AGH University of Science and Technology
Kraków, Poland

Faculty of Electrical Engineering, Automatics,
Computer Science and Electronics

Department of Computer Science

Doctoral Thesis

Efficient Algorithms of Automatic Discretization of Non-Trivial Three-Dimensional Geometries and its Object-Oriented Implementation

by

mgr inż. Tomasz Jurczyk

Supervisor: dr hab. inż. Krzysztof Boryczko

Kraków, 2007

Akademia Górniczo-Hutnicza
Kraków, Polska

Wydział Elektrotechniki, Automatyki,
Informatyki i Elektroniki

Katedra Informatyki

Rozprawa Doktorska

Efektywne algorytmy automatycznej dyskretyzacji nietrywialnych trójwymiarowych form geometrycznych oraz ich obiektowa implementacja

mgr inż. Tomasz Jurczyk

Promotor: dr hab. inż. Krzysztof Boryczko

Kraków, 2007

Abstract

The task of the mesh generator is partitioning of the given geometric domain into a finite number of simple elements, conforming to requirements specified by user. Depending on the application of meshes, the requirements may concern type of elements, their size, stretching and quality. The meshing process itself has to be robust (i.e. applicable to wide range of geometric models), efficient and scalable, in order to cope with ever increasing complexity of the discretized geometric models.

The main contribution of this thesis is the development and optimization of an automated generator of unstructured anisotropic surface and volume meshes. The triangular meshes are created on parametric surfaces using a modified technique of incremental Delaunay triangulation. The anisotropic characteristic of mesh elements was obtained using non-Euclidean metric via the developed system of coordinate transformation. The quadrilateral meshes can be constructed with a systematic conversion of triangular meshes. For three-dimensional problems, tetrahedral meshes can be generated, also using the Delaunay technique. The proposed improvements allow to increase the efficiency of discretization and enhance the procedure of boundary constraining, which is one of the main problem of 3D Delaunay triangulation.

Generation of meshes (both two- and three-dimensional) is realized using a special control space structure, responsible for delivery of a desired size and shape of finite element in any point of the discretized domain. There is described the proposed method of storing this information through a metric transformation tensor together with a set of associated operations like interpolation, intersection or comparison. There is also presented an automated schema of discrete control space creation, taking into account a number of metric sources gathered automatically (e.g. from model geometry or numerical adaptation) or obtained from the user. The proposed set of control parameters allows an easy adjustment of the general characteristic of the obtained mesh (e.g. curvature approximation or maximum anisotropy or gradation ratio).

There are also presented numerous examples and experimental analysis of the computational and memory complexity of the subsequent phases of mesh generation and transformation.

Streszczenie

Zadaniem generatora siatek jest podzielenie zadanego obszaru geometrycznego na skończoną liczbę prostych elementów, spełniających określone przez użytkownika wymagania. W zależności od zastosowania siatek, wymagania mogą dotyczyć zapewnienia odpowiedniego typu elementów, ich rozmiaru, wydłużenia oraz jakości. Oprócz wymagań stawianych tworzonym siatkom równie istotne jest zagwarantowanie odpowiedniej jakości samego procesu dyskretyzacji. Wraz z rosnącą złożonością i szczegółowością modelowanych obszarów kluczowa staje się także skalowalność procesu dyskretyzacji, umożliwiająca tworzenie siatek o bardzo dużej liczbie elementów.

Celem pracy jest implementacja i optymalizacja automatycznego generatora niestrukturalnych, anizotropowych siatek powierzchniowych oraz objętościowych. Siatki trójkątne tworzone są na powierzchniach parametrycznych z wykorzystaniem zmodyfikowanej techniki inkrementacyjnej triangulacji Delaunay'a. Anizotropowy charakter elementów siatki został uzyskany dzięki wprowadzeniu nieeuklidesowej metryki poprzez opracowany system transformacji współrzędnych. Siatki czworokątne mogą być tworzone poprzez systematyczną konwersję siatek trójkątnych. Dla problemów trójwymiarowych tworzone są siatki czworościenne, także z wykorzystaniem technik związanych z triangulacją Delaunay'a. Zaproponowane ulepszenia pozwalają zwiększyć efektywność dyskretyzacji i usprawnić procedurę odzyskiwania brzegu, stanowiącą jeden z głównych problemów tej techniki.

Generowanie siatek (zarówno dwu- jak i trójwymiarowych) odbywa się w oparciu o specjalną strukturę przestrzeni kontrolnej, wskazującej pożądany rozmiar i kształt elementów w dowolnym punkcie dyskretyzowanej przestrzeni. W pracy opisany jest zaproponowany sposób przechowywania tej informacji poprzez metryczny tensor transformacji, wraz z zestawem powiązanych operacji takich jak interpolacja, przecięcie czy porównanie. Przedstawiony jest także schemat automatycznego stworzenia dyskretnej struktury przestrzeni kontrolnej, uwzględniającej szereg informacji o rozmiarze elementów pozyskanych automatycznie (np. z geometrycznego opisu modelu lub numerycznej adaptacji), jak również zadanych bezpośrednio przez użytkownika. Zaproponowany zestaw parametrów sterujących pozwala na łatwą kontrolę charakterystyki siatki (np. dopasowania do krzywizny lub maksymalnego stopnia anizotropii lub gradacji rozmiaru elementów).

Uzyskane wyniki poparte są także licznymi przykładami i eksperymentalną analizą złożoności obliczeniowej i pamięciowej poszczególnych etapów tworzenia i transformacji siatki.

Acknowledgment

The great thanks belong to Dr. Barbara Głut for her encouragement in preparation of this thesis, her leadership and guidance. Without her help this work would not have been done.

I would like to thank my advisor, Prof. Krzysztof Boryczko, for his support throughout my doctoral research work. I am grateful to Prof. Jacek Kitowski for his invaluable advice. Thanks belong also to my family, friends and colleagues for their help and support.

This work was partially supported by the Polish Ministry of Science and Higher Education (MNiSzW) Grant No. 3T11F00828.

Contents

Chapter 1

Introduction

The mesh can be defined as a partitioning of some geometric domain into a finite number of simple polyhedral or polygonal pieces, called elements. The elements have to fully cover the discretization domain, they may not be empty or overlapping, and they can intersect only through vertices, edges or faces (in three-dimensions). The mesh generation is a critical step in a wide range of engineering tasks, such as the study of scientific simulation by finite element method (FEM), design automation with geometrical modeling, and presentation of data with graphics and visualization.

The robustness, efficiency and quality are among the most important properties of each procedure of mesh generation. The efficiency is usually understood as a low time and memory requirements, both for practical applications and asymptotically. The robustness means the possibility of discretization of general geometries and coping with degenerated cases. The quality may depend on the application of the mesh, but is usually evaluated basing on a good distribution of elements with proper size and geometric shape.

1.1 Background

A detailed survey of the worldwide research concerning the mesh generation problems can be found on sites maintained by Owen[1] and Schneiders[2]. Mesh generation surveys can be also found in [3–5].

The finite element meshes can be classified into two main categories: structured and unstructured. A structured mesh is characterized by regular topology (i.e. same number of adjacent elements) of all inner vertices of the mesh. The structured meshes (also called grids) allow to simplify the simulation code and may be required in fields like CFD (computational fluid dynamics), where strict alignment of elements is necessary for accurate capturing of physical phenomena. For non-trivial boundaries the block-structured or mixed techniques can be used to divide the domain into topological blocks. An overview of structured mesh generation can be found e.g. in [6].

Triangular/tetrahedral meshes For unstructured meshes no such requirement is imposed, which allows creation of highly graded meshes, usually triangular or tetrahedral (other types of elements can be also used). In recent years, a number of algorithms for construction of such meshes have been developed. Quadtree/octree decomposition, Delaunay triangulation and advancing front technique are among the most common meshing algorithms:

- The quadtree/octree methods, introduced by Shephard's group [7–9], use a recursive decomposition of a box containing the discretized domain. The created rectangular cells are split into regular inner elements and irregular outer elements, using a set of predefined templates. The method is fast and efficient, but it has problems with the poor quality of elements near the boundary. A number of variations have been proposed including e.g. sizing or anisotropy[10, 11], red-green refinement [12, 13] or combination of quadtree/octree method with other (usually Delaunay) techniques[14, 15]. The quadtree/octree technique is often applied to visualization and modeling of imaging data[16].

- The Delaunay Triangulation (DT), dual to Voronoï tessellation [17], defines the way to connect a set of points in space into a triangulation. Such triangulation has a number of valuable properties, especially in 2D. Although the Delaunay criterion was known for many years, the construction of DT using incremental insertion algorithm (in n dimensions) was first proposed by Bowyer[18] and Watson[19], with later work of Lawson[20]. There were also proposed a number of algorithms for generation of the Delaunay triangulation of a convex hull of points employing well-known techniques like divide and conquer, sweeping, bucketing, gift wrapping, or higher dimensional embedding[21, 22]. For randomly distributed points these methods can achieve $O(n \log n)$ complexity. However, since the Delaunay algorithm in its essential form creates only a convex hull of existing set of points, the additional operations are necessary for imposing the boundary and generation of additional nodes within the meshed geometry[23–26]. Detailed description of Delaunay-based techniques (and related problems) can be found in books by George and Borouchaki [27], and by Frey and George [28].

- In Advancing Front Technique (AFT) the mesh is created by inserting vertices and elements by layers starting from boundary and moving within the discretized domain[29–32]. This technique is characterized by high quality of elements near the boundary. Unfortunately, it has its problems with operation of closing the front, despite many proposed heuristics[29, 33].

Quadrilateral/hexahedral meshes Although the automated generation of quadrilateral and hexahedral meshes is more difficult than creation of simplicial ones, there are a number of applications, where such meshes are very desirable. There have been proposed a number of techniques creating quadrilateral/hexahedral meshes either directly or indirectly (i.e. by converting the previously generated triangular/tetrahedral mesh).

The multi-block mapped meshing is one of the first approaches to generation of non-simplicial meshes, where the domain is divided into a number of topologically simple

blocks[34, 35], which could be discretized using some structural-based technique. For an automatic decomposition of complex domains the medial-axes and medial-surface techniques[36] combined with some heuristic rules have been proposed. Still, the automatic decomposition problem is not a trivial one, and the method provides a limited flexibility of the mesh size control. In paving[37, 38] method the quadrilateral elements are created in layers along the front, with additional resolving (seaming) of special configurations. The three-dimensional version, plastering[39], is still under an intensive research due to its substantial complexity.

The indirect methods rely on a triangular/tetrahedral discretization of the domain, which is then converted into quadrilateral/tetrahedral mesh. In the simplest approach each triangle in the mesh can be split into three quadrilaterals (each tetrahedron into 4 hexahedra), but the quality of created meshes is very poor. Therefore, the joining of triangle pairs is typically used instead. The selection of triangles to join may be guided by an advancing front approach (as described by Lee and Lo[40, 41] or Owen[42]) or using circle/sphere packing[43–45]. Some attempts to extend this approach for hexahedral meshing[46] have been also described.

Mesh Optimization and Quality In most of the publications concerning mesh generation problem some sort of mesh optimization techniques [47–64] are included. The goal of mesh improvement is to increase the overall quality of the mesh as well as to reduce the number of elements with poor quality. The mesh quality is strongly connected with its application, e.g. it may be expressed by the accuracy and stability of the numerical solution. However, for an automated mesh generation problem such definition is impractical. Instead, a number of geometric measures[65–74] have been proposed, which are used to evaluate a general quality of meshes. Special quality indicators have been also published for anisotropic meshes[75–78], where purely geometric criteria are insufficient.

The most commonly used mesh improvement method is the Laplace smoothing based on iterative changing of the positions of nodes without topological modifications. Although computationally inexpensive and usually advantageous, this method may also decrease local quality in specific configurations. That is why Laplace smoothing is typically coupled with others topology (e.g. edge swapping[47]) or optimization based techniques[79]. The optimization based methods allow to obtain very good quality of the elements in the mesh, but the computational cost is usually high. The optimization usually applies variational[80], line search[81] or physical based methods[82].

1.2 Goals and Thesis

The main goal of this thesis is the development of efficient algorithms of automatic discretization of non-trivial three-dimensional geometries, optimized with respect to computational complexity and numerical stability.

A special attention is directed to the appropriate selection of data structures and representations of discrete meshes conforming to the object-oriented paradigm, ensuring extendability of the meshing system.

An important and innovative part of this work is development of the method of a

fully automated mesh generation for two- and three-dimensional domains with controllable parameters of the discretization process.

These goals are set in order to verify the following thesis:

Proper selection of discretization algorithms and data structures with object-oriented approach allows scalable and functional generation of meshes adapted to considered problems.

1.3 Outline of the Thesis

In this work an object oriented implementation of surface and volume mesh generation is presented. In the subsequent chapters there are described the applied techniques (e.g. anisotropic metric transformation), proposed structures and class hierarchies (for geometry, sizing and mesh representation) and an implementation of selected meshing algorithms (realizing the automated sizing, triangular/quadrilateral mesh generation on parametric surfaces and tetrahedral volume mesh generation).

The general schema of the implemented mesh generator, which highlights the main issues of the work, is depicted on Fig. 1.1. White boxes represent main tasks of the mesh generation process which were inspected in this work. Rounded boxes represent results (intermediate or final) created during the meshing.

The contents of this work has been organized as follows:

1. The concept of metric transformation tensor for producing anisotropic unstructured meshes in both two- and three-dimensions is introduced in Chapter 2 which is then utilized in all subsequent chapters. Using the coordinate transformation system allows to easily introduce the anisotropic metric in a wide number of meshing algorithms. For surface meshing in parametric space a concept of an associated parameterization tensor is proposed. A set of elementary operations like interpolation, intersection, smoothing, and comparison is provided for the selected form of metric. Aside from the direct formulation of metric transformation tensor, it can also be calculated automatically from a number of geometrical sources. The retrieving of metric tensor from curvature of surface or contour, and a metric generated by a simplex is described, which are used in the later phases of mesh generation process. Finally a selection of mesh quality coefficients are presented, which can be used to evaluate anisotropic meshes.

2. The creation and management of sizing metric throughout the discretized domain in a special structure called control space is described in Chapter 3. Instead of implementing one selected structure, a more general approach is used. The concept of an abstract adaptive control space interface is proposed which defines a set of operations necessary for initialization and utilization of such control space during the meshing process. Two example structures, quadtree/octree grid and background mesh, were implemented and tested. There are also proposed several optimizations for application of the control space during meshing. An iterative procedure of automated construction of control space for both surface and volume meshing is presented. The

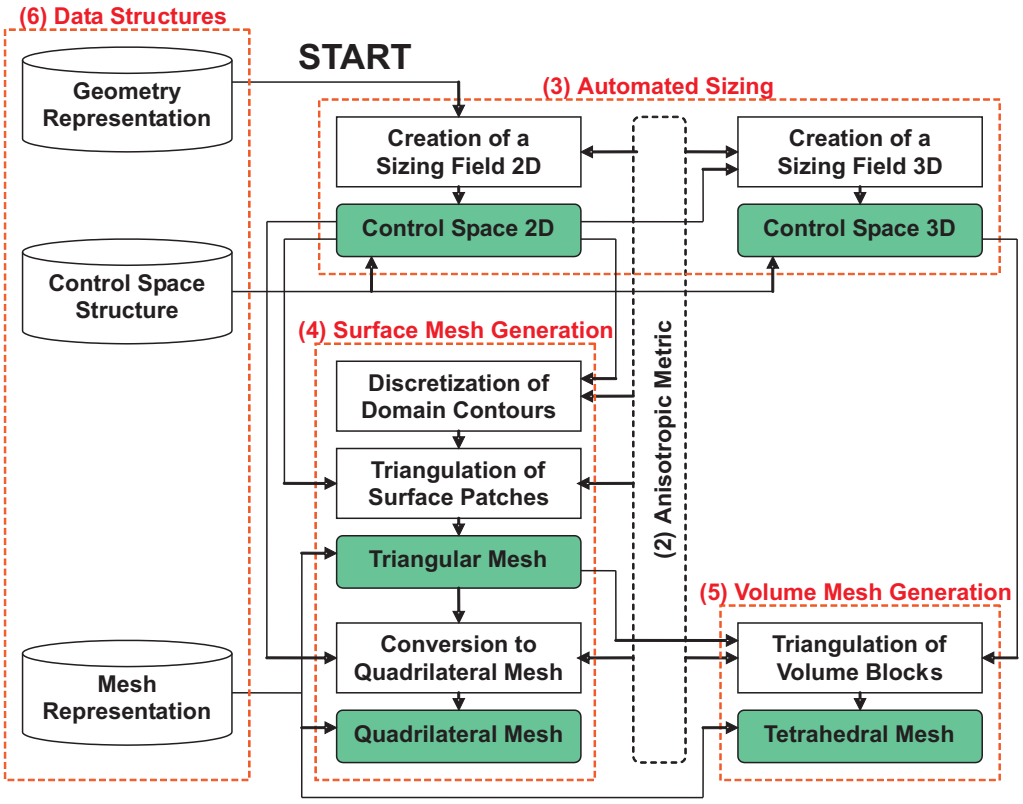


Figure 1.1: General schema of developed mesh generator

additional information which becomes available at the successive steps of mesh generations is used to adaptively refine the sizing field as required. Finally, there are listed the most important parameters guiding the process of control space initialization and their influence on the characteristic of the created mesh is shown in a number of examples.

3. In Chapter 4 the successive steps of mesh generation on parametric surfaces are presented. The triangular mesh of each surface patch is created using an incremental Delaunay triangulation algorithm in parametric space with anisotropic metric transformation. The boundary nodes created by discretization of patch contours are successively introduced into the mesh using the Delaunay criterion. The obtained convex hull triangulation is constrained by recovering boundary edges and removing obsolete elements. All triangles are evaluated and additional inner nodes are inserted until the metric *unit mesh* is achieved. Finally, some mesh improvement methods are applied. The selected steps of the presented algorithm (e.g. criteria of retriangulation, placement of inner nodes, metric introduction, etc.) are inspected in detail and a number of improvements are proposed in order to increase the efficiency of the process while preserving the required mesh quality level. The overall computational complexity is

analyzed and the results of practical experiments are provided.

For generation of quadrilateral meshes an indirect frontal approach is inspected which iteratively converts mesh triangles into quadrilaterals. Two methods described in literature (merging and morphing of triangles) were implemented with necessary adjustments for anisotropic meshing with metric transformation tensor approach. As a result of this research a mixed method is proposed where both inspected conversion method may be used alternately, depending on the local metric characteristics.

4. The generation of three-dimensional tetrahedral meshes is described in Chapter 5. The general algorithm is similar to the two-dimensional case, the Delaunay incremental insertion algorithm is also used for volume meshing. However, the extension of this algorithm is not straightforward – some aspects like boundary recovery, which can be easily done in 2D, are becoming a serious challenge in three dimensions. A multi-step procedure coping with this problem was implemented in order to avoid the necessity of inserting additional nodes at the domain boundary. A concept of systematic introduction of auxiliary nodes during the boundary mesh nodes triangulation is proposed, which allows to significantly increase the efficiency of the triangulation process.

5. In order to assure the extendability of the created mesh generation system an object oriented approach was used. The specification of the three main class hierarchies: for domain geometry representation, control space structure and mesh representation is described in Chapter 6. Some optimization methods oriented on increasing the efficiency of mesh management are also proposed. Finally, the computational complexity of the presented implementation is analyzed and a number of example results are shown.

1.4 Terminology

vertex, node (V) a zero-dimensional part of a higher dimensional entity, e.g. part of a geometry, a polygon, or polyhedron,

n-simplex $(n+1)$ affinely independent vertices,

edge (E) a one-simplex,

element (K) a polygonal or polyhedral piece in the mesh (usually triangle or quadrilateral in 2D, tetrahedron or hexahedron in 3D),

face (F) a two-dimensional entity, e.g. a polygon,

triangulation an arrangement of simplices (triangles or tetrahedra) completely covering a convex hull of a given set of vertices, without element overlapping and gaps,

constrained triangulation a triangulation with correct representation of a given set of boundary edges (and facets in 3D), not necessary convex,

mesh $\mathcal{T}_h$ of open domain Ω with boundary $\Gamma \subset \mathbb{R}^d (d = 2 \text{ or } 3)$ (where Γ is a set of segments or polygons) is a *mesh* if:

1. $\Omega = \bigcup_{K \in \mathcal{T}_h} K$,
2. each element K of $\mathcal{T}_h$ has a non-empty interior,
3. intersection of interiors of any two elements is empty,
4. intersection of two elements of $\mathcal{T}_h$ is either an empty set, a vertex, an edge or a face (in $\mathbb{R}^3$).

anisotropic mesh a mesh using stretched elements, where the desired edge length is a function of orientation,

graded mesh a mesh using elements that vary in size as a function of position,

structured mesh a mesh, where all elements and nodes (save for the boundary ones) have the same topology (i.e. same number of neighbors),

unstructured mesh a mesh, where elements and nodes can have different topology,

control space (CS) a covering of a domain responsible for storing sizing information,

Chapter 2

Anisotropic Metric Transformation

2.1 Introduction

Unstructured anisotropic mesh adaptation is now widely used in numerical simulations to improve the accuracy of the solution, reduce the computational time and properly capture the behavior of the simulated phenomena. The ever increasing requirements for automated mesh generator include the necessity of precise adjusting of the size, shape and quality of elements in the selected areas of the geometric domain. The desired element size and shape is strongly influenced by the solution field which in transient problems is constantly evolving [83, 84]. Since some physical problems exhibit strong anisotropic phenomena (e.g. some flow and phase change problems), the use of anisotropic meshes allow to maintain the required density in the locally selected directions, while using more coarse distribution of nodes in other directions.

2.1.1 Related Research

The anisotropic mesh adaptation techniques can be classified into three categories:

- mesh construction (remeshing) – the anisotropic mesh is generated using an algorithm governed by desired size and shape information in some form [85–87],
- mesh subdivision – the initial isotropic mesh is refined by specific split operations [31],
- mesh modification – a selected number of local modification operators are applied (usually consisting of swap, split, collapse and relocation) iteratively [88–90].

The remeshing gives usually the best quality of the mesh, although it can be inefficient if the meshes from two computational steps do not differ much, also the problem of solution field transfer must be solved. The subdivision avoids these problems, but it tends to produce over refined meshes and may have trouble recognizing curved boundaries. The third method allow to both coarsen and uncoarsen the mesh, but its effectiveness in terms of efficiency

and quality of may strongly depend on the selected set of modifications and manner of theirs application.

All these adaptation techniques require a way to control the anisotropic characteristics of the mesh. The information about stretch and size can be used e.g. for direct evaluation of mesh entities [31] or with an ellipse-packing approach [91, 92]. However, the most popular method used currently is introduction of anisotropy through a non-Euclidean metric used to evaluate the meshing criteria (typically by measuring metric length of mesh edges). This concept was proposed by Vallet[93] and further developed by Inria group[86, 87, 94] and others[95]. Since then it became a widely used approach for generation of unstructured anisotropic meshes in two and tree dimensions [27, 28].

2.1.2 Contents

This chapter presents definition of anisotropic metric and its introduction into the meshing process (Sec. 2.2). There are also described elementary operations (e.g. interpolation, intersection or comparison) for the proposed metric form (Sec. 2.3). Section 2.4 describes the implemented methods of metric data retrieving from different sizing sources. In Sec. 2.5 the selected coefficients of anisotropic mesh quality are defined.

2.2 Definition of Anisotropic Mesh Metric

The domains of mesh generations Ω will be the two and three dimensional manifolds with a boundary (surface patches and lumped domains) totally embedded in the three dimensional Euclidean space. Let us assume that there exists a metric tensor field on Ω which coordinates constitutes the 2×2 or 3×3 matrices $\mathcal{M}(P), P \in \Omega$ respectively [96]. For the sake of simplicity, the dependency of metric tensor coordinates on the manifold point will be frequently dropped in the description if it does not lead to ambiguity. Moreover we assume that both the manifold and the metric tensor field are regular enough, so that all operations performed later will be well defined. This tensor (specifying size, shape and direction of elements) can be written in matrix form for each point $P \in \Omega$ as $\mathcal{M}(P) = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1}$, where $\mathbf{R}$ is the eigenvector matrix (responsible for defining main directions e_i) and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_i)$ is the eigenvalue matrix, defining the required length of element edges along the main directions ($\lambda_i = 1/h_i^2,\ i = 1, \ldots, d$).

