. Beall, M. W. and Shephard, M. S., "A general topology-based mesh data structure", *Int. J. Num. Meth. Engrg.* v40, 1573-1596, 1997.

9. Bey, J., “Tetrahedral grid refinement”, *Computing* v55, 271-288, 1995.

10. Beyer, W. H., *CRC standard Mathematical Tables,* 28th ed., Boca Raton, FL: CRC Press, pp. 210-211,1987.

11. Biswas, R. and Strawn, R., "A new procedure for dynamic adaptation of three dimensional unstructured grids", In *31st Aerospace Science Meetin*g, 1993.

12. Bornemann, F., Erdmann, B. and Kornhuber, R., “Adaptive multilevel methods in three-dimension spaces”, *Int. J. Num. Meth. Engrg.* v36, 3187-3203, 1993.

13. Borouchaki, H. and Frey, P. J., "Adaptive triangular-quadrilateral mesh generation", *Int. J. Num. Meth. Engrg.* v41, 915-934, 1998.

14. Borouchaki, H., George, P. L., Hecht, F., Laug, P., and Saletl, E., “Delaunay mesh generation governed by metric specification: Part I. Algorithm”, *Finite Element in Analysis and Design* v25, 61-83, 1997.

15. Borouchaki, H., Hecht, F. and Frey, P. J., “Mesh Gradation control”, *Int. J. Num. Meth. Engrg.* v43, 1143-1165, 1998.

16. Bossen, F. J. and Heckbert, P. S., “A plaint method for anisotropic mesh generation”, *Proc. 5th Int. Meshing Roundtable*, Sandia Report 96-2301, Sandia Nat. Lab., Albuquerque, NM, pp. 63-76,1996.

17. Bowyer, A., "Computing dirchlet tessellations", *The Computer Journal* v24, 162-

166, 1981.

18. Brooks, A. N. and Hughes, T. J. R., "Streamline upwind / Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations", *Computer Methods in Applied Mechanics and Engineering* v32, 199-259, 1982.

19. Buscaglia, G. C. and Dari, E. A., “Anisotropic mesh optimization and its application in adaptivity”, *Int. J. Num. Meth. Engrg.* v40, 4119-4136, 1999.

20. Castro-Diaz, M. J., Hecht, F. and Mohammadi, B., “New progress in anisotropic grid Adaptation for inviscid and viscous flows simulations”, *Proc. 4th Int. Meshing Roundtable*, Sandia Nat. Lab., Albuquerque, NM, 1995.

21. Craig, A. W., Ainsworth, M., Zhu, J. Z. and Zienkiewicz, O. C., "h and h-p version error estimation and adaptive procedures from theory to practice", *Engineering with Computers* v5, 221-234, 1989.

22. de Cougny, H. L. and Shephard, M. S., “Surface meshing using vertex insertion", *Proc. 5th Int. Meshing Roundtable*, Sandia Nat. Lab., Albuquerque, NM, pp243-256, 1996.

23. de Cougny, H. L. and Shephard, M. S., “Parallel refinement and coarsening of tetrahedral meshes”, *Int. J. Num. Meth. Engrg.* v46, 1101-1125, 1999.

24. de Cougny, H. L., Shephard, M. S. and Georges, M. K., “Explicit node point mesh smoothing within the octree mesh generator”, *SCOREC report* 1990-10, Rensselaer Polytechnic Institute, Troy, NY, 12180.

25. de l’Isle, E. B. and George, P. L., “Optimization of tetrahedral meshes”, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, I. Babuska et al. Eds., IMA vol. 75, Springer, Berlin, pp. 97-128, 1993.

26. Dey, S., *Geometry-based three dimensional hp finite element modelling and computations*, PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 12180-3590, 1997.

27. Dompierre, J., Labbe, P., Guibault, F. and Camarero, R., "Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization", *Proc. 7th Int. Meshing Roundtable,* Dearborn, MI, 1998.

28. EDS Corp., "Parasolid V6 Functional Description", Electronic Data Systems Corporation, 13736 Riverport, Maryland Heights, MO 63043, 1994.

29. Freitag, L. A., Jones, M. T. and Plassmann, P. E., "An efficient parallel algorithm for mesh smoothing", *Proc. 4th Int. Meshing Roundtable,* Sandia Nat. Lab., Albuquerque, NM, pp.47-58, 1995.