Using this formulation, the distance between two points P_1 and P_2 in metric space can be defined as $\sqrt{\overrightarrow{P_1P_2}^T \mathcal{M} \overrightarrow{P_1P_2}}$ where $\mathcal{M}$ is assumed as locally constant along $\overrightarrow{P_1P_2}$. For variable metric between points, the distance can be defined as $\int_0^1 \sqrt{\overrightarrow{P_1P_2}^T \mathcal{M}(P(t)) \overrightarrow{P_1P_2}}\mathrm{d}t$ where $\mathcal{M}(P(t))$ stands for the metric tensor at the point $P(t) = P_1 + t\overrightarrow{P_1P_2}$ [97].

$\mathcal{M}$ is a symmetric positive-definite matrix [28]. As such it can be always decomposed into two matrices $\mathbf{M}$ and $\mathbf{M}^T$

$$\mathcal{M} = \mathbf{M}\mathbf{M}^T \ . \tag{2.1}$$

$\mathbf{M}$ can be considered as a *transformation tensor* from the physical space to a non-Euclidean metric space.

In most of the recently proposed anisotropic mesh generators the metric is introduced using the metric tensor to calculate the length of edges, which can be somewhat limiting. The two-dimensional application of the metric transformation tensor to anisotropic meshing was introduced by Vigo et al. in [98, 99]. It is also mentioned briefly in [89]. Simpson in [100] proposes a concept of *geometry independence* for a meshing engine, introducing a transformed *longest-edge coordinate system*.

The developed metric transformation system for both surface and volume meshing proposed in this work is somewhat similar to both mentioned concepts. Using this unified approach allows to introduce anisotropy into a wide range of mesh generation and modification algorithms.

2.2.1 Surface Meshing

For discretization of 3D surfaces in $\mathbb{R}^3$, an explicit parametric form is assumed. Each surface patch is specified by parameterization

$$\mathbf{p} : \mathcal{S} \to \mathcal{X}, \quad \mathcal{S} \subset \mathbb{R}^2, \quad \mathcal{X} \subset \mathbb{R}^3, \quad \mathbf{s} = (u, v), \quad \mathbf{p}(\mathbf{s}) = [\mathbf{p}^1(\mathbf{s}), \mathbf{p}^2(\mathbf{s}), \mathbf{p}^3(\mathbf{s})] \ , \tag{2.2}$$

regular and with continuous second partial derivatives, where the two-dimensional domain $\mathcal{S}$ is referred to as a parametric domain.

Each surface patch is discretized in its parametric two-dimensional space. The selected metric representation is also two-dimensional with an additional correction for distortion introduced by parameterization. The transformation tensor form $\mathbf{M}$ was used in the presented approach rather than the metric tensor form $\mathcal{M}$.

Sizing Component

The *sizing matrix* representing the desired size and shape of elements could be calculated as a simple product of a rotation and scaling matrices

$$\mathbf{M_s}(P) = \mathbf{RS}, \quad P \in \mathcal{S} \tag{2.3}$$

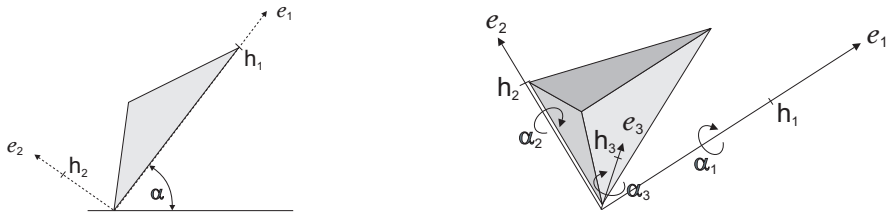
where

$$\mathbf{R} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}, \qquad \mathbf{S} = \begin{bmatrix} \frac{1}{h_1} & 0 \\ 0 & \frac{1}{h_2} \end{bmatrix}$$

and $\alpha(P)$ is the rotation angle of eigenvectors e_i, $h_i(P)$ are the required lengths of edges along the base directions (Fig. 2.1) at some point $P \in \mathcal{S}$.

Unfortunately, the matrix obtained from the equation 2.3 lacks several important properties (e.g. uniqueness and symmetry [101]), which are advantageous for operations like comparison or interpolation. Consequently, another formulation was used in this work [98]

$$\mathbf{M_s}(P) = \mathbf{RSR}^T = \begin{bmatrix} \frac{1}{h_1}\cos^2\alpha + \frac{1}{h_2}\sin^2\alpha & (\frac{1}{h_1} - \frac{1}{h_2})\sin\alpha\cos\alpha \\ (\frac{1}{h_1} - \frac{1}{h_2})\sin\alpha\cos\alpha & \frac{1}{h_1}\sin^2\alpha + \frac{1}{h_2}\cos^2\alpha \end{bmatrix} . \tag{2.4}$$



(a) two-dimensional simplex (triangle) (b) three-dimensional simplex (tetrahedron)

Figure 2.1: Anisotropic element specification

Parameterization Component

In order to account for possible distortion introduced by an arbitrary parameterization of the given surface patch, the first fundamental form $\mathbf{I}$ is used.

Definition 2.1 *Let $T_P\mathcal{S}$ be a tangent space at point $P \in \mathcal{S}$. Then the first fundamental form $\mathbf{I} : T_P\mathcal{S} \times T_P\mathcal{S} \to \mathbb{R}$ is the inner product of tangent vectors*

$$\mathbf{I}(\mathbf{x}_1, \mathbf{x}_2) \equiv \langle D\mathbf{p}(\mathbf{x}_1), D\mathbf{p}(\mathbf{x}_2) \rangle, \quad \mathbf{x}_1, \mathbf{x}_2 \in T_P\mathcal{S}, \tag{2.5}$$

where $D\mathbf{p}(\mathbf{x}) = u\mathbf{p}_u + v\mathbf{p}_v$, $D\mathbf{p}(\mathbf{x}) : T_P\mathcal{S} \to \mathbb{R}^3$, $\mathbf{x} = [u, v]^T$, D denotes differential, $\mathbf{p}_u$ and $\mathbf{p}_v$ are partial derivatives of patch $\mathbf{p}$ at point $P \in \mathcal{S}$ [102].

The first fundamental form may be written in the matrix form

$$\mathbf{I}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{G} \mathbf{x}_2, \quad \mathbf{x}_1, \mathbf{x}_2 \in T_P\mathcal{S} \tag{2.6}$$

where $\mathbf{G}$ is symmetric, positive-definite matrix (for non-degenerate parameterizations) composed of scalar products of partial derivatives $\mathbf{p}_u$ and $\mathbf{p}_v$ of patch $\mathbf{p}$ at point $P \in \mathcal{S}$

$$\mathbf{G} = \begin{bmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{bmatrix} = \begin{bmatrix} \langle \mathbf{p}_u, \mathbf{p}_u \rangle & \langle \mathbf{p}_u, \mathbf{p}_v \rangle \\ \langle \mathbf{p}_u, \mathbf{p}_v \rangle & \langle \mathbf{p}_v, \mathbf{p}_v \rangle \end{bmatrix} . \tag{2.7}$$

In case of degenerate parameterization (e.g. near the pole of the sphere parameterized with the spherical system), the coefficients g_{11} and g_{22} may become equal to 0. Such situation is unacceptable and the $g'_{ii} = \min(g_{ii}, g_{\min})$ value is assumed instead. This amendment allows to perform the meshing process successfully, although it must be noted that reparameterization of this area would typically give better results with respect to the obtained quality of elements.

The *parameterization matrix* $\mathbf{M_p}$ in point $P \in \mathcal{S}$ is calculated from G as a symmetric matrix fulfilling the equation

$$\mathbf{G} = \mathbf{M_p} \mathbf{M_p}^T . \tag{2.8}$$

Transformation Tensor

Finally, the product of the sizing matrix and the parameterization matrix is used during the meshing process to calculate the transformation tensor in point $P \in \mathcal{S}$

$$\mathbf{M} = \mathbf{M_s} \mathbf{M_p} . \tag{2.9}$$

The transformation introduced by $\mathbf{M}$ maps the domain $\mathcal{S}$ onto $\mathcal{S}^{\mathcal{M}} \subset \mathbb{R}^2$ and for each point $P \in \mathcal{S}$

$$P^{\mathcal{M}} = \mathbf{M}P, \quad P \in \mathcal{S}, \quad P^{\mathcal{M}} \in \mathcal{S}^{\mathcal{M}} \ . \tag{2.10}$$

Using this mapping, the distance in metric space between two points $P_1, P_2 \in \mathcal{S}$ can be calculated as Euclidean distance between the transformed points $P_1^{\mathcal{M}}, P_2^{\mathcal{M}}$

$$d_{\mathcal{M}}(P_1, P_2) = d(P_1^{\mathcal{M}}, P_2^{\mathcal{M}}) \ , \tag{2.11}$$

where the metric transformation tensor $\mathbf{M}$ is assumed to be locally constant along $\overrightarrow{P_1P_2}$.

Another useful transformation introduced by $\mathbf{M_p}$ maps the domain $\mathcal{S}$ onto $\mathcal{S}^{\mathcal{R}} \subset \mathbb{R}^2$ and for each point $P \in \mathcal{S}$

$$P^{\mathcal{R}} = \mathbf{M_p}P, \quad P \in \mathcal{S}, \quad P^{\mathcal{R}} \in \mathcal{S}^{\mathcal{R}} \ . \tag{2.12}$$

The distance between two points $\mathbf{p}(P_1), \mathbf{p}(P_2), P_1, P_2 \in \mathcal{S}$ can be calculated as Euclidean distance between the transformed points $P_1^{\mathcal{R}}, P_2^{\mathcal{R}}$

$$d_{\mathcal{R}}(P_1, P_2) = d(P_1^{\mathcal{R}}, P_2^{\mathcal{R}}) \ , \tag{2.13}$$

where the parameterization matrix $\mathbf{M_p}$ is assumed to be locally constant along $\overrightarrow{P_1P_2}$.

For variable metric, assuming linear interpolation of metric between two points and small metric difference, the distance can be also approximated by (2.11) and (2.13) calculating transformation matrix at the middle of segment P_1P_2

$$\mathbf{M} = \mathbf{M}(P_1 + 0.5\overrightarrow{P_1P_2}) \ . \tag{2.14}$$

2.2.2 Volume Meshing

Sizing Component

As in two-dimensional case, the symmetrical form of sizing matrix is also preferred

$$\mathbf{M_s}(P) \quad = \quad \mathbf{RSR}^T, \quad P \in \Omega \tag{2.15}$$

where columns of $\mathbf{R}$ are created by eigenvectors $e_i(P)$ (defining main directions) and the diagonal of $\mathbf{S}$ is composed of the inverted lengths $h_i(P)$ of edges along the main directions (Fig. 2.1) at point $P \in \Omega$.

Transformation Tensor

Since no parameterization is required for 3D meshing, the sizing matrix is directly used to construct the transformation tensor

$$\mathbf{M} = \mathbf{M_s} \ . \tag{2.16}$$

The transformation introduced by $\mathbf{M}$ maps the domain Ω onto $\Omega^{\mathcal{M}} \subset \mathbb{R}^3$ and for each point $P \in \Omega$

$$P^{\mathcal{M}} = \mathbf{M}P, \quad P \in \Omega, \quad P^{\mathcal{M}} \in \Omega^{\mathcal{M}} \ . \tag{2.17}$$

Using this mapping, the distance in metric space between two points $P_1, P_2 \in \Omega$ can be calculated as in (2.11). Since no parameterization is used for volume meshing, $d_{\mathcal{R}}(P_1, P_2)$ is equal to $d(P_1, P_2)$.

2.2.3 Ideal Conformity Criteria

With metric defined within the whole discretized domain, the desired mesh is then a *unit mesh*. The precise definition of the *unit mesh* may slightly vary in different approaches. The most common requirement for triangular meshes (also used in this work) is that the length of each edge E calculated in the specified metric should be equal to one.

For non-simplicial (e.g. quadrilateral) or higher dimension meshes the geometrical shape of elements has to be additionally enforced to be as close to equilateral (in metric space) as possible. In some works the definition is slightly different, for example Dolejsi in [103] defines 'unit-triangle' as an equilateral triangle inscribed into circle with unitary radius. An additional criterion, which could be included into mesh optimality evaluation, is aligning of mesh edges to the main directions of defined metric (i.e. Li in [89]).

2.3 Metric Related Operations

2.3.1 Interpolation

Metric is stored in discrete points of the control space but the continuous information is required during meshing process. In order to facilitate this the metric has to be appropriately interpolated between points. Although the actual methods of interpolation depend upon the type of control space structure, they are all derived from the most simple case – interpolation of metric between two points.

The metric can be interpolated in a number of ways [104]. The selected method may depend on application, the most commonly used approach utilizes simultaneous diagonalization and linear interpolation of the diagonal form [86]. In this work a direct linear interpolation of transformation tensor was used, which can be efficiently calculated for the selected form of metric representation. For metric defined as the transformation tensors in two points P_1 and P_2, the interpolated metric at some point Q lying on the P_1P_2 segment can be calculated as

$$\mathbf{M_s}(Q) = \left((1-t)\mathbf{M_s}^{-1}(P_1) + t\mathbf{M_s}^{-1}(P_2)\right)^{-1} \tag{2.18}$$

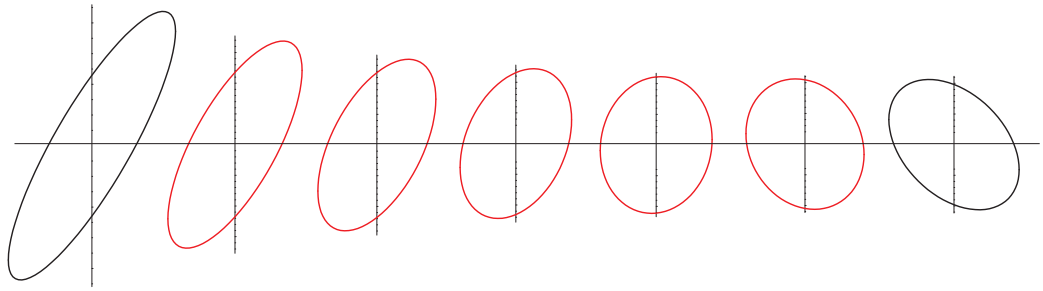
where $Q = P_1 + t\overrightarrow{P_1P_2}$.

Fig. 2.2 illustrates examples of two- and three-dimensional metric interpolation between two points (the metric is represented as ellipses or ellipsoids).

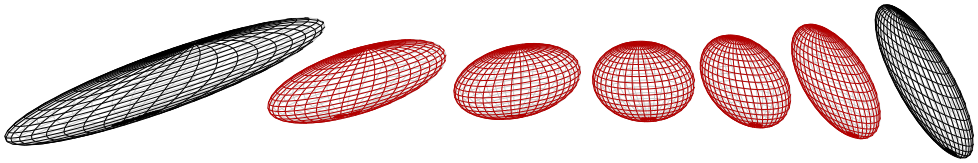
2.3.2 Intersection

If more than one type of metric source is available at some discrete point, the metric intersection procedure is used. The intersection metric can be geometrically interpreted as an ellipse (or ellipsoid) with the largest area (volume) contained within ellipses (or ellipsoids) associated with all the given metrics being intersected (Fig. 2.3).

An intersection of two metrics $\min_{\mathcal{M}}(\mathbf{M_{s1}}, \mathbf{M_{s2}})$ can be evaluated as a simultaneous reduction or diagonalization of quadratic forms [95, 104]. However, since the transformation matrix $\mathbf{M}$ is used instead of the metric tensor $\mathcal{M}$, the intersection method has to

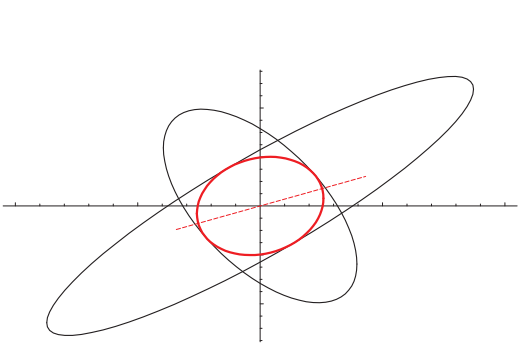


(a) two-dimensional

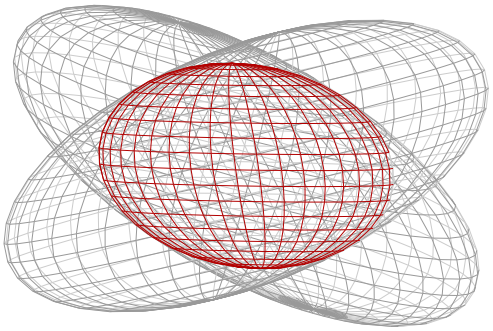


(b) three-dimensional

Figure 2.2: Interpolation of metric



(a) two-dimensional



(b) three-dimensional

Figure 2.3: Intersection of metric

be accordingly adjusted. The algorithm is similar in two- and three-dimensional cases. It should be noted that for surface meshing only the sizing tensor (without the parameterization adjustment) is considered. The implemented procedure consists of the following three phases:

1. Transformation of both tensors into metric tensor form.
2. Calculation of the intersection metric tensor $\mathcal{M}_{\min}$.
3. Transformation of the resultant metric back into transformation tensor form.

If there are more than two metrics defined in one point, all tensors have to be transformed into metric tensor form. Then, the second phase is carried out for the two first

tensors and the resultant metric is further intersected with all subsequent metrics one by one. Finally, the resultant tensor is converted back into the transformation tensor.

Calculating metric tensor from transformation tensor form

The easiest and most efficient way is to use the formula (2.1) directly and the required d-dimensional metric tensor can be calculated as

$$\mathcal{M} = \mathbf{M_s}\mathbf{M_s}^T \ . \tag{2.19}$$

Intersecting two metric tensors

In order to find the intersection of two metric tensors $\mathcal{M}_1$ and $\mathcal{M}_2$ the simultaneous diagonalization is used. The base $(e_i),\ i = 1, \ldots, d$ in which both tensors are diagonal can be found by calculating the eigenvectors of a tensor $\mathcal{N}$ constructed as

$$\mathcal{N} = \mathcal{M}_1^{-1}\mathcal{M}_2 \ . \tag{2.20}$$

Eigenvalues of $\mathcal{M}_1$ and $\mathcal{M}_2$ in this base are found using the Rayleigh quotient [104]

$$\lambda_{1i} = e_i^T \mathcal{M}_1 e_i \text{ and } \lambda_{2i} = e_i^T \mathcal{M}_2 e_i \ , \quad i = 1, \ldots, d \ . \tag{2.21}$$

Let $\mathbf{A}$ be a $d \times d$ matrix which columns are formed by respective eigenvectors e_i of $\mathcal{N}$. The intersection metric tensor $\mathcal{M}_{\text{min}}$ can be calculated as

$$\mathcal{M}_{\text{min}} = \mathbf{A}^{-1^T} \text{diag}(\lambda_i^{\text{max}}) \mathbf{A}^{-1} \ , \tag{2.22}$$

where $\lambda_i^{\text{max}} = \max(\lambda_{1i}, \lambda_{2i}), i = 1, \ldots, d$.

Converting metric tensor into symmetric transformation tensor

In order to calculate the matrix $\mathbf{M_s}$ the tensor $\mathcal{M}_{\text{min}}$ has to be decomposed into $\mathbf{RSR}^T$, where $\mathbf{S} = \text{diag}(\lambda_i)$ is a diagonal matrix formed by eigenvalues of $\mathcal{M}_{\text{min}}$ and the columns of $\mathbf{R}$ are calculated as normalized eigenvectors of $\mathcal{M}_{\text{min}}$. In case of degenerate (i.e. non-distinct) eigenvalues, the set of eigenvectors is selected to be complete and orthonormal. Since $\mathcal{M}_{\text{min}}$ is real, symmetric and positive definite, all eigenvalues are guaranteed to be positive and matrix $\mathbf{R}$ to be orthogonal. The eigensystem for both two- and three-dimensional matrices can be calculated by directly solving the characteristic equation.

The resultant metric $\min_{\mathcal{M}}(\mathbf{M_{s1}}, \mathbf{M_{s2}})$ in form of a sizing matrix $\mathbf{M_s}$ can be calculated as

$$\min_{\mathcal{M}}(\mathbf{M_{s1}}, \mathbf{M_{s2}}) = \mathbf{RS'R}^T \ , \tag{2.23}$$

where $\mathbf{S'} = \text{diag}(\sqrt{\lambda_i})$.

2.3.3 Comparison

In several phases of the mesh generation process (e.g. during an adaptation of a control grid to curvature of a surface patch) an operation of measuring the difference between two

metrics is required. Comparison of metrics can be also used to measure the metric quality of mesh elements.

For two metric transformation tensors $\mathbf{M_{s1}}, \mathbf{M_{s2}}$ the following residuals are calculated

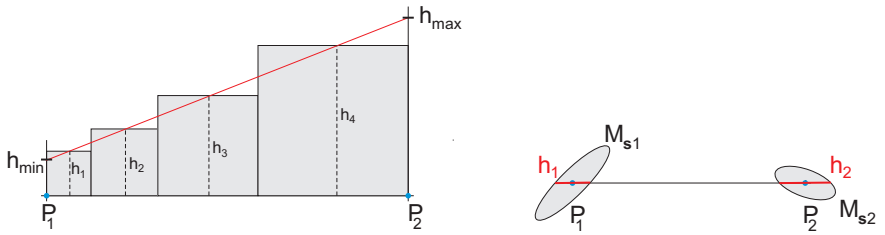
$$\begin{aligned} R_1 &= \mathbf{M_{s1}}^{-1}\mathbf{M_{s2}} - I \\ R_2 &= \mathbf{M_{s2}}^{-1}\mathbf{M_{s1}} - I \end{aligned} \tag{2.24}$$

and the difference is calculated using Euclidean norm of a matrix [75, 105]

$$\delta_{\mathcal{M}}(\mathbf{M_{s1}}, \mathbf{M_{s2}}) = ||R_1 + R_2|| \ . \tag{2.25}$$

2.3.4 Smoothing

Metric data come from different sources and the resulting metric map can have unacceptably large variation. While there have been presented a number of approaches for the gradation control of isotropic meshes [33, 97, 106], the anisotropic case still is not fully solved [97, 107]. The approach presented in this work allows to control the gradation of anisotropic sizing in both two- and three-dimensions. Since the required variation of the mesh elements may depend on the application, the gradation parameter $\gamma_{\mathcal{M}}$ was introduced. This gradation parameter simply limits the maximum increase of the required edge length (in any direction) between any two points of unitary distance (in metric space). Figure 2.4(a) presents an example case of maximum gradation of 1D elements length, where $\gamma_{\mathcal{M}} = h_{i+1}/h_i$.



(a) gradation ratio (b) length along selected direction

Figure 2.4: Acceptable metric transition

For any two points P_1 and P_2 the procedure starts with comparing the associated metrics $\mathbf{M_{s1}}$ and $\mathbf{M_{s2}}$. If $\delta_{\mathcal{M}}(\mathbf{M_{s1}}, \mathbf{M_{s2}}) < \epsilon_\delta$ (where ϵ_δ is some constant parameter) the rest of this procedure may be skipped. Otherwise, the requested length of elements along the selected vector $\overrightarrow{P_1P_2}$ is calculated as h_1 and h_2 (Fig. 2.4(b)). Maximum and minimum values are set as $h_{\max}$ and $h_{\min}$.

The coefficient a is calculated as

$$a = \frac{h_{\max} - h_{\min}}{d_{\mathcal{R}}(P_1, P_2)} \tag{2.26}$$

and is compared with $a_{\max}$ which is defined by the gradation ratio

$$a_{\max} = 2\frac{\gamma_{\mathcal{M}} - 1}{\gamma_{\mathcal{M}} + 1} \ . \tag{2.27}$$

If $a > a_{\max}$ the values are accordingly adjusted

$$a \leftarrow a_{\max} \tag{2.28}$$
$$h_{\max} \leftarrow a\, d_{\mathcal{R}}(P_1, P_2) + h_{\min}\ . \tag{2.29}$$

The maximum gradation factor s is established as

$$s = 1 - \frac{(1-\gamma_{\mathcal{M}})(1-a/2)d_{\mathcal{R}}(P_1,P_2)}{h_{\min}}\ . \tag{2.30}$$

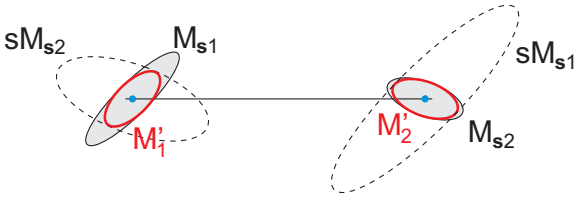


Figure 2.5: Constraining metric tensors

Using this factor metrics in both nodes can be adjusted (Fig. 2.5)

$$\mathbf{M'}_{s1} \leftarrow \min_{\mathcal{M}}(\mathbf{M}_{s1}, s\mathbf{M}_{s2}) \tag{2.31}$$
$$\mathbf{M'}_{s2} \leftarrow \min_{\mathcal{M}}(\mathbf{M}_{s2}, s\mathbf{M}_{s1})\ . \tag{2.32}$$

2.4 Metric Sources

The metric tensor can be given directly, but for automated mesh generation it is important to be able to gather sizing information from different sources.

2.4.1 Metric of Simplex

For each non-degenerate simplex K there is exactly one metric tensor $\mathcal{M}_K$ which transforms K to unit simplex. In $\mathbb{R}^d, d = 2$ or 3 the $d(d+1)/2$ components m_{ij} of the metric tensor $\mathcal{M}_K$ can be calculated by solving [93]

$$(P_j - P_i)^T \mathcal{M}_K (P_j - P_i) = 1, \text{ for } 1 \le i < j \le d+1\ . \tag{2.33}$$

For example in two dimensions, for triangle K with vertices $(P_1, P_2, P_3), P_1 = (x_1, x_2, x_3)$ we obtain the following set of equations

$$\begin{pmatrix} (x_2-x_1)^2 & 2(x_2-x_1)(y_2-y_1) & (y_2-y_1)^2 \\ (x_3-x_1)^2 & 2(x_3-x_1)(y_3-y_1) & (y_3-y_1)^2 \\ (x_3-x_2)^2 & 2(x_3-x_2)(y_3-y_2) & (y_3-y_2)^2 \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{22} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.34}$$

which has a unique solution for all non-degenerate triangles. The transformation tensor $\mathbf{M}_K$ can be calculated from $\mathcal{M}_K$ using the procedure described in 2.3.2. It should be noted that for meshing in parametric space of a surface patch, the obtained transformation tensor is a product of both sizing and parameterization components.

2.4.2 Surface Curvature

For a given surface patch with parametric formulation (2.2) the two-dimensional metric tensor can be calculated from the curvature of this surface in a given point [108].

Definition 2.2 *Let $T_P\mathcal{S}$ be a tangent space at point $P \in \mathcal{S}$. The second fundamental form* $\mathbf{II} : T_P\mathcal{S} \times T_P\mathcal{S} \to R$ *is the inner product of normal and tangent vectors*

$$\mathbf{II}(\mathbf{x}_1, \mathbf{x}_2) \equiv -\langle D\mathbf{n}(\mathbf{x}_1), D\mathbf{p}(\mathbf{x}_2)\rangle, \quad \mathbf{x}_1, \mathbf{x}_2 \in T_P\mathcal{S}, \tag{2.35}$$

where $D\mathbf{p}(\mathbf{x}) = u\mathbf{p}_u + v\mathbf{p}_v, \quad D\mathbf{p}(\mathbf{x}) : T_P\mathcal{S} \to R^3, \quad \mathbf{x} = [u, v]^T, \quad D$ *denotes differential,* $\mathbf{n}$ *is a vector normal to* $\mathbf{p}$ *at point* $P \in \mathcal{S}$, $\mathbf{p}_u$ *and* $\mathbf{p}_v$ *are partial derivatives of patch* $\mathbf{p}$ *at point* $P \in \mathcal{S}$ [102].

The second fundamental form may be written in the matrix form

$$\mathbf{II}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^T \mathbf{B} \mathbf{x}_2, \quad \mathbf{x}_1, \mathbf{x}_2 \in T_P\mathcal{S} \tag{2.36}$$

where $\mathbf{B}$ is symmetric (not necessary positive-definite) matrix

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} \langle \mathbf{n}, \mathbf{p}_{uu} \rangle & \langle \mathbf{n}, \mathbf{p}_{uv} \rangle \\ \langle \mathbf{n}, \mathbf{p}_{uv} \rangle & \langle \mathbf{n}, \mathbf{p}_{vv} \rangle \end{bmatrix} . \tag{2.37}$$

At any point $P \in \mathcal{S}$ of the patch domain the values of main curvatures χ_1, χ_2 and main directions $D\mathbf{p}(\mathbf{y}_1)$ and $D\mathbf{p}(\mathbf{y}_2)$ (where $\mathbf{y}_1, \mathbf{y}_2 \in T_P\mathcal{S}, \ \mathbf{y}_1, \mathbf{y}_2 \neq \mathbf{0}$) can be obtained from the first and second fundamental forms by finding extrema of function χ [102] given by equation

$$\mathbf{II}(\mathbf{y}, \mathbf{y}) - \chi \mathbf{I}(\mathbf{y}, \mathbf{y}) = 0 \ . \tag{2.38}$$

Using the matrix representations of fundamental forms (2.37) and (2.7) the equation (2.38) can be written as

$$(B - \chi G)\mathbf{y} = \mathbf{0} \ , \tag{2.39}$$

and the minimum and maximum values χ_1, χ_2 can be calculated solving the quadratic equation [102]

$$\det(B - \chi G) = 0 \ . \tag{2.40}$$

The curvature value along the selected direction can be interpreted as an inverse of the radius of the circle tangent to the surface at the given point. The requested length of element edges along the main directions are calculated as

$$h_i = \max(\min(\frac{\gamma_c}{\chi_i}, h_{\max}), h_{\min}), \quad i = 1, 2 \ , \tag{2.41}$$

where γ_c is the constant ratio of proportionality of the edge length and curvature radius. $h_{\min}$ and $h_{\max}$ (calculated from the diameter h_d of domain geometry and constant parameters: $h_{\min} = \gamma_{h_{\min}} h_d$ and $h_{\max} = \gamma_{h_{\max}} h_d$) are bounds necessary to avoid creation of too extreme elements. The stretch direction is calculated as an angle α_c of rotation of main directions related to the base parameterization vectors of the given surface patch.

Additionally, the anisotropy ratio γ_a is introduced controlling the maximum stretch ratio of elements

$$\frac{\max(h_i)}{\min(h_i)} \leq \gamma_a, \quad i = 1, \ldots, d \ . \tag{2.42}$$

From (α_c, h_1, h_2) the two-dimensional metric sizing component is calculated using (2.4).

2.4.3 Contour Curvature

The contour curves $\mathbf{c}$ are defined in parametric form

$$\mathbf{c} : \mathcal{C} \to \mathcal{X}, \quad \mathcal{C} \subset \mathbb{R}, \quad \mathcal{X} \subset \mathbb{R}^3, \quad \mathbf{c}(t) = [\mathbf{c}^1(t), \mathbf{c}^2(t), \mathbf{c}^3(t)] \ , \tag{2.43}$$

regular and with continuous second partial derivatives. The contours can be also defined in parametric space of patch $\mathbf{p}$

$$\mathbf{c_p} : \mathcal{C} \to \mathcal{S}, \quad \mathcal{C} \subset \mathbb{R}, \quad \mathcal{S} \subset \mathbb{R}^2, \quad \mathbf{c_p}(t) = [\mathbf{c_p}^1(t), \mathbf{c_p}^2(t)] \ , \tag{2.44}$$

where $\mathbf{c_p}$ is regular with continuous second derivatives and $\mathbf{c}$ is a superposition $\mathbf{p} \circ \mathbf{c_p}$.

The contour curvature κ can be calculated from [108]

$$\kappa = \frac{\langle \mathbf{n}, \mathbf{c}'' \rangle}{||\mathbf{c}'||^2} = \frac{||\mathbf{c}' \times \mathbf{c}''||}{||\mathbf{c}'||^3} \ , \tag{2.45}$$

where $\mathbf{c}'$ and $\mathbf{c}''$ are the first and second derivatives of $\mathbf{c}$ in point $P \in \mathcal{C}$.

The sizing parameters can be defined as

$$\begin{array}{rcl} h_1 & = & \max(\min(\frac{\gamma_c}{\kappa}, h_{\max}), h_{\min}) \\ h_2 & = & \gamma_{a_c} h_1 \\ \alpha & = & \alpha_c \end{array} \tag{2.46}$$

where α_c is calculated as an angle between the base vector of the parametric patch and the vector tangent to the contour $\mathbf{c_p}$ at point P. Since high value of anisotropy ratio γ_a can be inappropriate for curved contours (where changes of α_c are large), an additional parameter γ_{a_c} ($\gamma_{a_c} \leq \gamma_a$) was introduced here.

From (α_c, h_1, h_2) the two-dimensional metric transformation tensor is calculated using (2.4).

2.5 Mesh Quality Criteria

The actual evaluation of mesh quality depends on the application of the specific mesh. However, some kind of a universal measure is necessary for developing a mesh generator applicable in different areas. Over years many authors proposed a number of different quality coefficients for assessment of finite element meshes (composed of different types of elements). For isotropic meshes the quality coefficient of element is usually oriented on geometric quality, i.e. how close to equilateral is the given element. An extensive overview of geometric mesh quality coefficients can be found in [70, 71, 73].

2.5.1 Geometric Criteria in Metric Space

For anisotropic meshes the geometric coefficients have to be adjusted for properly evaluating elements stretched according to anisotropic metric. Quality coefficients based on lengths of edges can be easily modified by calculating these lengths with appropriate metric tensor [77]. For variable metric either an integration procedure for length calculation or metric averaging operation is used.

In this work the *mean ratio* shape measure is used as en example of this category of quality criteria. The *mean ratio* η for a regular simplex K (triangle or tetrahedron) was introduced by Liu and Joe[109] and can be written as

$$\eta = \frac{\sqrt[d]{\Pi_{i=1}^{d}\lambda_i}}{\frac{1}{d}\Sigma_{i=1}^{d}\lambda_i} = \begin{cases} \frac{2\sqrt{\lambda_1\lambda_2}}{\lambda_1+\lambda_2} = \frac{4\sqrt{3}S_K}{\Sigma_{1\leq i<j\leq 3}{l_{e_{ij}}}^2} & \text{in 2D,} \\ \frac{3\sqrt[3]{\lambda_1\lambda_2\lambda_3}}{\lambda_1+\lambda_2+\lambda_3} = \frac{12\sqrt[3]{9V_K^2}}{\Sigma_{1\leq i<j\leq 4}{l_{e_{ij}}}^2} & \text{in 3D,} \end{cases} \tag{2.47}$$

where the $\lambda_i,\ i = 1,\ldots,d$ are the eigenvalues of the metric tensor $\mathcal{M}_K$ in dimension d, $l_{e_{ij}}$ is the length of an edge in simplex joining vertices i and j, S_K and V_K are surface and volume of element K.

For measuring the quality of anisotropic elements, the coefficient η can be transformed into $\eta_{\mathcal{M}}$ which is calculated with (2.47) for vertices of the simplex transformed to metric space using an average transformation tensor for this simplex.

The mean ratio quality of a mesh $\overline{\eta}_{\mathcal{M}}(\mathcal{T}_h)$ can be defined as

$$\overline{\eta}_{\mathcal{M}}(\mathcal{T}_h) = \frac{1}{\mathrm{NK}}\sum_{i=1}^{\mathrm{NK}}\eta_{\mathcal{M}}(K_i)\ , \tag{2.48}$$

where NK is the number of elements in mesh $\mathcal{T}_h$.

2.5.2 Metric Edge Length

Geometric quality coefficients calculated in metric space properly respect the anisotropic characteristic of the required mesh. However, they still take into account only shape of elements, ignoring its size. For this reason the geometric coefficients are usually coupled with other coefficients, evaluating the size of element with respect to the prescribed sizing field. The most common approach is to use the metric length of edges (which should be unitary).

The average metric length of edges in a mesh $\overline{\mu}_{\mathcal{M}}(\mathcal{T}_h)$ can be defined as

$$\overline{\mu}_{\mathcal{M}}(\mathcal{T}_h) = \frac{1}{\mathrm{NE}}\sum_{i=1}^{\mathrm{NE}} l_{\mathcal{M}}(E_i)\ , \tag{2.49}$$

where NE is the total number of edges in mesh $\mathcal{T}_h$ and $l_{\mathcal{M}}(E_i)$ is the length of i-th edge in the mesh, calculated according to local metric. For triangular elements this measure is sufficient for evaluating both size and shape. However, for quadrilateral or tetrahedral elements unitary length of edges does not guarantee good shape of the element.

2.5.3 Metric Non-conformity Measure

Another interesting measure is the metric non-conformity coefficient [75, 76, 105]. The coefficient $\mathcal{E}_K$ of a simplex K can be calculated as a measure of discrepancy between metric tensor $\mathbf{M}_K$ of a simplex K and a metric tensor $\mathbf{M}(K)$ prescribed for this simplex in the control space

$$\mathcal{E}_K = \delta_{\mathcal{M}}(\mathbf{M}_K, \mathbf{M}(K))\ . \tag{2.50}$$

The metric tensor $\mathbf{M}(K)$ is calculated for a representative point for the simplex K.

The coefficient of non-conformity $\overline{\mathcal{E}}_K$ of a mesh $\mathcal{T}_h$ can be defined as

$$\overline{\mathcal{E}}_K(\mathcal{T}_h) = \frac{1}{\mathrm{NK}} \sum_{i=1}^{\mathrm{NK}} \mathcal{E}_{K_i} \ , \tag{2.51}$$

where NK is the number of elements in mesh $\mathcal{T}_h$.

$\overline{\mathcal{E}}_K$ can be used directly for evaluating quality of two-dimensional triangles and three-dimensional tetrahedra. For triangles on parametric surfaces (with coordinates in parametric space) the $\mathbf{M}_K$ is decomposed into sizing tensor $\mathbf{M}_{\mathbf{s}K}$ and parameterization matrix $\mathbf{M}_{\mathbf{p}K}$. Then, only the sizing tensor $\mathbf{M}_{\mathbf{s}K}$ is actually compared to tensor $\mathbf{M}_{\mathbf{s}}(K)$ from the control space.

2.6 Chapter Summary

The definition of anisotropic metric for mesh generation was presented for meshing of both the parametric surfaces and three-dimensional volumes. The selected metric form and a set of associated operations were used to create a transformed coordinate system, which allows for a fairly easy and elegant implementation of a wide range of meshing routines originally created for isotropic meshes.

Chapter 3

Control Space Implementation

3.1 Introduction

During the meshing process the generator uses a special *control space* (called shortly as CS) structure to provide the requested metric (governing the size, shape and directionality of elements) at any point within the domain being discretized. This sizing function may be influenced by the geometry of the meshed domain (e.g. curvature, feature size and proximity), topology of adjacent meshes, physical specifics (e.g. in simulations of flow, crack propagation or acoustic shock), user specification, etc.

3.1.1 Related Research

In early works the sizing function was defined by hand in a number of selected points of the meshed domain. Gradually, the process was automated and a concept of a background mesh (i.e. coarse triangulation of the sizing points within the meshed domain) has been developed[110]. For approximation of element size between the vertices, the simple triangle interpolation or some more advanced method like natural-neighbor interpolation[111] is applied. The drawback of the background mesh is its low efficiency, resulting from the necessity of finding the triangle containing the sought point. In order to speed up the retrieving of sizing information, regular grids have been also used. However, regular grids may fail to recognize the rapid local changes of sizing function, also the memory requirements can be too high. In order to alleviate these problems there are often used the hierarchical quadtree/octree grids[112], where the bounding box of the whole domain is recursively partitioned according to sizing function[10] or its gradient[113, 114]. Although it is efficient and robust, the problem of grid structure is its orientation-sensitivity. Another proposed approach is the medial axis transform used as a skeleton-based control space[11, 115].

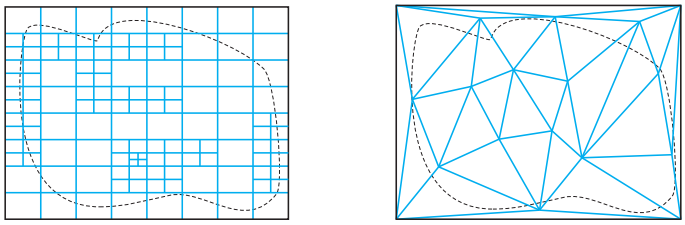
For isotropic meshes CS stores a scalar sizing function. In anisotropic case additional information like orientation or stretching are required, which makes the interpolation procedure less obvious. In meshing algorithms based on non-Euclidean metric transformation the vertices of CS structure store the metric tensor directly.

In this work a hierarchy of CS structures is proposed which provide the anisotropic sizing information in the form of metric transformation tensor. If there is available sizing data gathered from several different sources (e.g. model geometry, adaptation process, user input), a separate CS structure is created for each type of source. The resultant adaptive constrol space (ACS) is calculated using the superposition procedure, which takes into account both structure and metric data from the control vertices of all CSs.

In order to assure good quality of the created meshes the metric field within the ACS can be subjected to an additional smoothing procedure, which controls the maximum gradation ratio over neighboring elements within the mesh. For domains composed of several surface patches or volume blocks an iterative procedure of adjusting ACS for adjacent patches is used.

3.1.2 Contents

In Sec. 3.2 the general concept of ACS is described. The set of common operations is presented together with a classification of the provided metric source forms. The implementation details of two ACS structures (quadtree/octree grid and background mesh – Fig. 3.1) are provided in Sections 3.3 and 3.4. In Sec. 3.5 and iterative procedure of automated ACS creation during two- and three-dimensional mesh generation is proposed.



(a) quadtree (b) background mesh

Figure 3.1: Adaptive control space structures for surface meshing

3.2 Adaptive Control Space

In order to facilitate creation and utilization of different CS structures, the common interface for an adaptive control space (ACS) was created, defining a set of required methods [116]. Any ACS implementation should be able to initialize the CS from either continuous (e.g. curvature of surface or analytical equations) or discrete sources (e.g. discrete points or segments with metric description), interpolate or extrapolate metric in case of insufficient data, adjust metric in any sub-domain of already initialized structure and adapt its structure to variation of metric or surface parameterization.

The sizing information can be obtained directly from the user (in a number of methods), retrieved from the solver as a part of an adaptation process [117–123] (as a set of discrete points with sizing information) and/or gathered automatically from the geometric and topological description of the discretized domain [124]. The metric sources can have

different dimensions: 0D (discrete, e.g. for data from adaptation given in the vertices of a computational mesh), 1D (linear, e.g. for curvature of boundary contours), 2D (surface, e.g. for curvature of surface patch), and 3D (volumetric, e.g. from user description). All available sources are processed and stored in a single adapted ACS structure.

3.2.1 Elementary Operations

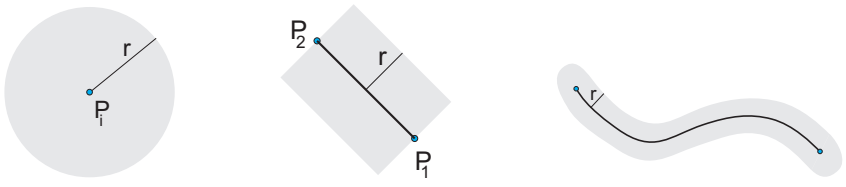
Thanks to the defined ACS interface, the automated sizing process is independent from the selected type of CS and is open for further extension of available set of ACS implementations. Apart from the most basic function returning the metric transformation tensor at the given point (which has to be defined in all implementations of CS) the ACS interface defines the following operations:

- inserting a discrete metric source at some point,
- setting a continuous metric source in the whole domain,
- initializing all control vertices basing on available metric sources (possibly requiring an extrapolation),
- adapting for variation of the parameterization matrix (two-dimensional case only),
- adjusting metric gradation according to the prescribed *metric maximum gradation ratio*,
- iterating over all control vertices of ACS.

There are also some operations which can be implemented on a general level:

- superposition of ACS and some other CS (both structure and metric values),
- inserting a metric source defined in an extended form (e.g. along some segment).

3.2.2 Extended Metric Source Definition

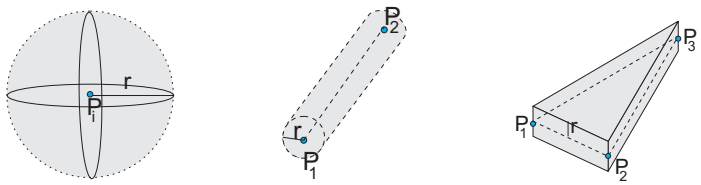


(a) circle (b) segment (c) curve

Figure 3.2: Extended metric source definition (2D)

In order to facilitate precise and convenient definition of size and stretching of elements throughout the domain, the metric source can be introduced in a few ways (Fig. 3.2, see also example mesh AGH1 – Fig. A.7):

- $(P, \mathbf{M_s}, r)$ – metric tensor $\mathbf{M_s}$ is treated as constant in the neighborhood of the point P with the radius $r \geq 0$.
- $(P_1, P_2, \mathbf{M_s}, r)$ – along the segment P_1P_2 and its neighborhood with length r the constant metric tensor $\mathbf{M_s}$ is assumed.
- $(c(t), t_1, t_2, \mathbf{M_s}, r)$ – constant metric tensor $\mathbf{M_s}$ is assumed along the given parametric curve $c(t)$ and its neighborhood with length r.



(a) sphere (b) segment (c) prism

Figure 3.3: Extended metric source definition (3D)

For three-dimensional meshing a similar set of extended metric source forms was created (Fig. 3.3) with additional possibility to describe metric within a prism defined by a face.

These forms of metric sources can be given directly by a user, they are also used by some higher-level procedures of automatic metric recognition (e.g. during adjusting for contour curvature). Processing of these forms of metric sources is implemented on a general level of ACS and consists of the following two operations:

- adaptation of ACS structure – outline of the extended metric source is represented by a set of points (with unitary distance according to metric given by this source), which are then inserted into the ACS structure as single discrete sources,
- setting the metric – all control vertices of the ACS which are contained within the domain of the extended metric source have its metric intersected with the new one.

3.2.3 Superposition

The superposition is computed as the minimum of ACS structures ACS_1 and ACS_2. The procedure consists of the following phases:

1. For each control vertex V_j of ACS_1 the intersection metric $\min_{\mathcal{M}}(\mathbf{M_s}_{V_j}^{\mathrm{ACS}_1}, \mathbf{M_s}^{\mathrm{ACS}_2}(V_j))$ is calculated, where $\mathbf{M_s}_{V_j}^{\mathrm{ACS}_1}$ is a transformation tensor stored at vertex V_j of ACS_1 and $\mathbf{M_s}^{\mathrm{ACS}_2}(V_j)$ is a transformation tensor calculated from ACS_2 at coordinates of V_j.
2. All control vertices from ACS_2 are inserted into ACS_1 as discrete point-sources, possibly modifying the structure of ACS_1.
3. Operation described in the first step is performed again.

If the second CS is not adaptive, the operation of CS structure adaptation for continuous metric source for ACS_1 is used instead.