30. Freitag, L. A. and Knupp, P. M., “Tetrahedral element shape optimization via the jacobian determinant and condition number”, *Proc. 8th Int. Meshing Roundtable*, South Lake Tahoe, CA, pp.247-258, 1999.

31. Frey, P. J. and George, P. L., “Mesh Generation”, *HERMES Science Europe Ltd.*,

2000.

32. Gago, J. P., De, S. R., Kelly, D. W., Zienkiewicz, O. C. and Babuska, I., “A posterior error analysis and adaptive processes in the finite element method. Part II: Adaptive mesh refinement”, *Int. J. Num. Meth. Engrg.* v19, 1621-1659, 1983.

33. Garimella, R. V. and Shephard, M. S., “Boundary layer mesh generation for viscous flow simulations”, *Int. J. Num. Meth. Engrg.* v49, 193-218, 2000.

34. George, P. L., Borouchaki, H. and Laug, P., “An efficient algorithm for 3D adaptive meshing”, *Finite Elements: Techniques and Developments*, B.H.V. Topping Eds.,Civil-Comp Press, Edinburgh, pp1-11, 2000.

35. George, P. L. and Hecht, F., “Nonisotropic grids”, *CRC Handbook of Grid Generation*, J. F. Thompson, B. K. Soni and N. P. Weatherill, Eds., CRC Press, Inc., Boca Raton, pp. 20.1-20.29, 1999.

36. Hughes, T. J. R., *The Finite Element Method*, Englewood Cliffs, NJ 07632.

37. Jansen, K. E., Shephard, M. S. and Beall, M. W., “On anisotropic mesh generation and quality control in complex flow problems”, *Proc. 10th Int. Meshing Roundtable*, Sandia Nat. Lab., Albuquerque, NM, pp.341-349, 2001.

38. Joe, B., "Three-dimensional triangulations from local transformation", SIAM J. Sci. Statist. Comput. v10, 718-741, 1989.

39. Joe, B., “Construction of three dimensional improved-quality triangulations using local transformations”, *SIAM J. Sci. Comput.* v16, 1292-1307, 1995.

40. Jones, M. T. and Plassmann, P. E., “Adaptive refinement of unstructured finite-element meshes”, *Finite Element in Analysis and Design* v40, 41-60, 1997.

41. Kallinderis, Y. and Vijayan P., "Adaptive refinement-coarsening scheme for three dimensional unstructured meshes", *AIAA Journal* v31(8), 1440-1447, 1993.

42. Kalro, V., Aliabadi, S., Garrard, W., Tezduyar, T., Mittal, S. and Stein, K., "Parallel finite element simulation of large ram-air parachutes", *Int. J. Num. Meth. Fluids* v24, 1353-1369, 1997.

43. Karamete, B. K., Beall, M. W. and Shephard, M. S., "Triangulation of arbitrary polyhedra to support automatic mesh generators", *Int. J. Num. Meth. Engrg.* v49, 167-191, 2000.

44. Khamayseh, A. and Kuprat A., “Anisotropic smoothing and solution adaptation for unstructured grids”, *Int. J. Num. Meth. Engrg.* v39, 3163-3174, 1996.

45. Kolman, B., *Elementary Linear Algebra*, The Macmillian Company, 1970.

46. Kunert, G., "Towards anisotropic mesh construction and error estimation in finite element method", *Numerical Meth. in Partial Differential Equations* v18, 625-648, 2002.

47. Labbe, P., Dompierre, J., Vallet, M-G, Guibault, F. and Trepanier, J-Y, “A measure

of the conformity of a mesh to an anisotropic metric", *Proc. 10th Int. Meshing Roundtable*, Sandia Nat. Lab., Albuquerque, NM, pp.319-326, 2001.

48. Lee, C. K., "Automatic adaptive mesh generation using metric advancing front approach", *Engineering Computations* v16, 230-263, 1999.

49. Li, L. Y., Bettess, P., Bull, J.W., Bond, T. and Applegath I., "Theoretical formulations for adaptive finite element computations", *Communications in Numerical Methods in Engineering* v11, 911-915, 1995.

50. Li, X., Shephard, M. S. and Beall M. W., "Accounting for curved domains in mesh adaptation", *Int. J. Num. Meth. Engrg.* v58, 247-276, 2003.