3.2.4 Utilization

During the meshing process, each time the local metric has to be established at any given coordinates (within the domain), the proper value is retrieved from CS. Depending on the structure of CS, the operation of metric retrieving can be costly, so several optimizations were implemented:

1. *Caching parameterization matrix in control vertices.* The parameterization matrix, required during surface meshing, can be calculated directly from the parametric representation of the surface patch or it could be stored within the ACS during its preparation. Storing this matrix within the ACS allows to significantly increase the efficiency of the meshing process without visible degradation of the mesh quality. The parameterization matrix is stored in the vertices of the ACS structure together with the sizing component and is interpolated using the same method.

2. *Caching metric tensor in mesh points.* After setting some metric as locally constant, the transformed coordinates of some points may have be required more than once (e.g. for calculation of some geometric property for all incident elements). In order to avoid an unnecessary repetition of metric transformations (and creation of temporary data), these coordinates can be cached within the vertices. An additional metric context counter is used to decide, whether the cached transformed coordinates for some vertices can be used or if they should be recalculated (because the current metric has been changed in the meantime).

3. *Proximity check for control space calls.* If the coordinates of the inspected point are close enough to the point, where the metric was recently calculated, the current metric may be considered valid, and no further operation would be needed. The maximum distance between points, which can be called 'close enough' is calculated in the metric space (i.e. it depends on the required length of edges according to the current metric).

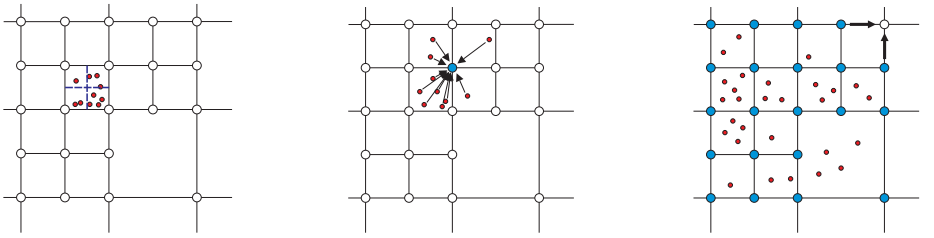
3.3 Quadtree/Octree Grid

A balanced quadtree/octree structure is used, with maximum difference of level for adjacent leaves equal to 1. The metric transformation tensor is stored in the vertices of tree nodes.

3.3.1 Introducing Discrete Data

Setting initial control space

This procedure is used in cases where a set of discrete point-sources with metric should define continuous field (e.g. metric calculated from the curvature of solution at the nodes of the mesh from a previous numerical adaptation step). It consists of the following three phases (Fig. 3.4):



(a) structure adaptation (b) initialization of vertices (c) metric extrapolation

Figure 3.4: Initializing quadtree CS structure using discrete data

1. *Preparing structure.* Each point-source with metric data is sift through the structure and is placed in a special list within the containing CS element (quadtree/octree leaf). The decision whether the current structure should be further refined depends on the following conditions:

   - *proximity of vertices* – if distance between the point-source Q and any already existing one V_i is too small (according to the metric of the new point-source Q: $d_{\mathcal{M}}(Q, V_i) < \epsilon_{\mathcal{M}}$), these point-sources are combined by calculating the metric intersection and skipping insertion of the new point,
   - *size sensitivity* – control element can be split if its size is too large compared with the size of mesh element described by the new metric (if $\max(l_{\mathcal{M}}(e_i)) > l_{\mathcal{M}\max}$ where e_i are the edges of control element and $l_{\mathcal{M}\max}$ is prescribed threshold value),
   - *metric difference* – control element can be split if the metric of stored point-sources is too different ($\max(\delta_{\mathcal{M}}(\mathbf{M_s}(Q), \mathbf{M_s}(P_i))) > \epsilon_\delta$).

   In case of splitting the control element all point-sources stored within are placed into proper sub-elements using the same procedure recursively.

2. *Initialization of control vertices.* The vertices of the ACS are initialized with the weighted average of metric of all point-sources from the adjacent leaves (Fig. 3.4(b)).

3. Finally, an iterative procedure of metric extrapolation for still uninitialized control vertices (having no source points with metric description in any adjacent element) is performed (Fig. 3.4(c)).

Updating control space

After the ACS is initialized it can be still adjusted by applying additional metric information at some points. For each point-source $(Q, \mathbf{M_s}_Q)$ the following procedure for structure adaptation is used:

1. Retrieve the current metric $\mathbf{M_s}_{\mathrm{ACS}}(Q)$ from the ACS.

2. If metric difference is very small (i.e. $\delta_{\mathcal{M}}(\mathbf{M_s}_Q, \mathbf{M_s}_{\mathrm{ACS}}(Q)) < \epsilon_\delta$) no adaptation is necessary.

3. Calculate new transformation tensor $\mathbf{M}_{\mathbf{s}Q}'$ as an intersection of metric tensors from the new point and from the current ACS: $\mathbf{M}_{\mathbf{s}Q}' = \min_{\mathcal{M}}(\mathbf{M}_{\mathbf{s}Q}, \mathbf{M}_{\mathbf{s}\,\mathrm{ACS}}(Q))$.

4. If the alteration would be very small (i.e. $\delta_{\mathcal{M}}(\mathbf{M}_{\mathbf{s}Q}', \mathbf{M}_{\mathbf{s}\,\mathrm{ACS}}(Q)) < \epsilon_\delta$) no adaptation is necessary.

5. Finally, if leaf edges E_i are sufficiently long ($\max(l_{\mathcal{M}}(E_i)) > l_{\mathcal{M}_{\max}}$) according to the new metric $\mathbf{M}_{\mathbf{s}Q}$ the leaf can be split (maintaining balanced tree property).

After the structure has been adapted the intersection operation for each control vertex V_i of the leaf containing the new point-source is used:

$$\mathbf{M}_{\mathbf{s}V_i} \leftarrow \min_{\mathcal{M}}(\mathbf{M}_{\mathbf{s}V_i}, \mathbf{M}_{\mathbf{s}Q}) \ .$$

3.3.2 Introducing Continuous Data

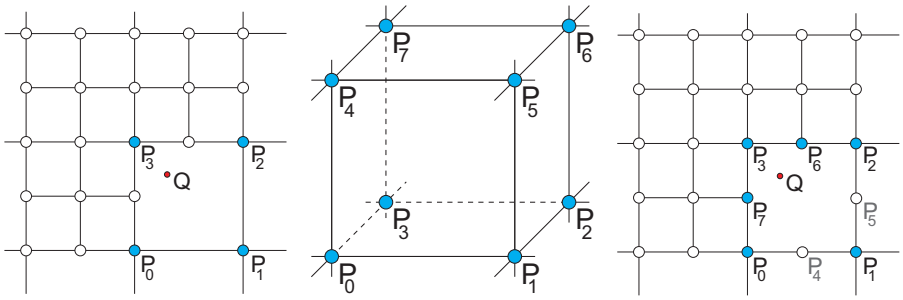
For continuous data (e.g. metric in analytic form or gathered from the curvature of the surface) all vertices of the current ACS structure are initialized with the available data (for already initialized nodes the intersection is used). Then for each leaf of the quadtree/octree structure two metric tensors are calculated at middle point (P_m) of the leaf: $\mathbf{M}_{\mathbf{s}}(P_m)$ comes directly from the continuous metric source and $\mathbf{M}_{\mathbf{s}\,\mathrm{ACS}}(P_m)$ is interpolated from the ACS. If difference between these two tensors is too large ($\delta_{\mathcal{M}}(\mathbf{M}_{\mathbf{s}}(P_m), \mathbf{M}_{\mathbf{s}\,\mathrm{ACS}}(P_m)) > \epsilon_\delta$) this leaf is split (with balancing) and the adaptation procedure is recursively called for all sub-leaves.

3.3.3 Adapting to Parameterization

For surface meshing an additional step is required – adaptation of the ACS structure to the parameterization field. The goal of this operation is not to modify the sizing field, but rather to refine the structure in a way which will allow to use it more efficiently during the meshing process. After this operation the parameterization matrix in each control element can be treated as sufficiently constant for operations like metric interpolation between control vertices which allow to perform such calculations directly, without any parameterization adjustments. Additionally, if caching of parameterization matrix in control vertices is enabled, this adaptation is necessary for proper interpolation of this matrix.

3.3.4 Smoothing

Global smoothing of ACS uses iteratively the operation of metric smoothing between two points. For quadtree/octree structure a recursive method of tree traversing is used consisting of two phases. In the first phase the tree is updated ‘upwards’, propagating the smoothing from the leaves of the tree up to the main level. In the second phase the reverse ‘downwards’ direction is used. For each inspected control leaf, pairs of vertices of all edges are checked and metric for these pairs of control vertices is smoothen accordingly.



(a) quadtree linear (b) octree linear (c) quadtree quadratic

Figure 3.5: Interpolation for quadtree/octree leaf

Utilization

The procedure starts with localizing the leaf containing the point for which the metric value is required. The metric is then calculated from the vertices of the rectangular leaf using linear interpolation (Fig. 3.5):

$$\mathbf{M_s}(Q) = \sum_{i=0}^{n} N_i(\mathbf{x})\mathbf{M_s}(P_i), \quad \mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_d), \tag{3.1}$$

where n is the number of nodes (4 for quadtree linear, 8 for octree linear or 8 for quadtree quadratic 'serendipity' interpolation) and $N_i(\mathbf{x})$ are the shape functions. If quadtree quadratic interpolation is used and not all middle vertices are available, interpolated metric from edge vertices is used instead.

3.4 Background Mesh

The background mesh structure is stored and managed in a way similar to the actual mesh being generated, using the same set of meshing procedures (in uniform Euclidean metric).

3.4.1 Introducing Discrete Data

Setting initial control space

Initializing the background mesh ACS structure consists of the following steps:

1. *Inserting point-sources.* Points with metric being inserted into the background mesh become the control vertices of this structure. An additional check is performed in order to avoid placing new vertices too close to already existing ones. If distance in metric space (according to $\mathbf{M_s}_Q$) of the new vertex Q to one of the other vertices P_i is smaller than some threshold ($d_{\mathcal{M}}(Q, P_i) < \epsilon_{\mathcal{M}}$), the metric at P_i is intersected with the metric of vertex Q and the insertion is cancelled.

2. *Triangulation of set of control vertices.* The triangulation method used here is identical with the one used for creation of the actual mesh, with metric set to identity.

Since in some cases the given set of control vertices can be distributed in the domain in an irregular manner, an additional refinement procedure can be used. The goal of this operation is to increase the overall geometrical quality of the control mesh, which usually helps to locate the containing triangle/tetrahedra of the inspected point in a possibly short time.

3. *Classification of the interpolation type.* Optionally, should different interpolation methods be used during the generation, all control triangles have to be classified according to the required interpolation type [125, 126].

Updating control space

The procedure for updating the ACS structure is very similar to the quadtree/octree case described in Sec. 3.3.1. The only difference is the split operation, which in case of mesh is replaced by actually inserting the new vertex into the control mesh with local retriangulation (and reclassification if required).

3.4.2 Introducing Continuous Data

For adapting to continuous data a regular set of point is created for initial mesh, with metric calculated directly from the source. Then, for all triangles/tetrahedra the middle point is checked for quality of metric interpolation (i.e. whether metric interpolated from the element and calculated directly from source are close enough). If necessary, additional points are inserted in the circumsphere of elements with local retriangulation and the procedure is repeated for all new elements.

3.4.3 Adapting to Parameterization

This operation is performed in a similar way as in the quadtree/octree case described in Sec. 3.3.1.

3.4.4 Smoothing

For global smoothing of background mesh structure the connectivity between mesh vertices is used. First, all mesh edges are marked as unchecked. Then the iterative procedure calls smoothing operation for control vertices of each edge, marking it as checked. The smoothing operation of a pair of metric source-vertices (described in Sec. 2.3.4) may adjust any or both of these vertices. If any control vertex has its metric value changed, all adjacent edges are again marked as unchecked. Finally, procedure converges to a state, where all edges are checked and there are no further changes in control vertices, which ends the global smoothing operation.

3.4.5 Utilization

The interpolation procedure starts with finding the triangle (or tetrahedron) of the background mesh which contains the given point Q. Traversing the triangles/tetrahedra in

the mesh in the direction of the given point is used, with an additional quadtree/octree structure (built during the triangulation step) for selection of good starting element.

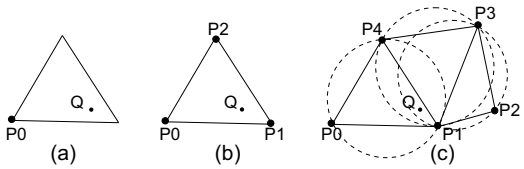


Figure 3.6: Selection of vertices for metric interpolation in background mesh: (a) $\mathcal{A}_s$ – single vertex of the containing triangle, (b) $\mathcal{A}_t$ – all vertices of the containing triangle, and (c) $\mathcal{A}_o$ – all vertices of all triangles containing inspected point Q in theirs circumcircles

The set $\mathcal{A}_*$ of interpolation vertices is selected according to one of the following methods (Fig. 3.6):

$\mathcal{A}_s$ – an arbitrary vertex of the triangle/tetrahedron is selected and the metric from this vertex is taken directly (most efficient, but applicable only in areas of near to constant metric description),

$\mathcal{A}_t$ – all vertices of the selected triangle/tetrahedron are used for interpolation,

$\mathcal{A}_o$ – (*natural neighborhood*) the set consists of all vertices of the containing element and all vertices of other triangles/tetrahedra which also have the point Q in theirs circumspheres.

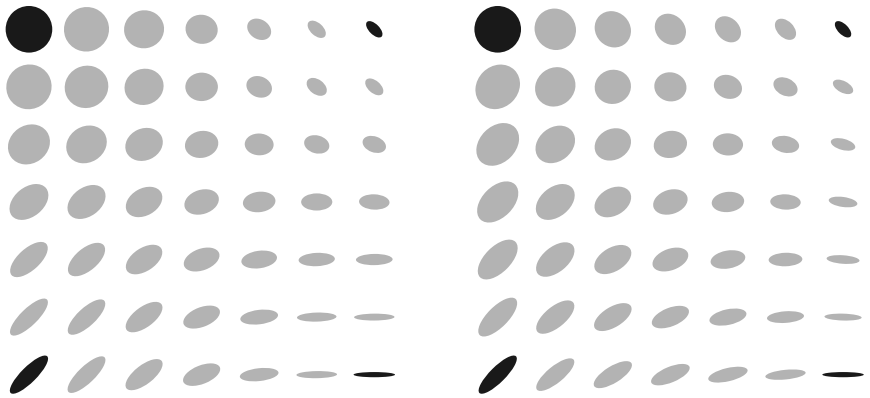
One of these methods can be selected arbitrarily for the whole control space, or an automated procedure of classification can be used[126]. From this set of vertices the resulting metric is calculated using the weighted average formula:

$$\mathbf{M_s}(Q) = \frac{1}{\sum_{P_i \in \mathcal{A}_*} \omega_i} \sum_{P_i \in \mathcal{A}_*} \mathbf{M_s}_{P_i} \omega_i \tag{3.2}$$

where $\omega_i = d_{\mathcal{R}}(Q, P_i)^{-2}$ is the inverse squared-distance weight, calculated from points coordinates in transformed space $\mathcal{S}^{\mathcal{R}}$ (for surface meshing where control space is adapted for parameterization variance, the distance in parametric space is used directly). If $d_{\mathcal{M}}(Q, P_i) < \epsilon_{\mathcal{M}}$, the metric from the point P_i is taken directly, skipping the interpolation.

3.4.6 Comparison of Metric Interpolation Methods

Figure 3.7 shows the results of metric interpolation in the background mesh ACS structure (using shape functions) compared to the results obtained in the quadtree ACS (using inverse squared-distance weighted average). Black ellipses denote known metric in four vertices of a rectangular domain, gray ones represent the interpolated metric between them. Although some small differences can be noticed, both methods provide very similar results.



(a) weighted average (background mesh) (b) shape functions (quadtree)

Figure 3.7: Interpolation of metrics using different methods

3.5 Automated Sizing

3.5.1 Adaptive Surface Meshing

Algorithm 3.1 shows subsequent steps of automated creation and actualization of the ACS structure. The domain being discretized is assumed to consist of one or more surface patches (possibly connected), where each surface provides a parametric representation. Each patch is triangulated separately in its parametric space. A special care must be taken for assuring smooth variation of elements over adjacent patches. The procedure iteratively checks and adjust ACS in all patches until required conditions are fulfilled (typically up to several steps).

```
1  foreach surface patch p do  create initial ACS_p
2  mark all 3D-contours c as invalid
3  while ∃ invalid c or ∃ invalid p do
4      foreach invalid c do
5          mark all p adjacent to c as invalid
6      foreach invalid p do
7          update ACS_p with length of discretization segments of c
8          I step of patch discretization – triangulation of boundary nodes
9          mark p as valid-I
10         update ACS_p with proximity of boundary segments
11         foreach adjacent c do
12             if c discretization too coarse for ACS_p then  mark c as invalid
13 foreach not valid-II p do
14     II step of patch discretization – inner nodes insertion, smoothing, etc.
15     mark p as valid-II
```

Algorithm 3.1: Preparation of ACS for surface mesh generation

Creating initial control space

```
1 calculate bounding rectangle
2 create new ACS
3 initialize ACS with surface curvature
4 foreach available user CS (2D or 3D) do
5 |  intersect current ACS with user CS
6 update with contour curvature
7 adapt to parameterization
8 smoothen
```

Algorithm 3.2: Creation of initial ACS

Algorithm 3.2 presents steps of initial creation of ACS for meshing of surface patch. During this phase there are introduced all metric sources which are available directly.

The surface curvature source provides the metric information throughout the whole surface patch. Using the procedure described in Sec. 2.4.2, the metric transformation tensor is calculated and stored in the vertices of ACS.

The curvature of contours allows to gather the metric information along the contours. The contour is probed at discrete intervals and the calculated metric data (Sec. 2.4.3) is inserted into ACS as a set of discrete metric sources.

User control space While the automatic recognition of mesh sizing in most cases produces meshes of acceptable quality, there are sometimes situations, where some adjustment from user is necessary in order to include information about the desired structure of the mesh which is not directly possible to obtain from the geometric description of the discretized domain.

In order to facilitate such interaction, a number of possibilities has been proposed. The metric description can be introduced through analytical formulation (in a selected sub-domain or in the whole domain), in discrete points or using an extended form.

Another example of the metric data, which can be used in created generator, is the information obtained from the numerical solver in the adaptation process[125]. The metric information is gathered from the vertices of the current computational mesh, defining the metric in discrete points.

Depending on the type of the user metric data, it is introduced directly into initial ACS, or a separate CS is created and the superposition of two CS is used.

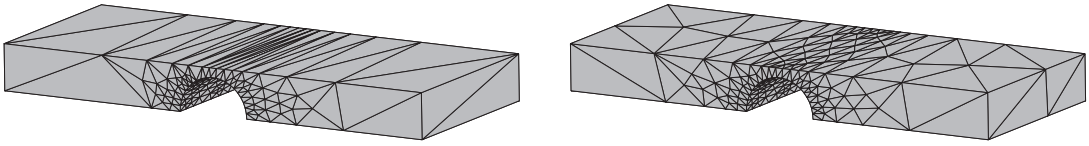
Updating control space

After discretization of contours and initial triangulation of boundary vertices for the given surface patch, additional sizing information becomes available and can be used to check and adjust the ACS:

- *Length of boundary segments.* Length $l_{\mathcal{M}}(E_b)$ of each boundary edge E_b is calculated according to local metric space. If edge it too long ($l_{\mathcal{M}}(E_b) > l_{\mathcal{M}_{\max}}$) the contour

containing this edge is marked as *invalid* and should be discretized again. If edge is too short ($l_{\mathcal{M}}(E_b) < 1/l_{\mathcal{M}_{\max}}$) the contour is marked as *invalid* and additionally the ACS in the neighborhood is updated accordingly (using the length of this segment).

- *Proper representation of curvilinear boundaries.* Distance between the mesh segment and the underlying curvilinear contour is measured, and in case of too rough discretization the ACS in this area is accordingly updated (using curvature of the contour directly) and the whole contour is marked as *invalid*.
- *Vicinity of boundaries.* Triangles of the initial triangulation with too short height h_t from one of its boundary edge ($l_{\mathcal{M}}(h_t) < 1/l_{\mathcal{M}_{\max}}$) are inspected and the ACS is appropriately updated if necessary.



(a) without cross-patch updating (b) with cross-patch updating

Figure 3.8: Effect of cross-patch control space updating

Figure 3.8 shows an example multi-patch (closed) domain meshed with and without additional ACS update during meshing process.

3.5.2 Volume Meshing

For volume mesh generation the automated sizing algorithm is extended by implementing an additional loop (Alg. 3.3) necessary for achieving the proper characteristic of 3D mesh elements generated between boundary surface meshes [127].

Creating initial control space

Algorithm 3.4 presents steps of initial creation of ACS for meshing of surface patch. The initial ACS for each block is created basing on all available three-dimensional user metric sources and the size of elements from the boundary surface meshes.

Control spaces from boundary patches The three-dimensional metric from adjacent surface discretizations can be gathered either by retrieving metric directly from the ACS of each surface patch (with an additional adjustment for surface parameterization) or by calculating metric from each mesh triangle (with local averaging procedure). In either case, the two-dimensional metric in eigensystem form $(e_1, e_2, \lambda_1, \lambda_2)$ can be extended to 3D by adding an additional eigenvector e_3 (orthogonal to both e_1 and e_2) and eigenvalue λ_3 calculated as:

$$\lambda_3 = \gamma_a \min(\lambda_1, \lambda_2) \ , \tag{3.3}$$

where γ_a is the maximum anisotropy ratio.

```
1  while ∃ invalid surface patch p do
2      foreach invalid p do  discretize p using Alg. 3.1
3      foreach invalid volume block v do
4          gather discretization of adjacent p into 3D boundary mesh
5          if not ∃ACS_v then  create initial ACS_v
6          foreach adjacent p do
7              if discretization of p is too coarse for ACS_v then
8                  update ACS_p
9                  mark p as invalid
10     foreach invalid v with no invalid adjacent p do
11         I step of v discretization (triangulation of boundary nodes)
12         mark v as valid
13         update ACS_v with proximity of boundary faces
14         foreach incident p do
15             if discretization of p is too coarse for ACS_v then
16                 update ACS_p
17                 mark p as invalid
18                 mark v as invalid
19 foreach v do  II step of discretization – inner nodes insertion, smoothing, etc.
```

Algorithm 3.3: Preparation of ACS for volume mesh generation

```
1 calculate bounding box
2 create new ACS
3 foreach available user CS (3D) do
4     update current ACS with user CS
5 add metric from surface patches CS
6 smoothen
```

Algorithm 3.4: Creation of initial ACS

Updating control space

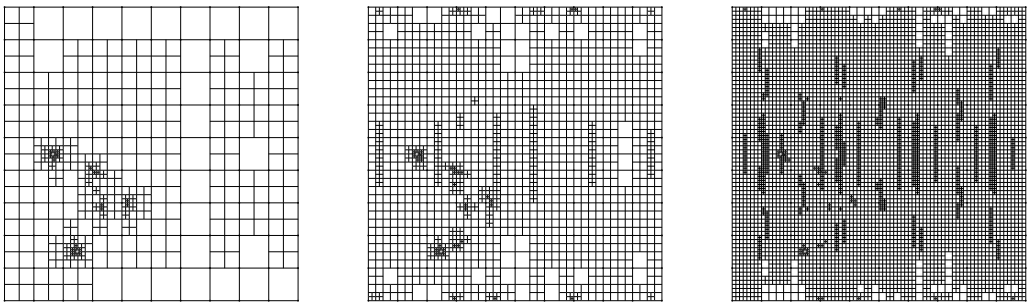
After the first step of volume block discretization (i.e. triangulation of boundary nodes) is performed, the proximity of volume boundary faces can be checked, which may trigger rediscretization of one or more boundary surface meshes.

3.5.3 Parameters for Automated Mesh Sizing

The process of automated meshing is influenced by a number of metric sizing parameters. Some parameters can be used to directly control general characteristic of the mesh (e.g. maximum anisotropy ratio or gradation of mesh element sizes). Other parameters may influence the precision of the adaptation of ACS structure, which also has effect on the resulting mesh[128] and the efficiency of the control space initialization process[129].