51. Li, X., Shephard, M. S. and Beall M. W., "3D anisotropic mesh adaptation by mesh modifications", in preparation for submitting to *Computer Methods in Applied Mechanics and Engineering.*

52. Liu, A. and Joe, B., "Relationship between tetrahedron shape measures", *BIT* v34, 268-287, 1994.

53. Liu, A. and Joe, B., "On the shape of tetrahedra from bisection", *Mathematics of Computation* v63, 141-154, 1994.

54. Liu, A. and Joe, B., "Quality local refinement of tetrahedral meshes based on bisection", *SIAM Journal on Scientific Computing* v16, 1269-1291, 1995;

55. Liu, A. and Joe, B., "Quality local refinement of tetrahedral meshes based on 8-subtetrahedron subdivision", *Mathematics of Computation* v65, 1183-200, 1996.

56. Lo, S. H., "3D anisotropic mesh refinement in compliance with a general metric specification", *Finite Elements in Analysis and Design* v38, 3-19, 2001.

57. Lohner, R., "Adaptive remeshing for transient problems", *Computer Methods in Applied Mechanics and Engineering* v75, 195-214, 1989.

58. Lohner, R., "Extensions and improvements of the advancing front grid generation technique", *Communications in Numerical Methods in Engineering* v12, 683-702, 1996.

59. Mäntylä, M., *Introduction to solid modeling*, Computer Science Press, Rockville, Maryland, 1988.

60. Maubach, J. M., "Local bisection refinement for N-simplicial grids generated by reflection", *SIAM J. Sci. Comput.* v16, 210-227, 1995.

61. Mavripilis, D. J., "Adaptive mesh generation for viscous flows using delaunay triangulating", *Journal of Computational Physics* v90, 271-291, 1990.

62. Moller, P and Hansbo, P, "On advancing front mesh generation in three dimension", *Int. J. Num. Meth. Engrg.* v38, 3551-3569, 1995.

63. Onate, E. and Bugeda, G. A., "Study of mesh optimality criteria in adaptive finite element analysis", *Engineering computations* v10, 307-320, 1993.

64. Owen, S. J. and Saigal, S., "Surface mesh sizing control", *Int. J. Num. Meth. Engrg.* v47, 497-511, 2000.

65. Pain, C. C., Umpleby, A. P., de Oliveira, C. R. E. and Goddard, A. J. H., “Tetrahedral mesh optimization and adaptivity for steady-state and transient finite element calculations”, *Int. J. Num. Meth. Engrg.* v190, 3771-3796, 2001.

66. Peraire, J., Peiro, J. and Morgan, K., “Adaptive remeshing for three-dimensional compressible flow computation”, *J. Comput. Phys.* v103, 269-285, 1992.

67. Peraire, J., Peiro, J., Morgan, K. and O. C. Zienkiewicz, “Finite Element Euler Computations in Three Dimensions”, *Int. J. Num. Meth. Engrg.* v26, 2135-2159, 1988.

68. PTC, "Pro/TOOLKIT Reference Manual", Parametric Technologies Corp., 128 Technology Drive, Waltham, MA 02154, 1997.

69. Remacle, J.-F., Flaherty, J. E. and Shephard, M. S., "An adaptive discontinuous Galerkin technique with an orthogonal basis applied to compressible flow problems", *SIAM Review* v45, 53-72, 2003.

70. Remacle, J.-F., Li, X., Chevaugeon N. and Shephard M. S., “Transient mesh adaptation using conforming and non conforming mesh modifications”, *Proc. 11th Int. Meshing Roundtable*, Sandia Nat. Lab., Albuquerque, NM, 2002.

71. Remacle, J.-F., Li, X., Shephard M. S. and Flaherty, J. E., “Anisotropic Adaptive Simulation of Transient Flows using Discontinuous Galerkin Methods”, *in preparation for submitting to int. J. Num. Meth. Engrg.*

72. Rippa, S, “Long and thin triangles can be good for linear interpolation”, *SIAM J. Numer. Anal.* v29, 257-270, 1992.

73. Rank, E. and Babuska, I., “An expert system for the optimal mesh design in the hp-Version of the finite element method”, *Int. J. Num. Meth. Engrg.* v24, 2087-2106.

74. Rivara, M.C., "Selective refinement/derefinement algorithms for sequences of nested triangulaions", *Int. J. Num. Meth. Engrg.* v28, 2889-2906, 1989.

75. Rivara, M. C., “A 3-D refinement algorithm suitable for adaptive and multi-grid techniques”, *Communications in Applied Numerical Methods* v8, 281-290, 1992.

76. Rivara, M. C., "New Longest-edge algorithms for the refinement and/or improvement of unstructured triangulations", *Int. J. Num. Meth. Engrg.* v40, 3313-3324, 1997.

77. Schroeder, W. J, *Geoemtric triangulations with application to fully automatic 3-D mesh generation*, PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, 12180-3590, 1991.

78. Sheldon Axler, *Linear Algebra Done Right*, second edition, Springer, 1997.

79. Shephard, M. S., “Meshing environment for geometry-based analysis”, *Int. J. Num. Meth. Engrg.* v47, 169-190, 2000.

80. Shephard, M. S. and Georges, M., “Automatic three dimensional mesh generation by finite octree technique”, *Int. J. Num. Meth. Engrg.* v32, 709-749,1991.

81. Shephard, M.S. and Georges, M., "Reliability of automatic 3-D mesh generation", *comp. Meth. Appl. Mech. and Engng.* v101, 443-462, 1992.

82. Spatial Technology Inc., "ACIS Interface Guide and ACIS API Guide", Spatial Technology, Inc., 2425 55th Street, Building A, Boulder CO, 80301-5704, 1990.

83. Speares, W. and Berzins, M., "A 3D unstructured mesh adaptation algorithm for time dependent shock-dominated problems", *Int. J. Num. Meth. Fluids* v25, 81-104, 1997.

84. Stein, K., Benney, R., Kalro, V., Tezduyar, T. E., Leonard, J. and Accorsi, M., "Parachute fluid-structure interactions: 3-D computation", *Comput. Methods Appl. Mech. Engrg.* v190, 373-386, 2000.

85. Stroustrup, B., *the C++ Programming Language*, third edition, Addison-Wesley, 1998.

86. Szabo, B. and Babuska, I., *Finite Element Analysis*, Wiley Interscience, New York, 1991.

87. Toussaint, G. T., Verbrugge, C., Wang, C. and Zhu, B., "Tetrahedralization of simple and non-simple polyhedra", *Proc. 5th canadian conference on computational geometry*, university of waterloo, Canada, pp.24-29, 1993.

88. Traxler, C. T., "An algorithm for adaptive mesh refinement in n dimensions", *Computing* v59, 115-137, 1997.

89. Weiler, K. J., "The radial-edge structure: a topological representation for non-manifold geometric boundary representations", In Wozney, M. J., McLaughlin, H. W. and Encarnacao, J. L., editors. *Geometric Modeling for CAD applications*, North Holland, 1988,

90. Whiting, C. H. and Jansen, K. E., "A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis", *Int. J. Num. Meth. Fluids.* v35, 93-116, 2001.

91. Yamakaw, S. and Shimada, K., “High quality anisotropic mesh generation via ellipsoidal bubble packing”, *Proc. 9th Int. Meshing Roundtable*, Sandia Report 2000-2207, Sandia Nat. Lab., Albuquerque, NM, pp. 263-273, 2000.

92. Zhang, Z., "Successive subdivisions of tetrahedra and multigrid methods on tetrahedral meshes", *Houston J. Math.* v21, 541-556, 1995.

93. Zienkiewicz, O. C. and Zhu, J. Z., “A simple error estimator and adaptive procedure for practical engineering analysis”, Int. J. Num. Meth. Engrg. v24, 337-357, 1987.

94. Zienkiewicz, O. C. and Zhu, J. Z., “The superconvergent patch recovery and a posterior error estimator. Part 2. Error estimators and adaptivity”, *Int. J. Num.*

Meth. Engrg. v33, 1365-1382, 1992.

APPENDIX A

Adaptive Mesh Metric Specification Using Second Derivatives

For linear elements, error can be indicated by the second order derivatives of a solution variable. This appendix describes a method to define a mesh metric field in terms of a given field of second order derivatives (Hessian matrix field).