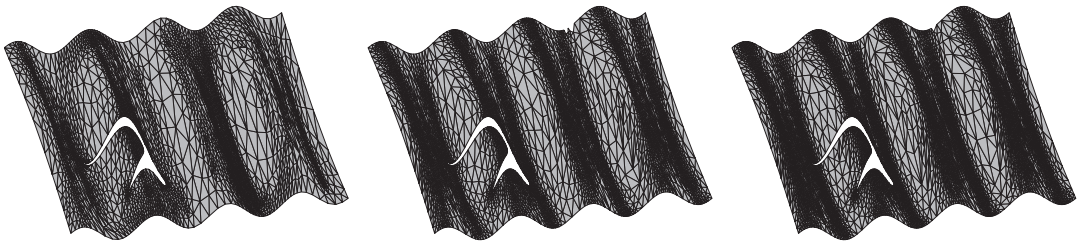
Metric difference threshold

The *metric difference threshold* ϵ_δ is an example of a parameter influencing the structure of the ACS and the created finite element mesh. Decreasing the value of this parameter typically results in more refined structure of ACS (Fig. 3.9) which allow to enhance the precision of fitting the created mesh to the modeled domain (Fig. 3.10). In the presented example (with quadtree ACS structure) increasing the value of this coefficient results in lowering of the resolution of the control grid structure, which in consequence lowers the number of elements in the final mesh.



(a) $\epsilon_\delta = 10.0$ ($\mathrm{NP}_c = 875$) (b) $\epsilon_\delta = 1.0$ ($\mathrm{NP}_c = 2306$) (c) $\epsilon_\delta = 0.1$ ($\mathrm{NP}_c = 7914$)

Figure 3.9: Influence of metric difference threshold on quadtree ACS structure (NP_c – number of control vertices)

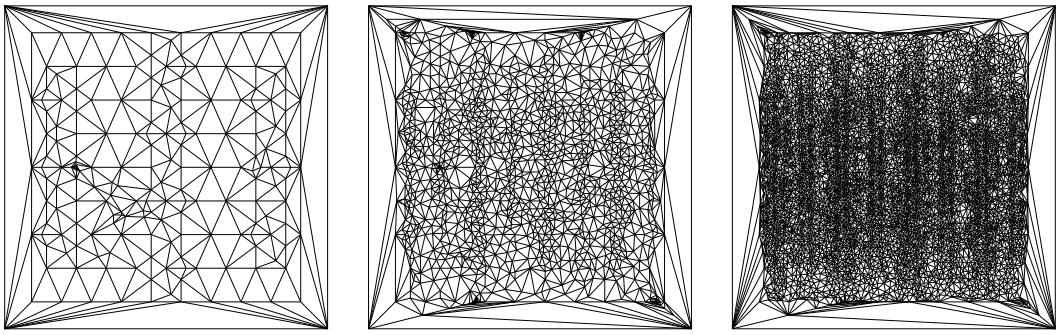


(a) $\epsilon_\delta = 10.0$ (NT = 4829) (b) $\epsilon_\delta = 1.0$ (NT = 8189) (c) $\epsilon_\delta = 0.1$ (NT = 11177)

Figure 3.10: Influence of metric difference threshold on created mesh (NT – number of mesh elements in the final mesh)

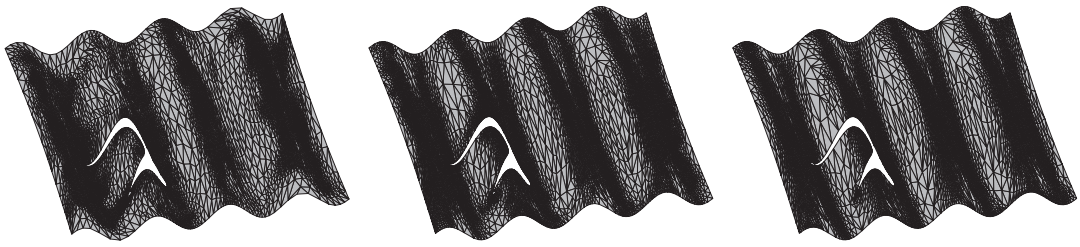
If the background mesh ACS structure is used in the same example (Fig. 3.11 and 3.12) the created meshes have a similar number of elements. Insufficient adaptation of the ACS structure has resulted in this case in inappropriate sizing (either too dense or too sparse) in various sub-domains of the discretized domain.

In both cases decreasing the *metric difference threshold* has resulted in better adaptation of the mesh to the underlying geometry. However, the time required for preparation of a more refined ACS (containing more control vertices) increases, which reduces the efficiency of the whole meshing process (Tab. 3.1).



(a) $\epsilon_\delta = 10.0$ ($\mathrm{NP}_c = 182$) (b) $\epsilon_\delta = 1.0$ ($\mathrm{NP}_c = 1208$) (c) $\epsilon_\delta = 0.1$ ($\mathrm{NP}_c = 7164$)

Figure 3.11: Influence of metric difference threshold on background mesh ACS structure (NP_c – number of control vertices)



(a) $\epsilon_\delta = 10.0$ (NT = 13998) (b) $\epsilon_\delta = 1.0$ (NT = 15832) (c) $\epsilon_\delta = 0.1$ (NT = 14231)

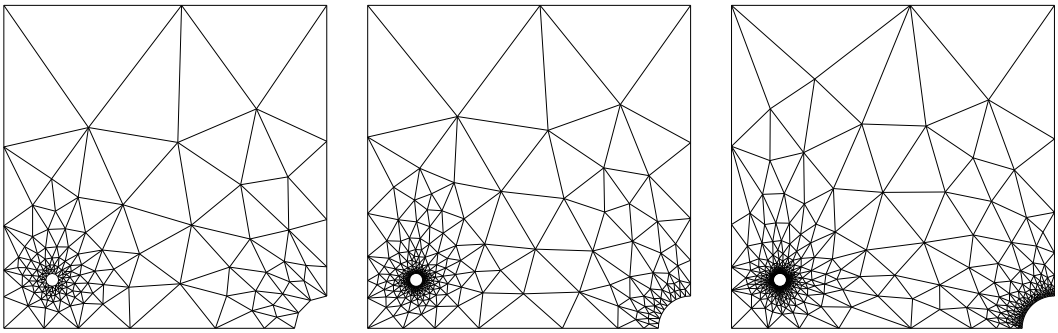
Figure 3.12: Influence of metric difference threshold on created mesh (NT – number of mesh elements in the final mesh)

Table 3.1: Mesh generation with different values of metric difference threshold, ϵ_δ – metric difference threshold, NP_c – number of vertices in ACS, NP and NT – number of points and triangles in the resulting mesh, NT/s – total mesh generation speed (triangles per sec.)

	quadtree				background mesh			
ϵ_δ	NP_c	NP	NT	NT/s	NP_c	NP	NT	NT/s
10.0	875	2568	4829	14038	182	7192	13998	11057
1.0	2306	4274	8189	11142	1208	8149	15832	6801
0.1	7914	5793	11177	5634	7164	7359	14231	2958

Curvature ratio

Figure 3.13 presents three planar meshes generated for an example rectangular domain with different values of curvature ratio coefficient. As can be seen, by adjusting this parameter the precision of discretization of contours (and surfaces) in the modeled domain can be easily controlled.

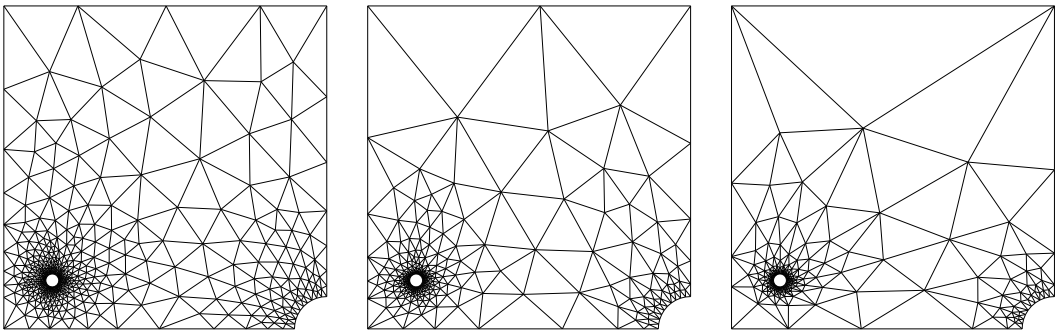


(a) $\gamma_c = 0.45$ (b) $\gamma_c = 0.15$ (c) $\gamma_c = 0.05$

Figure 3.13: Influence of curvature sizing ratio (planar mesh)

Metric gradation

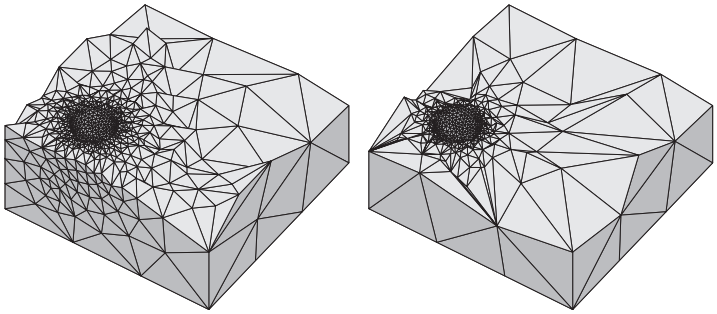
Figure 3.14 presents meshes of an example rectangular domain meshed with different value of metric gradation. For $\gamma_{\mathcal{M}} = 1$ the resultant mesh would have a constant size of all elements in the mesh. Increasing the value of this coefficient allows to create meshes with higher gradation, as larger differences between adjacent elements are possible.



(a) $\gamma_{\mathcal{M}} = 2$ (b) $\gamma_{\mathcal{M}} = 4$ (c) $\gamma_{\mathcal{M}} = 100$

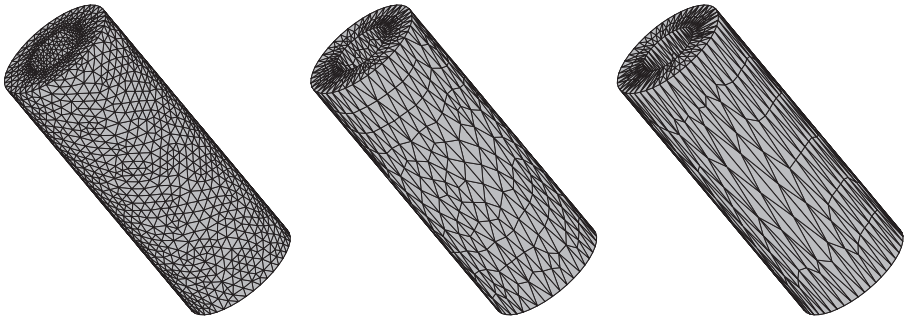
Figure 3.14: Influence of metric gradation ratio (planar mesh)

Influence of the mesh gradation ratio on volume meshes is depicted on Fig. 3.15. Both meshes were generated for a hexahedral domain containing a spherical hole and a cross-section of the volume mesh is shown.



(a) $\gamma_{\mathcal{M}} = 2$ (b) $\gamma_{\mathcal{M}} = 5$

Figure 3.15: Influence of metric gradation ratio (volume mesh)



(a) $\gamma_a = 1$ (b) $\gamma_a = 3$ (c) $\gamma_a = 10$

Figure 3.16: Influence of maximum anisotropy ratio (surface mesh)

Anisotropy ratio

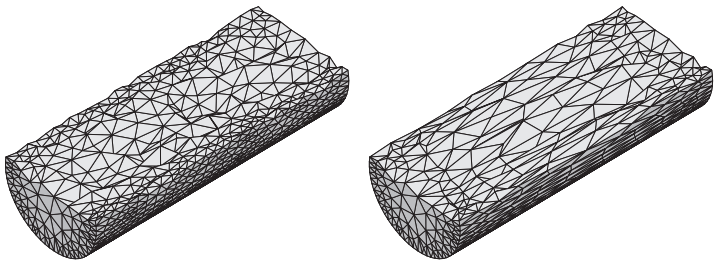
While anisotropic elements allow to better adapt the mesh to specific problem, there are also cases where the anisotropic shape of elements is not desirable. The maximum anisotropy ratio γ_a allows to control the maximum stretching of elements in the mesh. For $\gamma_a = 1$ an isotropic mesh can be produced. The anisotropy ratio coefficient is applied for all discrete forms of metric definition, its influence on surface mesh of a cylinder is shown on Fig. 3.16.

Figure 3.17 presents cross-sections of volume meshes created with different values of the maximum anisotropy ratio γ_a.

In order to obtain even larger flexibility of the mesh generator, the parameters can be defined separately for two- and three-dimensional generation procedures (Fig. 3.18).

Summary

Table 3.2 presents the most important parameters which were used in the described mesh generator. The default values included in this list were selected in a result of a number of experiments and cooperation with users of the implemented generator software.



(a) $\gamma_a = 2$ (b) $\gamma_a = 10$

Figure 3.17: Influence of maximum anisotropy ratio (volume mesh)

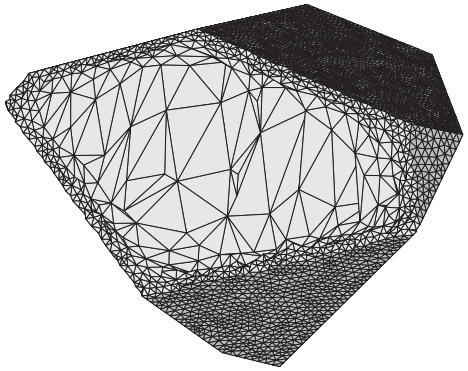


Figure 3.18: Different maximum size ratio for surface and volume mesh generation

Table 3.2: Main parameters influencing process of automated mesh sizing

parameter	default	description
γ_a	5.0	anisotropy ratio
$\gamma_{\mathcal{M}}$	4.0	metric gradation ratio
γ_c	0.15	curvature sizing ratio
γ_{a_c}	2.0	anisotropy ratio for contour curvature
$\gamma_{h_{\max}}$	0.5	maximum length ratio
$\gamma_{h_{\min}}$	10^{-4}	minimum length ratio
$\epsilon_{\mathcal{M}}$	10^{-5}	threshold for distance in metric space
ϵ_δ	0.2	threshold for metric difference
$l_{\mathcal{M}_{\max}}$	2.0	metric length threshold for adaptation

3.6 Chapter Summary

In this chapter the concept of a control space structure, guiding the meshing process, was discussed. An object-oriented specification of an abstract adaptive control space interface was proposed. The ACS interface contains a set of operations necessary for storing and management of a discrete metric data. Implementations of two different structures: quadtree/octree grid and background mesh were presented as examples. A number of optimization for control space utilization were proposed for both the general ACS interface and the specific implementations.

An iterative procedure of automated construction of control space for both surface and volume meshing has been developed and described. The additional information which becomes available at the successive steps of mesh generations is used to adaptively refine the sizing field as required.

A number of global parameters have been supplied for controlling the process of ACS initialization and utilization during meshing. The selection of most important parameters has been listed and theirs influence on the created meshes has been shown in a number of examples for both two- and three-dimensional meshing problems.

Chapter 4

Anisotropic Mesh Generation on Parametric Surfaces

4.1 Introduction

The mesh generation on three-dimensional surfaces may be treated as a complete task required for surface simulations or visualization. It can be also seen as an intermediate step for three-dimensional (hierarchical) mesh generation, where surface mesh defines the boundary for volume meshing.

4.1.1 Related Research

Delaunay triangulation

The Delaunay triangulation (DT), dual to Voronoï tessellation, defines the connection of a cloud of points in space into a triangulation. A DT in d dimensions is a unique triangulation (save for a degenerated case with $d+1$ co-spherical points) constructed on a set of points such that a circumsphere of any simplex does not contain any other point from this set. Detailed description of Delaunay based techniques (and related problems) can be found in books by George and Borouchaki[27], and by Frey and George [28].

The construction of a DT (using incremental insertion algorithm in d dimensions) was first proposed by Bowyer[18] and Watson[19], and further explored by Lawson[20], Hecht and George[24], Hermeline[130], and Joe[131].

There were also proposed a number of algorithms for generation of the DT of a convex hull of points, employing techniques like divide and conquer, sweeping, gift wrapping, or higher dimensional embedding. Su and Drysdale[21] present a survey and an empirical comparison of these five categories of DT algorithms.

Since the DT algorithm itself creates only a convex hull triangulation of a given set of points, some additional operations are necessary for calculation of this set, imposing the boundary and generation of nodes within the meshed domain[24]. The typical approach is to discretize the domain boundary to produce an initial set of mesh points. This set is

then triangulated using Delaunay technique and constrained by recovering the boundary and removing obsolete elements from outside the domain. This mesh is further refined by incremental insertion of inner nodes with local retriangulation also using the Delaunay criterion.

In two dimensions the constraining to boundary operation is simple and can be achieved by iterative swapping of edges[24] or insertion of additional points in the missing edges.

The inner nodes are usually inserted at simplex circumcenters [132–134] which in two-dimensional case allows to obtain a minimum bound on any angle in the mesh (with exception of small angles introduced by domain boundary[135]), if points are inserted in a special order. The points can be also inserted in simplex centroids[25], using the Voronoï segments[136, 137] or mesh edges[138, 139], they can be also generated with an advancing front approach[140, 141].

Studies of generating isotropic triangular meshes with *shape* guarantees (e.g. bounded aspect ratio or minimum angle) can be found in [142–150]. Efficiency aspects of meshing procedure with Delaunay insertion algorithm and appropriate data structures together with numerical considerations are reviewed in [151–154].

Conversion to quadrilateral mesh

The quadrilateral mesh can be generated directly basing on the geometric description of surface patch only, using techniques like decomposition and mapped meshing[155], medial-axis with template operators[156] or paving[38]. An alternative approach, the indirect generation, utilizes the triangular mesh as a reference and the quadrilateral mesh is created by merging or splitting triangles. The order of triangles selection and the method of forming new quadrilaterals differentiates various indirect algorithms. An advancing front approach with systematic joining of triangle pairs was proposed by Lee and Lo[40, 41]. In Q-Morph developed by Owen[42], the frontal approach is also used, but in the forming of new quadrilaterals some techniques from the paving method is applied, creating layers of quadrilateral elements. In some works[43–45] a concept of circle/ellipse packing is used to create a mixed quad-dominant mesh, which can be transformed to an all-quadrilateral mesh by splitting all mesh triangles and quadrilaterals.

Mesh improvement

The goal of mesh improvement techniques is to increase the quality of the mesh, according to some mesh quality indicator[68, 70, 71, 73, 74, 76]. A number of methods have been proposed in the literature, either as a stand-alone procedures or as a part of some mesh generation algorithms. The improvement methods may be topological (e.g. swapping of mesh edges) or geometrical (repositioning of mesh vertices). Usually several types of mesh improvement techniques are used alternately in order to achieve best results.

The most commonly used mesh improvement method is the Laplacian smoothing based on iterative moving each vertex to barycenters of all incident vertices, which has been described in a number of variations[79]. The mesh vertices may be also relocated basing on inner angles[53] or other criteria[49]. The most popular topological method is swapping

of edges[47] guided by selected quality coefficient of adjacent elements. The optimization techniques, with large potential but also with high computational costs, usually apply variational[80], line search[81] or physical based methods[82].

4.1.2 Contents

In this chapter there is described the process of generation of triangular and quadrilateral meshes on three-dimensional surfaces (Fig. 4.1) [157].

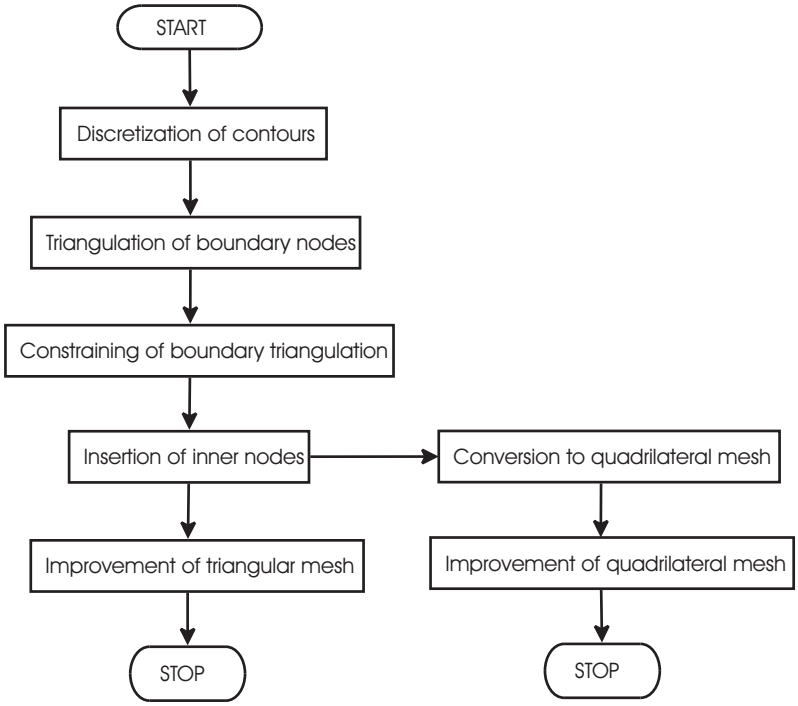


Figure 4.1: Overview of mesh generation process on three-dimensional surfaces

The triangular mesh generation is based on the incremental refinement triangulation based on Delaunay criteria [18, 19]. There is presented the implementation of this technique for the proposed system of anisotropic metric introduction by coordinate transformation (Sec. 4.2). The successive phases of mesh generation: discretization of contours (Sec. 4.3), triangulation of boundary vertices with constraining (Sec. 4.4), insertion of inner nodes (Sec. 4.5) and mesh improvement (Sec. 4.6) are inspected and adjusted for anisotropic metric introduction with respect to efficiency of meshing process and quality of the obtained discretizations.

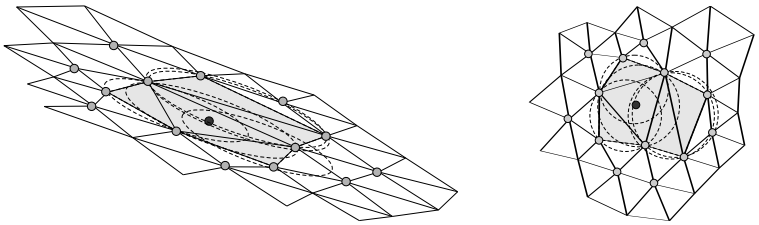
For quadrilateral mesh generation the implementations of two indirect conversion methods for metric transformation system are presented (Sec. 4.7). A new mixed approach is proposed utilizing concepts from both methods. The quality of created meshes can be enhanced with some implemented quadrilateral mesh improvement methods (Sec. 4.8).

4.2 Incremental Retriangulation

The essential operation for the chosen method of triangular mesh generation is the mesh retriangulation procedure restoring the Delaunay property. With introduction of variable anisotropic metric, this property is no longer global and the retriangulation procedures have to be properly adjusted. In this work both *empty circumcircle* (EC) and *inner angles* (IA) criteria have been successfully adapted to work with metric introduced via the technique of coordinate transformation [158, 159]. Both procedures run in the parametric space of the given patch, like all other algorithms used during the discretization of parametric surfaces.

4.2.1 Criterion of Empty Circumcircle

The introduction of new node into the existing mesh can be accomplished using the empty circumcircle criterion, which requires locating and removing of each triangle having this node in its circumcircle. Procedure starts with triangle containing the inserted node. Adjacent triangles are iteratively checked with *in-circumcircle* test. Selected triangles are removed, and the vertices of the created empty (guaranteed to be star shaped) cavity are connected with the new node, forming new triangles (Fig. 4.2).