Consider a Hessian matrix field of solution variable u defined over a set of points. Let $\boldsymbol{H}_i(u)$ be the Hessian matrix at the i^{th} point ($i=1,n$) with n be the number of points. The corresponding mesh metric at the i^{th} point can be constructed by decomposing and scaling $\boldsymbol{H}_i(u)$, particularly:

$$T_i(u) = \begin{bmatrix} \boldsymbol{e}_1^T & \boldsymbol{e}_2^T & \boldsymbol{e}_3^T \end{bmatrix} \begin{bmatrix} \lambda'_1 & 0 & 0 \\ 0 & \lambda'_2 & 0 \\ 0 & 0 & \lambda'_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_1 \\ \boldsymbol{e}_2 \\ \boldsymbol{e}_3 \end{bmatrix}$$

with

$$\lambda'_j = max\left(k_i|\lambda_j|, \frac{1}{h_{max}^2} \right) \qquad j = 1, 2, 3$$

where $|\lambda_j|$ is the absolute value of j^{th} eigenvalue of Hessian matrix $\boldsymbol{H}_i(u)$, $\boldsymbol{e}_j$ is the j^{th} unit eigenvector of $\boldsymbol{H}_i(u)$, h_{max} is the maximal allowable mesh edge length in the mesh (since $\boldsymbol{H}_i(u)$ can be singular, it is needed to apply h_{max} in case λ_j is zero or close to zero) and k_i is a factor to be determined for each $\boldsymbol{H}_i(u)$ ($i=1,n$) to scale the mesh metric $\boldsymbol{T}_i$ to reasonable mesh edge length.

Additional error indicator / estimator information and mesh size information of the previous step are required to determine a reasonable k_i for each $\boldsymbol{H}_i(u)$ ($i=1,n$). One way of determining k_i is to use an indicator, I_i, to the leading interpolation error term defined below, and apply a constant I^{new} to all points over the domain to enforce a constant distribution of I_i.

$$I_i = h_i^2 \cdot |\lambda|_{max} \qquad \text{i=1,n}$$

where $|\lambda|_{max} = max(|\lambda_0|, |\lambda_1|, |\lambda_2|)$ with λ_j denote the j^{th} eigenvalue of $\boldsymbol{H}_i(u)$, and h_i is the desired mesh edge length in the direction associated with eigenvalue $|\lambda|_{max}$ to be obtained at the i^{th} point, which can be used to determine k_i.

Figure A.1 illustrates the method of determining constant I^{new} in a 1D construct. The black bullets on x-axis indicate a set of points in the mesh domain where second order derivatives are given. In terms of the mesh size of the previous mesh and the given second derivatives, a distribution of error indicator I_i (i=1,n) can be obtained for the previous step. The poly-segments and circles in Figure 1 indicate a possible distribution for the 1D case. The mean value of error indicator

in the previous step, I_{mean}, can then be computed as indicated by solid horizon line. The reasonable constant I^{new} (indicated by the dashed line) to achieve in the new mesh metric definition can be selected as a faction of I_{mean}, i.e.

$$I^{new} = \gamma \cdot I_{mean} = \frac{\gamma}{n} \sum_{i=1}^{n} I_i$$

where γ is the reduction rate of the interpolation error indicator and n is the number of points where Hessian matrices are defined.

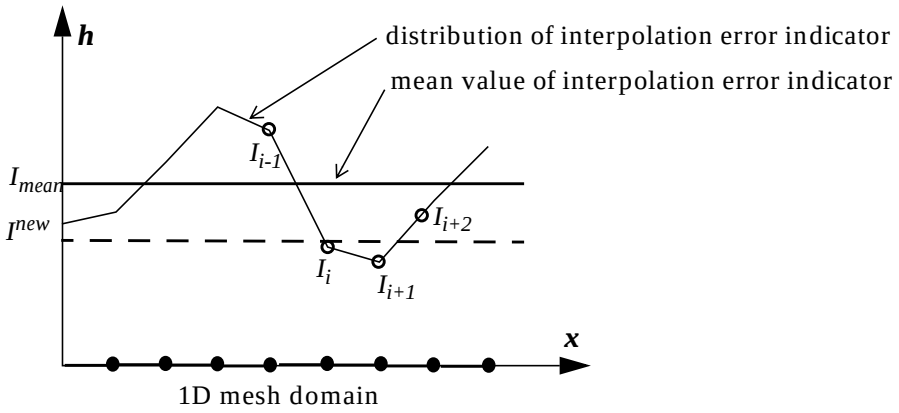


Figure A.1 Determination of I^{new} using information of previous step.