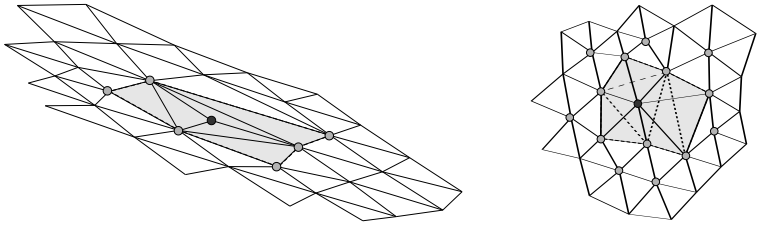
(a) parametric space $\mathcal{S}$ (b) transformed space $\mathcal{S}^{\mathcal{M}}$

Figure 4.2: Insertion of new node using empty circumcircle criterion

For each inserted node local metric is retrieved from CS once and metric is assumed as locally constant for all checked triangles. The coordinates of new node and vertices of checked triangles have to be transformed into metric space. If this procedure is used during the phase of inner nodes insertion, the quality of all new triangles is recalculated, which requires additional calls to CS and transformation of vertices.

4.2.2 Criterion of Inner Angles

Using the inner angles criterion[28] the procedure starts with locating the containing triangle, which is then removed and substituted by three new triangles (Fig. 4.3). If the node is located at the edge, two triangles have to be removed and four new elements are inserted instead. After insertion, the edges in the adjacent elements are swapped with respect to inner angles criterion. The swapping procedure starts with checking the new triangles and theirs neighbors for fulfillment of the criterion. For each swapped edge, the new triangles and theirs neighbors have to be iteratively checked.



(a) parametric space $\mathcal{S}$ (b) transformed space $\mathcal{S}^{\mathcal{M}}$

Figure 4.3: Insertion of new node using inner angles criterion

For each inserted node local metric is retrieved from CS once. The coordinates of new node and vertices of each pair of adjacent triangles being checked for inner angles test have to be transformed into metric space. As in the empty circumcircle criterion, if this procedure is used during the phase of inner nodes insertion, the quality of all modified triangles is recalculated, which requires additional calls to CS and transformation of vertices.

In the classic approach the iterative checking and swapping of diagonals for pairs of incident triangles is carried on until all triangles in the mesh fulfill the criterion of inner angles. For meshing with variable anisotropic metric this procedure has to be modified. The assumption of invariability of the metric has a limited range and the inner angles criterion checking should not be used for triangles lying too far from the inserted point. The allowed range of checking could be approximated basing on the local metric and its gradation (which is also stored in ACS). Another, more simple and efficient solution used in this work was to limit the maximum number of diagonal swaps per single point insertion. Table 4.1 presents example statistics* of mesh generation using criterion of inner angles with limited number of allowed diagonal swaps per point inserted into triangulation. As this example shows, introducing such limit has no undesirable effect on either the quality of the mesh nor the meshing time.

Table 4.1: Limiting number of diagonal swaps for retriangulation with inner-angles criterion (SURF0B), $N_{\mathrm{d}}^{\max}$ – swaps limit, $\overline{N}_{\mathrm{ds}}^{B}$ and $\overline{N}_{\mathrm{ds}}^{I}$ – average number of diagonals swaps during boundary and inner nodes triangulation, τ_P – triangulation speed (NP/s)

$N_{\mathrm{d}}^{\max}$	NP $[10^3]$	$\overline{N}_{\mathrm{ds}}^{B}$	$\overline{N}_{\mathrm{ds}}^{I}$	$\overline{\eta}_{\mathcal{M}}$	$\sigma_{\eta_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$\overline{\mathcal{E}}_K$	$\tau_P[10^3/s]$
∞	255.3	7.4	2.4	0.973	0.035	1.026	0.151	1.954	14.9
1000	255.3	7.4	2.4	0.973	0.035	1.026	0.151	2.071	14.9
100	255.5	5.6	2.3	0.973	0.034	1.026	0.151	2.540	15.0
50	255.4	5.2	2.3	0.972	0.035	1.026	0.151	1.697	15.1
10	255.4	2.8	2.3	0.972	0.035	1.026	0.151	2.005	15.0

Instead of assuming constant metric during the whole phase of retriangulation, the metric could be also calculated independently for each pair of triangles being checked. This

*The set of mesh quality coefficients ($\overline{\eta}_{\mathcal{M}}$, $\sigma_{\eta_{\mathcal{M}}}$, $\overline{\mu}_{\mathcal{M}}$, $\sigma_{\mu_{\mathcal{M}}}$, and $\overline{\mathcal{E}}_K$) used in this work is defined in Sec. 2.5 and included in the list of symbols and abbreviations. All presented running times of implemented procedures were measured with a single Dual Core AMD Opteron (2x2GHz) machine with 4GB memory, unless explicitly stated otherwise.

approach would allow to achieve better matching of elements with respect to the metric field. However, it would also introduce an additional cost of many CS calls. Also, in some cases the swapping loop might have problems with converging (series of infinitely repeating swaps are possible). In the selected approach this more local variation of mesh retriangulation using inner angles criterion is used in the final steps of triangulation only, during the mesh improvement phase.

4.2.3 Comparison of Delaunay Criteria

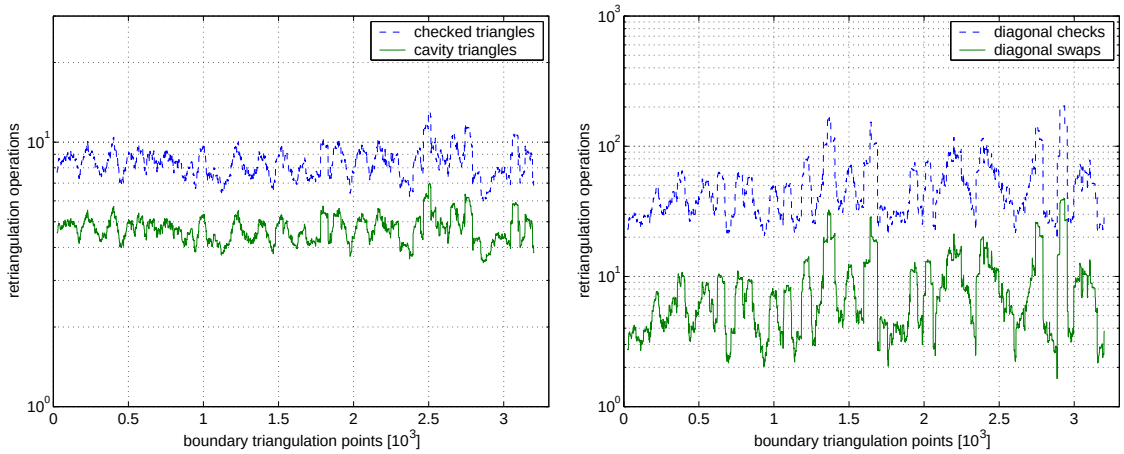
Both criteria of Delaunay retriangulation were implemented for used coordinate transformation system and their efficiency and suitability for anisotropic meshing with metric was evaluated. Table 4.2 shows statistics for triangulation of meshes AN1 (Fig. A.2), SURF0C (Fig. A.4) and SURF1 (Fig. A.5) with both empty circumcircle and inner angles criteria.

Table 4.2: Metric utilization for surface triangulation, NP – number of points, $\overline{N}_{\mathrm{CS}}$ – number of CS calls, $\overline{N}_{\mathrm{m}}$ and $\overline{N}_{\mathrm{p}}$ – average number of coordinate transformations to and from metric space per inserted point, $\overline{N}_{\mathrm{cc}}$ and $\overline{N}_{\mathrm{ct}}$ – average number of in-circle checks and cavity triangles, $\overline{N}_{\mathrm{dc}}$ and $\overline{N}_{\mathrm{ds}}$ – average number of diagonal checks and swaps

mesh	NP	$\overline{N}_{\mathrm{CS}}$	$\overline{N}_{\mathrm{m}}$	$\overline{N}_{\mathrm{p}}$	$\overline{N}_{\mathrm{cc}}$	$\overline{N}_{\mathrm{ct}}$	$\overline{N}_{\mathrm{dc}}$	$\overline{N}_{\mathrm{ds}}$
boundary nodes triangulation								
AN1	9803	1.0	13.7	0.0	6.7	3.8		
SURF0C	3225	1.0	17.7	0.0	8.4	4.7		
SURF1	52	1.0	13.9	0.0	6.3	3.8		
AN1	9803	1.0	8.9	0.0			33.7	4.3
SURF0C	3225	1.0	11.6	0.0			51.1	8.1
SURF1	52	1.0	8.9	0.0			18.4	1.6
inner nodes triangulation								
AN1	97998	15.3	56.0	1.1	8.1	5.0		
SURF0C	251908	14.3	51.4	1.0	7.4	4.3		
SURF1	10094	14.2	53.8	1.0	7.8	4.4		
AN1	98091	26.8	34.5	1.1			25.0	3.6
SURF0C	252028	14.2	45.0	1.0			20.7	2.4
SURF1	10111	15.3	48.9	1.0			23.9	3.2

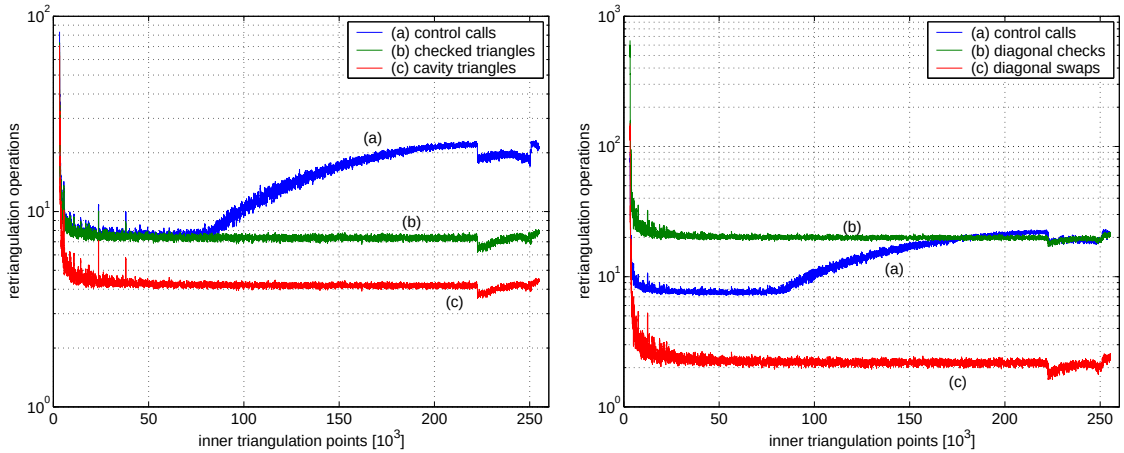
Figure 4.4 shows distribution of checking and modification operations during insertion of inner nodes for mesh SURF0C. Figure 4.5 shows distribution of checking and modification operations (for SURF0C). There can be observed a distinct spike at the beginning of this phase resulting from bad structure of the mesh at that time. However, the situation quickly normalizes and for the majority of inserted points the number of modifications is stable.

The variation of the average number of control calls per inserted node results from the hybrid method of evaluation of mesh elements quality for refining. The metric for basic quality evaluation is calculated in one point (barycenter of triangle). If element reaches the quality above the refinement threshold the length of triangle edges are also checked in local metric, which requires additional three control space calls. As the triangulation process advances, the number of triangles requiring the additional quality check gradually



(a) empty circumcircle criterion (b) inner angles criterion

Figure 4.4: Delaunay retriangulation of boundary nodes (50-running average)



(a) empty circumcircle criterion (b) inner angles criterion

Figure 4.5: Retriangulation of inner nodes (50-running average) using different criteria

increases. The discontinuity in charts is caused by reaching another threshold value, where the retriangulation method is slightly adjusted.

Table 4.3 presents statistics of total meshing time (together with smoothing) and quality of created meshes for both Delaunay retriangulation criteria. Retriangulation with empty circumcircle criterion proves to be consistently faster than the inner angles method and has been chosen as default in the presented mesh generator.

4.3 Discretization of Contours

For discretization of contours of surface patches, an iterative placement of points for creating edges of unitary length in local metric is used [160, 161]. Procedure runs in parametric

Table 4.3: Comparison of retriangulation methods, EC – empty circumcircle criterion, IA – inner angles criterion, NT – number of triangles in the final mesh, and τ_T – triangulation speed (triangles per sec.)

		NT $[10^3]$	$\overline{\eta}_{\mathcal{M}}$	$\sigma_{\eta_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$\overline{\mathcal{E}}_K$	$\tau_T[10^3/s]$
AN1	(EC)	205.8	0.962	0.046	1.024	0.159	2.797	18.9
	(IA)	206.0	0.962	0.047	1.023	0.160	2.873	16.5
SURF0B	(EC)	507.7	0.973	0.034	1.026	0.151	1.813	22.7
	(IA)	508.0	0.973	0.035	1.026	0.151	1.954	19.8
SURF1	(EC)	20.2	0.939	0.061	0.989	0.207	1.609	24.4
	(IA)	20.3	0.940	0.060	0.986	0.207	1.414	21.6

space of the selected patch, and the obtained discretization may be further refined according to control space defined in other patches adjacent to the given contour.

select $\mathbf{p}$ – one of surface patches adjacent to contour $\mathbf{c_p}$
calculate iteratively a set of contour parameters $\{t_i : d_{\mathcal{M}}(\mathbf{c_p}(t_i), \mathbf{c_p}(t_{i+1})) = 1\}$
where $t_0 = t_{\text{begin}}$ and $t_n = t_{\text{end}}$
if $d_{\mathcal{M}}(\mathbf{c_p}(t_{n-1}), \mathbf{c_p}(t_n)) \neq 1$ **then**
  adjust positions of few last nodes
if *contour is adjacent to several surface patches* **then**
  check conformity of control space for adjacent patches
if *required even number of segments* ***and*** $n \neq 2k$ **then**
  insert additional node and adjust positions of neighboring nodes
for $i \leftarrow 1$ **to** $n-1$ **do** calculate $P_i^{3d} \leftarrow \mathbf{p}(\mathbf{c_p}(t_i))$

Algorithm 4.1: Discretization of three-dimensional contour

All contours are discretized independently, procedure of discretization for each contour (Alg. 4.1) consists of following steps:

1. *Choosing adjacent surface patch.* Each contour can be adjacent to more than one surface patch. In such case the discretization of this contour should take into account metric data from control spaces of all adjacent patches. In the proposed approach one of these patches is selected for initial discretization, which is then further adjusted using metric information from all other patches. The selection of patch for the initial discretization is based on an estimated metric length of contour according to control space of each patch, and the patch with maximum value (i.e. which will require highest number of nodes) is chosen.

2. *Creating set of discretization nodes.* Starting from one of vertices of the contour (with curve parameter $t_{\text{begin}} = t_0$) successive nodes with parameter values t_i are calculated in order to fulfill the condition $d_{\mathcal{M}}(\mathbf{c_p}(t_{i-1}), \mathbf{c_p}(t_i)) = 1$ using local metric and iterative refinement.

3. *Adjusting last segment.* The metric length of the last segment $d_{\mathcal{M}}(\mathbf{c_p}(t_{n-1}), \mathbf{c_p}(t_n))$ is usually shorter than requested by CS. In order to improve the quality of discretiza-

tion, this difference $\delta_t = 1 - d_{\mathcal{M}}(\mathbf{c_p}(t_{n-1}), \mathbf{c_p}(t_n))$ is distributed over several previous segments. If $\delta_t < 0.5$ then selected number of previous segments are proportionally shortened. Otherwise the node t_{n-1} is removed and the length of some previous segments is increased.

4. *Conformity checking for metric field from other adjacent patches.* If there are more than one surface patch adjacent to this contour, an additional checking is performed. For each adjacent patches its CS is used to calculate the local metric of successive discretization segments. If length of any segment is too large ($d_{\mathcal{M}}(\mathbf{c_p}(t_{i-1}), \mathbf{c_p}(t_i)) > 2$) it is split by inserting new nodes until the segment is sufficiently refined.

5. *Checking even number of segments.* If the resultant mesh is to be quadrilateral, the condition of even number of discretization segments for each patch has to be preserved. In such case an additional node is inserted into each contour with odd number of segments. In order to make the insertion possibly smooth, the longest segment is chosen and a selected number of neighboring nodes are also adjusted.

6. *Projecting discretization nodes to all adjacent patches.* After the set of nodes is ready it has to be projected into parametric space of all adjacent patches (other than the initially selected one). For simple representation (like plane) the parametric coordinates can be calculated directly. For more complex representations an iterative procedure is used, where each node becomes starting point for searching for the next one.

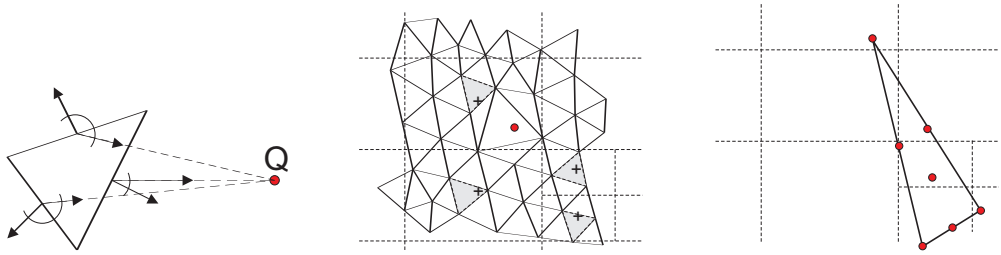
4.4 Triangulation of Boundary Nodes

The goal of this procedure is to create a triangulation of a given set of boundary nodes, gathered from all exterior and interior contours of this patch. The Delaunay triangulation and incremental insertion algorithm was selected in this work.

The procedure starts with creating an initial simple triangulation consisting of two triangles forming a rectangle containing the whole set of points to be triangulated. The nodes are then being inserted in randomized order with local retriangulation of the mesh in order to restore the Delaunay property. Randomization improves the efficiency of the process be increasing the quality of the intermediate structure of the created triangulation. Since the created DT was based only on nodes, an additional processing of boundary edges is required. During this constraining phase all boundary edges missing in the created mesh are recovered (by swapping of edges which for 2D triangulation is trivial and guaranteed to succeed) and obsolete elements (outside the domain) are removed. The obsolete elements are identified using a marking procedure. All elements directly adjacent to boundary contours are marked as valid, using the input geometry specification. The adjacency information is then used to propagate the marking to other valid triangles, avoiding trespassing of contours. After the iterative procedure is finished, all unmarked elements can be safely removed.

4.4.1 Localization of Containing Triangle

The important part of the triangulation of boundary nodes by incremental insertion is the localization of the containing triangle. The cost of this operation has large impact on the overall efficiency of the whole algorithm. In the presented work the combination of two concepts is used: traversing the neighboring triangles in the direction of the inserted point (Fig. 4.6(a)) and selection of good starting triangle with quadtree structure (Fig. 4.6(b)).



(a) selection of adjacent triangle

(b) starting triangle

(c) representative points for triangle

Figure 4.6: Localization of containing triangle

Each created or modified triangle is sifted through the seeding-quadtree in order to update the best candidate for some leaves of the quadtree. Removing of triangle also requires updating the quadtree by removing references. Each triangle is sift through the quadtree according to one or more representative points (Fig. 4.6(c)). The more points for representation of triangles, the better (on average) are the selected triangles for each leaf of the quadtree. However, the cost of quadtree update is also proportionally increased.

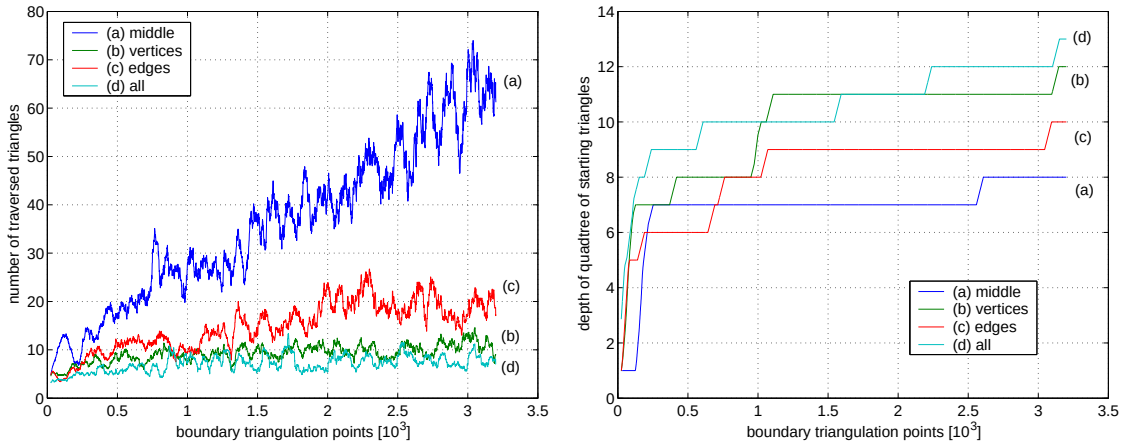
Figure 4.7(a) shows experimental statistics (for triangulation of mesh SURF0B) of 50-running average number of traversed triangles per each inserted point, for different sets of representative points: (a) barycenter of triangle, (b) all vertices of triangle, (c) midpoints of all edges of triangle and (d) all these points. The maximum depth of seeding-quadtree in all inspected cases is shown on Fig. 4.7(b).

Figure 4.8 presents influence of the leaf threshold of seeding-quadtree on boundary triangulation time for different sets of representative points of triangle. Smaller value of this threshold decreases the expected number of traversed triangles, but the size of the seeding-quadtree is growing, which increases the cost of updates of this structure.

Basing on the results obtained from practical tests on a number of various meshes, the set of vertices was selected as best option for representative points, with quadtree leaf threshold value of 300.

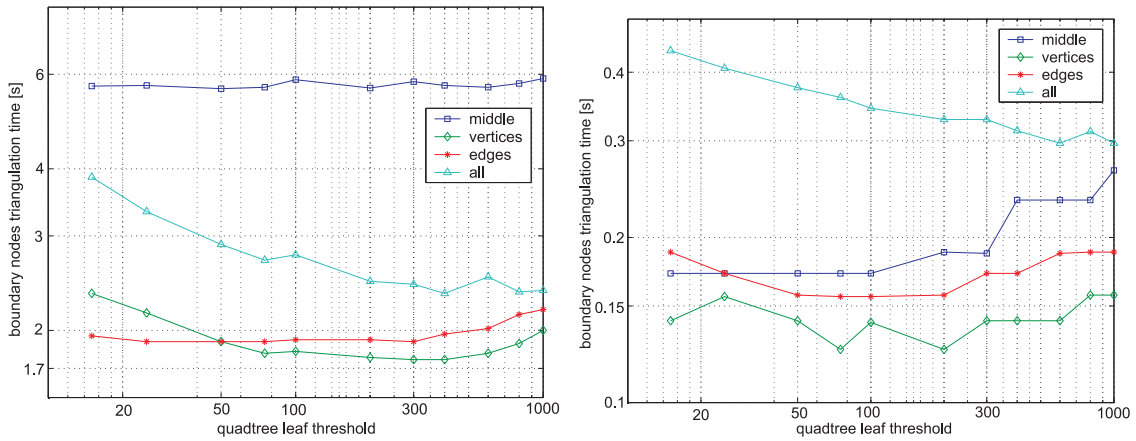
4.5 Insertion of Inner Nodes

The mesh is refined according to the prescribed sizing map by incremental introducing of new nodes. The triangles to improve are selected basing on the quality coefficient evaluating how close the given triangle resemblance the ideal triangle prescribed by local metric. The



(a) traversed triangles (50-running average) (b) maximum depth of seeding-quadtree

Figure 4.7: Selection of representative points for sifting triangles through seeding-quadtree



(a) mesh AN1 (b) mesh SURF0

Figure 4.8: Influence of leaf threshold of containing triangle quadtree on boundary triangulation time

new node can be inserted in the center of the circumscribed circle or in the middle of the longest edge of the triangle, with retriangulation.

This phase starts with reorganizing the triangles into heap-list which requires calculation of quality (proper size and shape) of all elements. Then, in each step of the main loop the worst triangle is taken from the heap and an additional node is inserted in its vicinity. The mesh is locally retriangulated and quality of affected elements is updated together with theirs position in the heap structure.

The quality of triangle K (required for guiding the incremental refinement process) is calculated as

$$q(K) = \frac{S_{\circ_I}}{S_{\circ_K}} \ , \tag{4.1}$$

where $S_{\circ_K}$ is the area of the circle circumscribed on the evaluated triangle and $S_{\circ_I}$ is the

area of the circumcircle of the ideal triangle (according to the local metric). In ideal *unit mesh* quality of all triangles should be equal to 1. If $q(K) < 1$ then the triangle is larger than required by local metric, and may require refining by inserting an additional node in its vicinity. Calculation of circumcircle area instead of taking the triangle area directly makes the quality coefficient sensitive for both size and shape discrepancy.

The procedure of mesh refinement stops when the quality of the worst element in the heap structure is already better than the selected quality threshold q_t. Due to the discrete characteristic of the refinement procedure the quality threshold has to be smaller than 1 in order to avoid over-refinement of the mesh. In the current implementation the default value of quality threshold $q_t = 0.58$ is used, which produces final meshes with an average metric length of edges close to 1.

The local metric is retrieved from CS for barycenter of the triangle selected for refinement. This metric is used for transformation of triangle vertices for calculation of new node and for reverse-transformation of this node before insertion into mesh. During evaluation of triangle quality this approach was found to be insufficient for areas of high metric gradation, where the refinement procedure might have been stopped too early. Therefore, for triangle quality evaluation during the mesh refinement procedure an additional condition is applied. If the quality $q(K)$ calculated in local metric from the barycenter of the triangle K is higher than the quality threshold q_t (which would block further refinement of this element) the length of all edges is also evaluated. If the metric length of any edge (with metric calculated for the middle of an edge) is too high ($l_{\mathcal{M}}(E) > 1.5$) the quality coefficient $q(K)$ is set to $0.95q_t$ in order to enforce the refinement of this triangle.

After selecting the current triangle to be improved, an additional node is inserted for refining the mesh in its vicinity and create better elements. Such node can be inserted e.g. in the middle of the longest edge of the triangle (MLE) or in the center of the circumscribed circle (CCS). The first method is fast and straightforward, the refined triangle is always guaranteed to be the containing triangle of the new node. The other method, inserting new node in the circumcircle produces meshes with better quality [162, 163], but it requires additional operations:

1. The circumcenter may lie outside the original triangle, which requires searching for containing triangle with neighbor-traversing procedure starting from the original triangle (due to the constrained triangulation this localization procedure is not guaranteed to succeed).

2. The circumcenter coordinates are calculated in metric space (for transformed coordinates of triangle vertices) and have to be transformed back to parametric space.

3. For variable metric field the Delaunay property is no longer global and the following cases have to be checked:

   - new node may lie outside of the boundary,
   - new node may lie too near to the contours,
   - new node may be too near of some other mesh point (vertices of the containing triangle are checked),

- the metric variation between the original triangle and the new node may be too large (i.e. after retriangulation according to the metric defined in the new node, the original triangle could remain unaltered).

If any of the above stated conditions is found to be true, the insertion is cancelled and the refinement of the selected triangle is postponed by ascribing to it an artificially large quality coefficient.

Table 4.4 presents examples of influence of the method of inner nodes insertion on the quality of obtained meshes and the speed of the meshing process for anisotropic AN1 (Fig. A.2) and surface mesh SURF0C (Fig. A.4). Beside the two mentioned methods: circumcenter of the triangles and middle of the longest edge of the triangle, an additional method (MLE+CCS) was proposed and tested. In this mixed approach both MLE and CCS methods were combined and used at different phases of mesh generation. The MLE method is used at the initial stages of triangulation of inner nodes, when large ratio of cancelled insertions are observed if using CCS method. After the quality of triangles reaches some prescribed threshold, the method is switched to CCS, which helps to obtain good quality of the final mesh.

Table 4.4: Influence of the method of inner node insertion, NT – number of triangles in the final mesh, τ_T – triangulation speed (triangles per sec.)

	NT $[10^3]$	$\overline{\eta}_{\mathcal{M}}$	$\sigma_{\eta_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$\overline{\mathcal{E}}_K$	$\tau_T[10^3/s]$
				AN1			
CCS	205.8	0.962	0.046	1.024	0.159	2.797	18.9
MLE	210.0	0.953	0.050	1.018	0.167	2.872	20.0
CCS+MLE	205.1	0.962	0.046	1.026	0.159	2.798	19.4
				SURF0C			
CCS	507.7	0.973	0.034	1.026	0.151	1.813	22.7
MLE	519.0	0.960	0.039	1.020	0.162	1.978	23.5
CCS+MLE	508.3	0.971	0.035	1.026	0.152	2.244	22.8

4.6 Improvement of Triangular Mesh

After the triangulation of surfaces is complete, the quality of mesh can be improved by application of some quality-enhancement oriented local mesh transformation procedures. The goal of improvement process is to increase the overall quality of the mesh (evaluated as an average quality of all mesh elements) and also minimize the number of elements with very low quality. The improvement process implemented in this work applies iteratively a set of local procedures (Alg. 4.2).

4.6.1 Laplace Smoothing

The popular Laplace smoothing method relocates nodes of the mesh in order to locally even out lengths of inner edges. During every pass of the procedure each internal node in

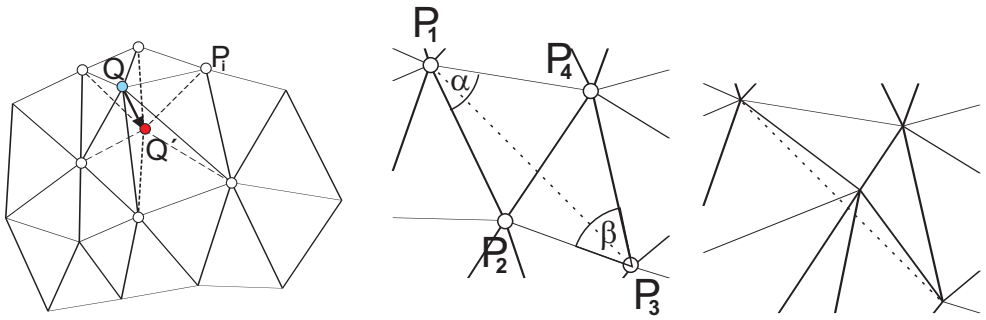
```
1 for i ← 1 to N_imp do
2 |   Laplace smoothing
3 |   topological edge swapping
4 |   geometric edge swapping
5 postprocessing
```

Algorithm 4.2: Triangular mesh improvement

the mash is placed at the barycenter of incident nodes P_i (Fig. 4.9(a))

$$Q' = \frac{1}{\mathrm{NE}(Q)} \sum_{i=1}^{\mathrm{NE}(Q)} P_i \ , \tag{4.2}$$

where $\mathrm{NE}(Q)$ is the number of edges adjacent to vertex Q.



(a) Laplace smoothing (b) edge swap (c) swap impossible

Figure 4.9: Local mesh transformation

Since for some configurations of the mesh (especially near concave boundary), the Laplace smoothing can result in a corrupted mesh (some elements may become inverted), an additional local check of adjacent mesh elements correctness is required after each relocation of the node. If the resulting mesh is invalid, the last modification is cancelled.

For anisotropic meshing with metric transformation approach the following adjustment of the Laplace formula (4.2) is proposed

$$\begin{array}{rcl} \mathbf{x}^{\mathcal{M}} & = & \sum_{i=1}^{\mathrm{NE}(Q)} \overrightarrow{QP_i}^{\mathcal{M}} \\ Q' & = & Q + \mathbf{x}^{\mathcal{R}} \end{array} \ , \tag{4.3}$$

where each vector $\overrightarrow{QP_i}^{\mathcal{M}}$ is taken in metric space from the middle point of that vector, and the conversion from $\mathbf{x}^{\mathcal{M}}$ to $\mathbf{x}^{\mathcal{R}}$ is calculated in metric from vertex Q.

In order to increase the efficiency of the improvement process while retaining the ability to preserve the anisotropic characteristic of the mesh, a mixed approach is proposed. The metric version (4.3) is used if the metric gradation ratio for a given point is sufficiently high. For areas with lower value of the metric gradation ratio, the metric can be safely assumed as locally constant and the simpler formulation (4.2) can be used.

In a single iteration of this procedure each node of the mesh is subjected to the Laplace smoothing operation once. The sequence of nodes selection for improvement is arbitrary.

4.6.2 Topological Edge Swap

The topological operations are usually applied in order to increase the quality of mesh by removing highly irregular nodes. For anisotropic graded meshing this category of methods has to be applied carefully, because they tend to reduce the gradation of mesh and may decrease the compliance of the mesh elements to the metric prescribed in CS.

The procedure used in this work operates by swapping of edges in order to even out the rank of neighboring mesh nodes (Fig. 4.9(b)). The edge P_1P_3 is swapped to P_2P_4 if

1. the quadrilateral $P_1P_2P_3P_4$ is convex (Fig. 4.9(c)),
2. $\mathrm{NE}(P_1) + \mathrm{NE}(P_3) > \mathrm{NE}(P_2) + \mathrm{NE}(P_4) + 2$.

In order to preserve the desired gradation of mesh, this method is used only in areas of low metric variance (i.e. where the metric gradation ratio $\gamma_{\mathcal{M}}$ calculated from CS is below some threshold). In a single iteration of this procedure each edge is checked only once.

4.6.3 Geometric Edge Swap

The edge swapping operation can be also used for geometric mesh improvement. The only difference is the selected criterion for application of this operation. The Delaunay inner angles criterion (which for Fig. 4.9(b) could be written as $\cos\alpha + \cos\beta \geq 0$) can be used directly. If the criterion (calculated for coordinates transformed according to local metric) is not fulfilled, the edge swap is performed.

The geometric edge swap procedure is applied iteratively for all edges in mesh (using a marking system to avoid unnecessary calculations) until the condition is satisfied throughout the mesh.

4.6.4 Postprocessing

After the selected number of iteration of the above described heuristics are applied, an additional procedure is used oriented on localizing and improving of any mesh element with very bad quality, which may have been left by previous operations. If any such element is found an local improvement procedure is used to further enhance the local quality.

4.6.5 Results

Table 4.5 presents the results of improvement process for several iterations steps ($N_{imp} = 5$) for an example mesh SURF0C. As in most typical meshing cases the effects of mesh improvement are clearly visible in few first steps and the influence of further smoothing is very small. In can be also clearly seen on Fig. 4.10. The default number of improvement steps $N_{imp} = 2$ for triangular meshes has been assumed as default in this work.

4.7 Conversion to Quadrilateral Mesh

If a quadrilateral mesh is required, an indirect method of triangles to quadrilaterals conversion is applied [164, 165]. The triangles are converted in a specific sequence, starting from

Table 4.5: Influence of successive improvement iterations on triangular mesh quality

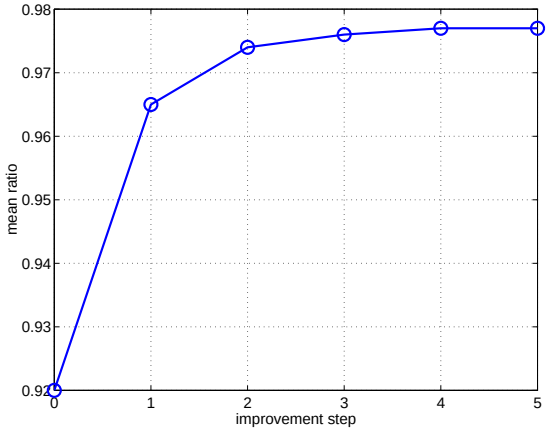
step	$\overline{\eta}_{\mathcal{M}}$	$\sigma_{\eta_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$\overline{\mathcal{E}}_K$
–	0.920	0.070	1.021	0.189	2.552
1	0.965	0.038	1.001	0.145	1.816
2	0.974	0.031	0.997	0.141	2.032
3	0.976	0.031	0.996	0.142	1.890
4	0.977	0.032	0.995	0.145	3.297
5	0.977	0.034	0.994	0.147	1.757

the boundaries of the surface patch. An all-quadrilateral mesh can be obtained provided that the number of boundary edges is even. In this work two methods of mesh conversion were implemented and adjusted for anisotropic surface meshing with metric transformation approach. The first one, based on simple *triangle merging* was introduced by Lee and Lo[40] for planar isotropic meshes, and further extended in [41, 166]. The latter one, *triangle morphing* algorithm (Q-Morph) was proposed by Owen [42]. Both algorithms use an advancing front approach for systematic forming of quadrilaterals at the front edges. However, while the *merging* method joins pairs of triangles, the Q-Morph uses more sophisticated approach, where each created quadrilateral may replace more than two triangles with additional modification of mesh in the vicinity.

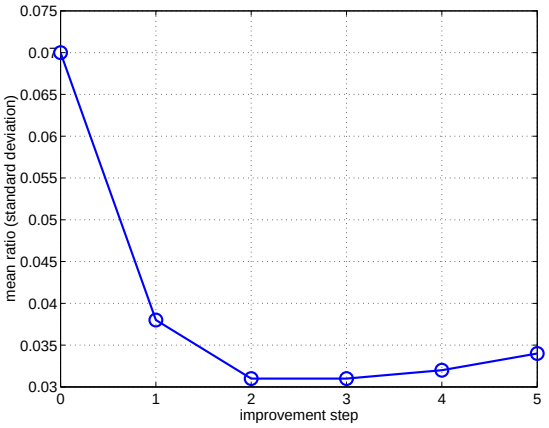
The two implementation of quadrilateral conversion algorithms (merging and morphing) can be used as a stand-alone procedures. In general, the merging method (without further smoothing) is much faster but tends to create meshes with lower quality. The morphing method produces quadrilateral meshes with higher quality (especially near the boundaries) at the cost of increased running time and complexity of algorithm. Since the general concept of both methods is similar, another approach is proposed in this work – the *mixed* method (Alg. 4.3). In this approach the morphing method is used in areas where well-aligned elements of good quality are expected (i.e. near boundaries and at areas of low metric gradation), otherwise the less costly simple merging is used.

```
1 gather a set of front edges A_E
2 classify all front edges in A_E
3 while A_E ≠ ∅ do
4 |   select front edge E_f with lowest rank
5 |   if E_f is near boundary or metric gradation at E_f is low then
6 |   |   create quad using morphing (Alg. 4.5)
7 |   else
8 |   |   create quad using merging (Alg. 4.4)
9 |   update A_E
```

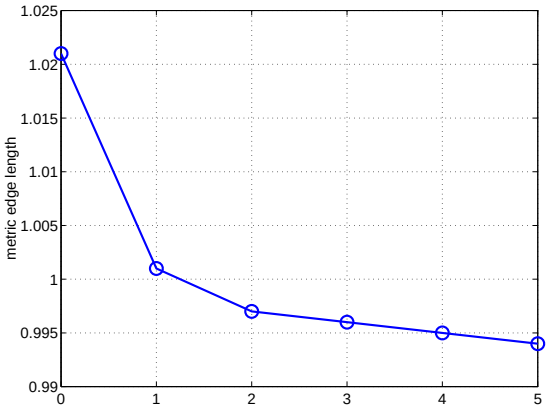
Algorithm 4.3: Generation of quadrilateral mesh using mixed approach



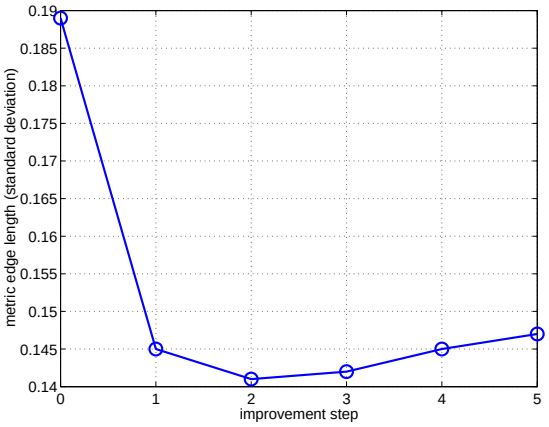
(a) mean ratio $\overline{\eta}_{\mathcal{M}}$



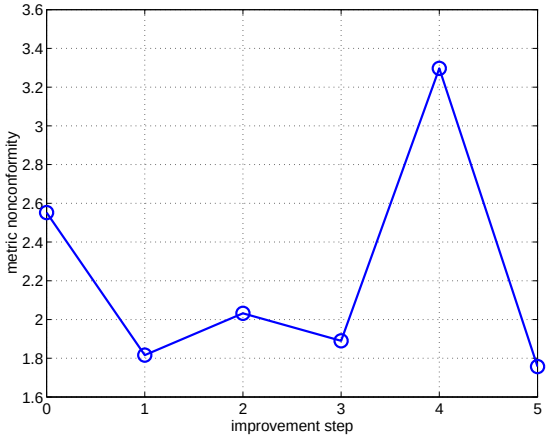
(b) mean ratio (standard deviation) $\sigma_{\eta_{\mathcal{M}}}$



(c) metric edge length $\overline{\mu}_{\mathcal{M}}$

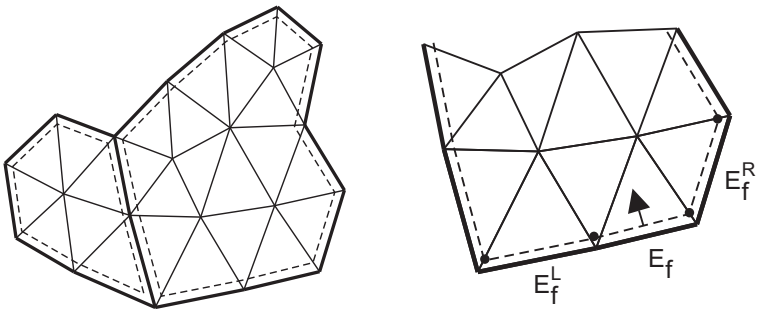


(d) metric edge length (standard deviation) $\sigma_{\mu_{\mathcal{M}}}$



(e) metric nonconformity $\overline{\mathcal{E}}_K$

Figure 4.10: Quality improvement of example mesh SURF0C



(a) conversion front (b) front edge

Figure 4.11: Creation of front for quadrilateral conversion

4.7.1 Creation of Conversion Front

The first step of the conversion algorithm is gathering, building and classification of front consisting of one or more cycles. All edges in mesh are inspected and front cycles are built by tracing boundary edges from each encountered unmarked boundary edge. The edges have to be marked on both sides, since inner boundaries are also possible (Fig. 4.11(a)).

After the whole front structure is ready, each front edge is being classified according to the following coefficients:

1. *Type* of front edge a_1 denotes its availability for creation of quadrilateral. If both side front edge angles $\alpha_{\{L,R\}} < \alpha_1$ (where $\alpha_{\{L,R\}} = \text{angle}(E_f, E_f^{\{L,R\}})$ and α_1 is assumed as $\frac{3\pi}{4}$), the coefficient $a_1 = 0$. $a_1 = 1$ if the condition is fulfilled by one of the side edges and $a_1 = 2$ otherwise.

2. *Small side angles* – when one of the side angles is very small $\alpha_{\{L,R\}} < \alpha_2$ (where α_2 is assumed as $\frac{\pi}{8}$), the coefficient $a_2 = 1$, otherwise $a_2 = 0$.

3. *Level* coefficient $a_3 = 0$ for the initial set of front edges and the number is incremented for each successive layer (i.e. new front edges created while forming new quadrilaterals).

4. *Metric length* coefficient $a_4 = \lfloor l^2_{\mathcal{M}}(E_f) \rfloor$ if $l^2_{\mathcal{M}} > 2$, otherwise $a_4 = 0$.

5. *Metric gradation* coefficient $a_5 = \lfloor \gamma_{\mathcal{M}}(E_f) - 1 \rfloor$.

The final rank of each front edge is calculated as a weighted average of all coefficients:

$$\text{rank}(E_f) = \sum_{i=1}^{5} \omega_i a_i \ . \tag{4.4}$$

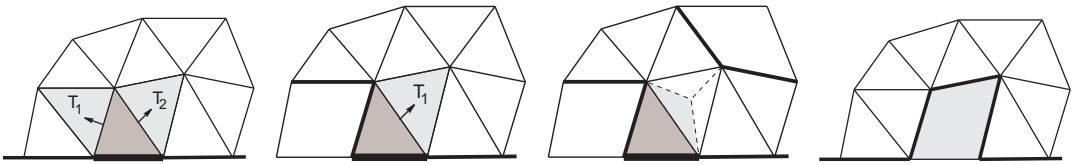
All edges are then organized in the heap structure, from which the edges with lowest ranking are successively fetched as a base front edges for creation of new quadrilaterals.

The selection of the base front edges is not arbitrary and it has heavy impact upon the efficiency of the process of the conversion and on the quality of the resulting mesh. With $\omega_1 = \omega_3 = \omega_4 = \omega_5 = 1$ and $\omega_2 = 10$ all coefficient have roughly equal importance with a preference for a special case, which needs appropriate handling.

4.7.2 Merging of Triangles

```
1  if two available adjacent triangles then
2  |   evaluate metric quality of potential quadrilaterals
3  |   select adjacent triangle with higher quality
4  else if one available adjacent triangle then
5  |   select this triangle
6  else
7  |   no available adjacent triangle
8  leave single triangle
9  update front edges
10 return
11 if merging creates front-cycle with odd number of segments then
12 |   split side triangle into three elements
13 merge two triangles into one quadrilateral
14 improve triangular mesh locally
15 update front
```

Algorithm 4.4: Conversion to quadrilateral by merging



(a) two incident triangles (b) one incident triangle (c) triangle split (d) new quad

Figure 4.12: Forming new quadrilateral by merging

The merging algorithm consists of the following steps:

1. Identifying the triangle incident to the selected base front edge.

2. Evaluation of elements incident to the base triangle. If both triangles are available (Fig. 4.12(a)) and can be merged with the base triangle, one of them has to be selected. The metric quality of both potential quadrilateral elements is calculated and the triangle forming quadrilateral with better quality is chosen. If only one triangle is available (Fig. 4.12(b)) this step can be skipped. Additionally, if the initial number of segments in front-cycles is not even, one of triangles in the mesh may have to be left in the final mesh.

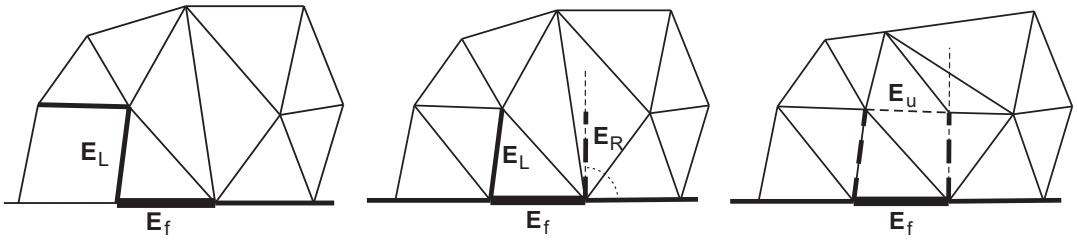
3. If the opposite vertex of the triangle selected for merging belongs to the front, an additional check is necessary whether the front joining is possible (in order to ensure an all-quadrilateral mesh, all front cycles have to maintain an even number of edges). If the joining operation would create a front cycle with odd parity, the triangle is split into three parts (Fig. 4.12(c)).

4. The two selected triangles are removed and a new quadrilateral is created instead (Fig. 4.12(d)). The triangular mesh near the created quad is improved with edge-swapping procedure and finally, the front edges are properly updated.

4.7.3 Morphing of Triangles

```
1 improve locally mesh ahead of front
2 check side front edges
3 check seams
4 check upper (front) edge
5 set side edges for quad
6 set upper edge for quad
7 form new quad
8 improve mesh ahead of front
9 update front
```

Algorithm 4.5: Conversion to quadrilateral by morphing



(a) using front side edge (b) setting side edges (c) recovering upper edge

Figure 4.13: Forming new quadrilateral by morphing

The morphing algorithm consists of the following main steps:

1. The angles between the selected base edge and both adjacent front edges area checked. If the angle is not too large (in metric space), such front edge can be used directly as the side edge of the new quadrilateral (Fig. 4.13(a)).

2. Depending on the angle and length ratio of adjacent front edges, one of three topological operations may be applicable (Fig. 4.14). These operations allow to preserve good quality of created elements also in areas of large size variation. If conditions for any of these operations are fulfilled, the mesh is locally transformed and this conversion iteration is terminated.

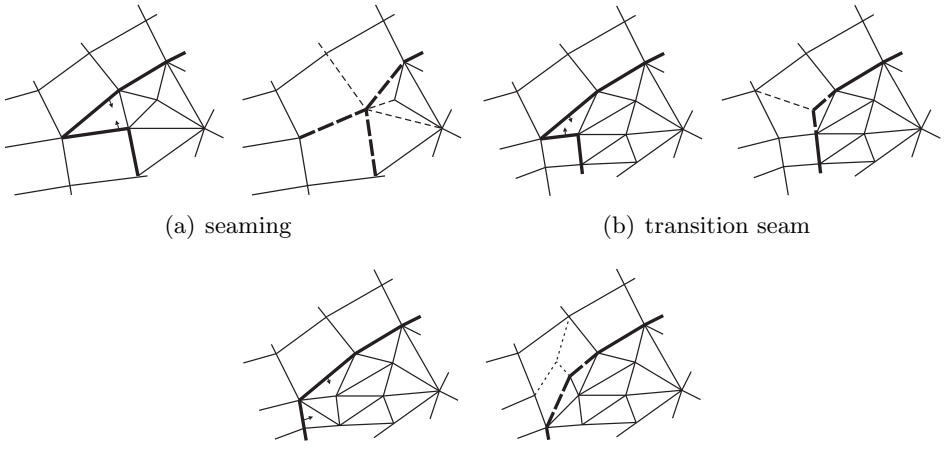
3. Edges of the new quadrilateral, which can not be formed by the existing side front edges have to be created otherwise. Depending on quality conditions (based on angles and size calculated in metric space) the new edges are found either using some edge already existing in the mesh or by creating such edge with local transformation like edge-swapping or splitting (Fig. 4.13(b)). If the selected side edges are joining or

splitting some front-cycle, an additional care has to be taken in order to avoid creation of cycles with odd number of segments.

4. After the side faces of the new quad are ready, the upper edge has to be set. If it is not directly available, an edge recovery procedure is used. Unlike the boundary edge recovery described earlier, here this procedure may require to additionally move some mesh points out of the line of the edge being recovered. Recover the upper edge of the quadrilateral (by edge swapping if needed) (Fig. 4.13(c)).

5. When all edges of the new quadrilateral are ready, the actual transformation may take place. All triangles inside these edges are removed iteratively (possibly with some points) and the new quadrilateral is formed.

6. Finally, a number of local improvement operations are applied in order to increase the quality of the quadrilateral and to adjust this element with a layer of adjacent quadrilaterals. Also the mesh before the front is smoothed in order to improve the robustness and efficiency of the successive conversion steps. After the local improvement is done, the front is updated and all affected front edges are reclassified.



(a) seaming (b) transition seam

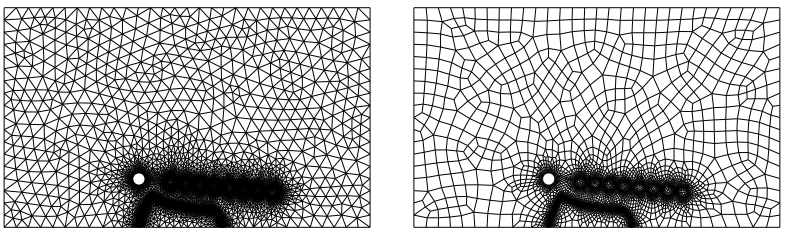
(c) transition split

Figure 4.14: Topological corrections during quadrilateral conversion

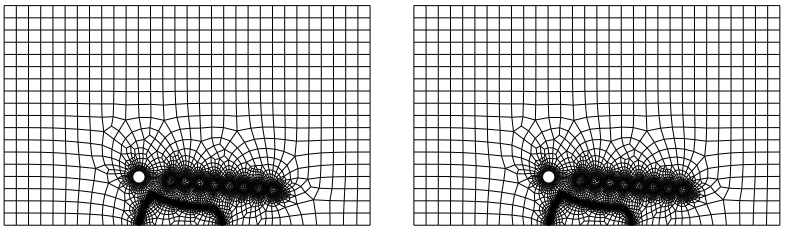
Since the morphing method sets high quality standards for created quadrilaterals, there are possible situations, where the algorithm may fail to succeed in one of the conversion steps (e.g. in finding proper side edges). If such situation occurs, the selected base front edge is reclassified and pushed back, to be considered later. If the problem persists the whole algorithm fails to finish successfully, the quality requirements (like side edge angle ranges) are slightly loosened and the conversion is restarted for all triangles in the mesh, which were left. If this solutions does not help, the simple merging method is used directly, which guarantees successful completion of the conversion process.

4.7.4 Results for Different Conversion Methods

Figures 4.15 (NISA1) and 4.16 (SURF0A) present results of quadrilateral mesh conversion for surface and planar domain using different methods (statistics in Tab. 4.6).

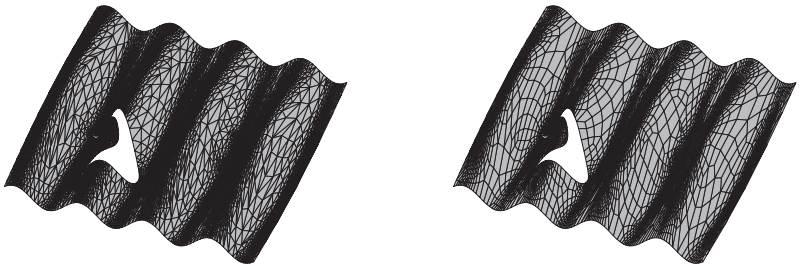


(a) triangular mesh (b) merging

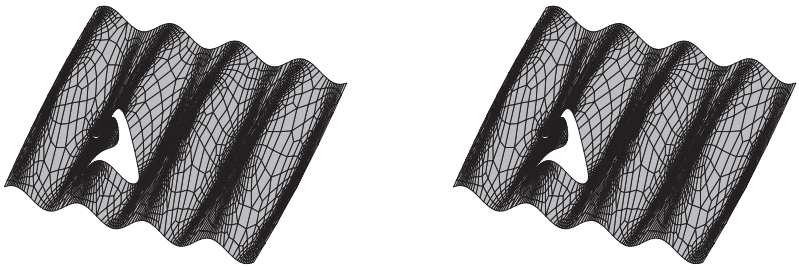


(c) morphing (d) mixed

Figure 4.15: Quadrilateral conversion of planar mesh



(a) triangular mesh (b) merging



(c) morphing (d) mixed

Figure 4.16: Quadrilateral conversion of surface mesh

Table 4.6: Different conversion methods, NQ – number of quadrilateral elements, t_c – conversion time, t_s – improvement time

	NQ $[10^3]$	$t_c[s]$	$t_s[s]$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	NQ $[10^3]$	$t_c[s]$	$t_s[s]$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$
	NISA1					SURF0A				
merging	20.4	23.81	1.17	0.936	0.180	6.0	0.67	0.29	0.908	0.221
morphing	20.2	21.80	1.09	0.928	0.166	5.1	1.11	0.24	0.980	0.242
mixed	20.2	22.03	1.10	0.929	0.167	5.1	1.00	0.25	0.980	0.242

4.8 Improvement of Quadrilateral Mesh

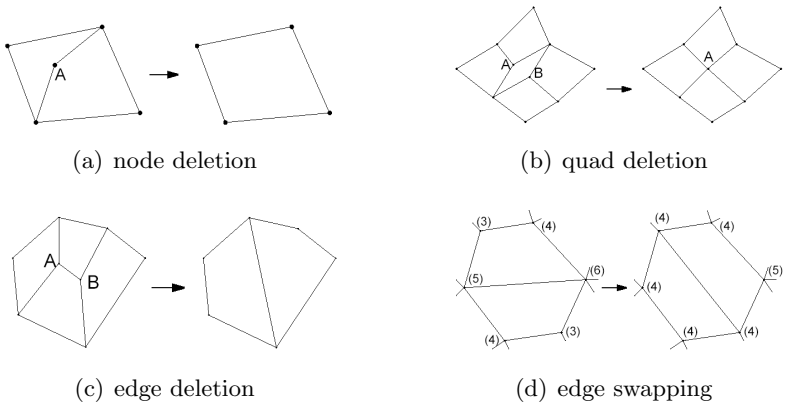
Once the quadrilateral mesh is generated, several improvement techniques can be applied in an iterative manner (Alg. 4.6).

```
1 for i ← 1 to N_imp do
2     topological transformation
3     Laplace smoothing
4 postprocessing
```

Algorithm 4.6: Quadrilateral mesh improvement

4.8.1 Topological Transformation

Several methods (Fig. 4.17) can be used to improve the topological quality of the mesh (i.e. based on the connectivity information only). Procedures belonging to this category usually smoothen the transitions between elements by swapping edges or removing unnecessary entities (e.g. vertices or edges). With respect to anisotropic meshes such quality enhancement may potentially worsen the conformity of this mesh to the prescribed CS. However, if used with care, these procedures can substantially lessen the number of really bad quality quadrilaterals.



(a) node deletion (b) quad deletion

(c) edge deletion (d) edge swapping

Figure 4.17: Topological smoothing of quadrilateral mesh

4.8.2 Laplace Smoothing

A mixed approach for Laplace smoothing with anisotropic metric was used as described for improvement of triangular meshes.

4.8.3 Postprocessing

Additionally, special care is taken of the quality of elements adjacent to the boundaries of the domain, as they are usually important for the further simulation process.

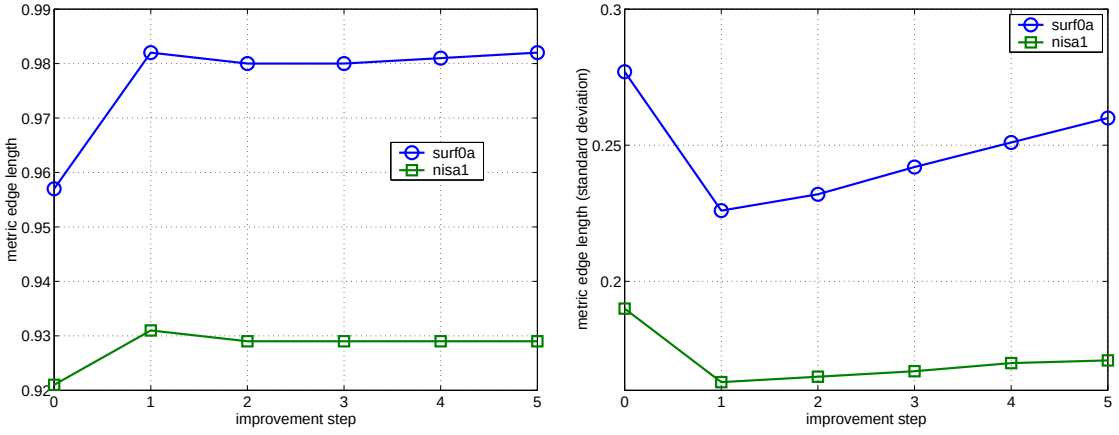
4.8.4 Results

Table 4.7 presents the results of improvement process for several iterations steps ($N_{imp} = 5$) for quadrilateral meshes SURF0A and NISA1.

Table 4.7: Influence of successive improvement iterations on quadrilateral mesh quality

step	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$
	SURF0A		NISA1	
–	0.957	0.277	0.921	0.190
1	0.982	0.226	0.931	0.163
2	0.980	0.232	0.929	0.165
3	0.980	0.242	0.929	0.167
4	0.981	0.251	0.929	0.170
5	0.982	0.260	0.929	0.171

The effects of improvement are substantial in few first steps and the influence of further smoothing is very small or even degrading (Fig. 4.18). The default number of improvement steps $N_{imp} = 2$ has been assumed in this work for quadrilateral meshes.



(a) metric edge length $\overline{\mu}_{\mathcal{M}}$ (b) metric edge length (standard deviation) $\sigma_{\mu_{\mathcal{M}}}$

Figure 4.18: Quality improvement of example quadrilateral meshes

4.9 Chapter Summary

The successive steps of mesh generation on parametric surfaces were presented. The triangular mesh of each surface patch is created using an incremental Delaunay triangulation algorithm in parametric space with anisotropic metric transformation. The boundary nodes created by discretization of patch contours are successively introduced into the mesh using the Delaunay criterion. The obtained convex hull triangulation is constrained by recovering boundary edges and removing obsolete elements. All triangles are evaluated and additional inner nodes are inserted. Finally, some mesh improvement methods are applied. The selected steps of the presented algorithm (e.g. criteria of retriangulation, placement of inner nodes, metric introduction, etc.) were inspected in detail and a number of improvements were proposed.

For generation of quadrilateral meshes an indirect frontal approach was inspected which iteratively converts mesh triangles into quadrilaterals. Two methods described in literature (merging and morphing of triangles) were implemented with necessary adjustments for anisotropic meshing with metric transformation tensor approach. As a result of this research a mixed method is proposed where both inspected conversion method may be used alternately, depending on the local metric characteristics. Finally, a number of examples are provided illustrating the performance of the implemented procedures.

Chapter 5

Anisotropic Volume Mesh Generation

5.1 Introduction

An algorithm of anisotropic volume mesh generation based on Delaunay retriangulation is described. The input data of the volume mesh generator are the surface meshes of boundaries of the three-dimensional domain. The input data may also contain additional meshes of one- or two-dimensional geometries embedded in the three-dimensional domain (inner boundaries). It is assumed, that the input surface meshes are final and will not be modified during the process of volume mesh generation.

5.1.1 Related Research

An extension of Delaunay incremental insertion algorithm to three-dimensional meshing with boundary recognition has been pioneered by George[24] and Weatherill[25] and further researched in [26, 27, 151, 167–170]. The Delaunay kernel for three-dimensional refinement was generalized for anisotropic meshing with metric tensor by [86, 171].

Much work has been and still is devoted to overcome two main problems of three-dimensional constrained Delaunay mesh generation: ensuring the integrity of boundaries and avoiding creation of very flat tetrahedra called slivers. In typical approach the recovery of boundary edges and faces which are absent in the created mesh (due to the unconstrained triangulation) is carried out by alternated application of local transformations and insertion of additional nodes near or at the missing edges or faces[24, 25]. While effective in many situations, the successful recovery of boundary using this method is not guaranteed. In some cases the volume triangulation is coupled with refinement of the surface meshes[172, 173], which however may require insertion of excessive number of nodes at the missing boundaries. Other methods[174] involve decomposition of the geometry into simpler convex regions posing less problems with recovery of their boundary.

The other problem of three-dimensional Delaunay triangulation is the creation of bad quality sliver elements, which have all four vertices lying nearly on the same plane. In some approaches the mesh generation algorithms are adjusted in order to avoid creation of such

elements [175, 176]. The procedure of removing sliver elements can be also applied as a part of the improvement phase, after the tetrahedral mesh is created [150, 169, 177]. In most cases these methods require introduction of additional nodes, resizing elements or even introduce some carefully chosen characteristics of the mesh (i.e. weighted triangulation). Not all of these approaches can be applied to anisotropic meshing with variable contents of the control space.

5.1.2 Contents

In this chapter there is described the process of generation of tetrahedral meshes on three-dimensional surfaces (Fig. 5.1).

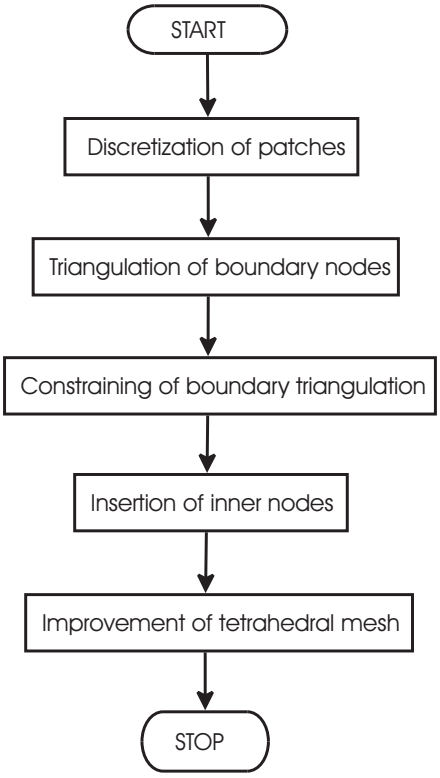


Figure 5.1: Overview of mesh generation process of three-dimensional volumes

The mesh generation using modified Delaunay technique (Sec. 5.2) begins with triangulation of a set of vertices gathered from the boundary surface mesh. Starting from an initial simple mesh enclosing all boundary nodes, the boundary triangulation is created using an incremental insertion procedure (Sec 5.3). After constraining of the mesh by recovery of missing boundary edges and faces (Sec. 5.4) the mesh is further refined with additional mesh points (Sec. 5.5) following the information from the control space. The meshing process is finalized with some procedures oriented on improving the quality of mesh elements (Sec. 5.6).

5.2 Incremental Retriangulation

The meshes are constructed using a modified algorithm of incremental insertion working in Riemannian metric space [127]. The essential part of this algorithm is the operation of local mesh reorganization which inserts a single vertex into mesh maintaining its validity and optionally some additional mesh properties. The retriangulation criteria implemented in this work for three-dimensional meshing are based on techniques derived from the Delaunay triangulation: empty sphere and inner angles criterion, also described in the previous chapter for surface meshing. Due to anisotropy introduced by metric transformation and weaker properties of three dimensional Delaunay triangulation, both techniques had to be properly adjusted.

Both procedures of retriangulation start with a valid mesh $\mathcal{T}_h^n(\Omega)$ with n nodes, new vertex $V \in \Omega, V \notin \mathcal{T}_h$ and the tetrahedron K_c containing the new vertex V. The result of retriangulation is a valid mesh $\mathcal{T}_h^{n+1}$ including V. The problem of finding the containing tetrahedron concerns mainly the phase of boundary nodes triangulation and is described in more detail in that section.

5.2.1 Criterion of Empty Circumsphere

The retriangulation procedure using this criterion consists of the following steps:

1. *Finding all tetrahedra which contain the new vertex V in their circumsphere.* This set of elements $\mathcal{C}_V$ (called cavity) is created by iterative inspection of adjacent tetrahedra starting from the containing tetrahedron K_c. The *in-sphere* test is calculated in metric space retrieved from CS for vertex V and treated as locally constant during the whole procedure of retriangulation. In case of positive result of the in-sphere test a tetrahedron is inserted into $\mathcal{C}_V$ and its neighbors are set to be also checked. The already checked tetrahedra are marked in order to avoid repetitive testing of any element. The procedure is terminated when there are no more unmarked tetrahedra adjacent to the tetrahedra in $\mathcal{C}_V$.

2. *Pruning the initial cavity.* From the initial cavity $\mathcal{C}_V$ of tetrahedra marked for deletion the set $\mathcal{C}_f$ of cavity faces is constructed (also called *cavity shell*). For each tetrahedron in $\mathcal{C}_V$ all its faces are checked and any boundary face or face adjacent to a tetrahedron not belonging to $\mathcal{C}_V$ is inserted into $\mathcal{C}_f$. Due to limited numerical precision and constrained triangulation, the set $\mathcal{C}_f$ created using this procedure is not guaranteed to be 'star-shaped' with respect to the new vertex V (i.e. some vertices in this cavity shell may not be directly visible from the vertex V). Since such situation would result in creation of inverted tetrahedra, an additional procedure is used which inspects all faces from $\mathcal{C}_f$. If any face together with the new vertex would produce an invalid tetrahedron (or tetrahedron with volume below the prescribed threshold $\epsilon_{\mathcal{M}}$), the tetrahedron adjacent to this side of face is removed from $\mathcal{C}_V$ ($\mathcal{C}_f$ is also modified, which requires additional checks).

   For constrained triangulation (i.e. during the boundary recovery and further procedures when some edges or faces are marked as fixed) only the tetrahedra adjacent to

the given element through a non-boundary face are considered. An additionally check is also required to ascertain that including some tetrahedron into $\mathcal{C}_V$ (and removing it later) will not produce any stranded boundary face or edge. In case of detecting such situation this element is also removed from $\mathcal{C}_V$.

3. *Modification of mesh.* After validation all elements from the final cavity $\mathcal{C}_V$ are removed and for each face from $\mathcal{C}_f$ a new tetrahedron is created (joining the face with the new vertex) and inserted into mesh. Depending on the current phase of mesh generation process some additional operations may be required for both the removed set of elements (e.g. removing references from the octree structure for containing element localization) and the new ones (e.g. calculation of their quality and updating the heap organization of mesh elements).

In the refinement mode (i.e. when the inserted point is not an obligatory vertex from the surface mesh) an additional condition is checked: whether the tetrahedron which is to be improved by this insertion is actually part of the final $\mathcal{C}_V$ set of elements to be removed. If it is not the case, insertion of this node is cancelled altogether as pointless.

5.2.2 Flipping of Edges and Faces

The edge-flipping algorithm presented in the previous chapter for two-dimensional Delaunay meshing can be also extended to 3D [174]. Using this approach the containing tetrahedron is directly split in order to accommodate the inserted node. The node can be placed within the element (one tetrahedron is replaced by four), on one of the faces (two tetrahedra are replaced by six) or on one of the edges (m tetrahedra incident to this edge is replaced by $2m$ blocks). After insertion of the new node an iterative procedure of edge- and face-flipping (Alg. 5.1) is used to locally restore the Delaunay property of mesh.

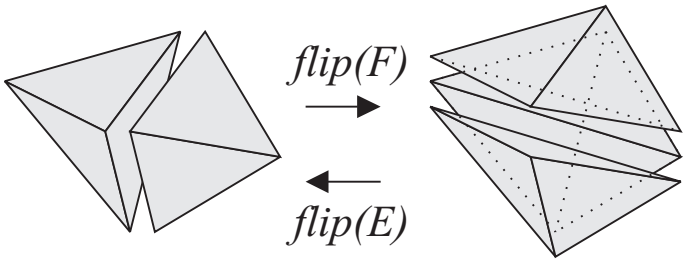


Figure 5.2: Basic transformations (T32/T23): edge- and face-flip

Operation of *flip*(F) transforms two tetrahedra with common face into three tetrahedra with common edge (Fig. 5.2). The basic *flip*(E) performs the reverse transformation. The edge-flip operation can be also extended to cases with more than three tetrahedra incident to an edge. Each operation requires the validity check, whether the transformation would not create inverted elements (i.e. tetrahedra with negative volume). The next condition to be checked is whether the transformation is advantageous i.e. whether the local quality of tetrahedra would increase. The quality condition can rely on enhancing the average or minimum quality represented by metric volume or some other quality coefficient.

```
1  S_V ← set of vertices of initial tetrahedra
2  while S_V ≠ ∅ do
3      foreach non-boundary node V ∈ S_V do
4          foreach edge E incident to V do
5              if unmarked(E) and flip_possible(E) and flip_advantageous(E) then
6                  flip(E)
7                  unmark all edges and faces of modified tetrahedra {K_i}
8                  S_V ← S_V ∪ vertices of {K_i}
9                  break
10             else
11                 mark(E)
12             foreach face F incident to E do
13                 if unmarked(F) and flip_possible(F) and flip_advantageous(F) then
14                     flip(F)
15                     unmark all edges and faces of modified tetrahedra {K_i}
16                     S_V ← S_V ∪ vertices of {K_i}
17                     break
18                 else
19                     mark(F)
20     S_V ← S_V \ V
```

Algorithm 5.1: Local retriangulation with edge- and face-flipping

These quality tests are calculated in local metric space of the given set of tetrahedra to be transformed.

In order to avoid infinite loops, the stop condition of empty set of active nodes S_V is accompanied by additional conditions:

- if some prescribed maximum number of transformations is reached,
- if the same number of transformations repeats in three consecutive steps.

Due to a significant number of swaps involved and significantly lower efficiency (and robustness) in comparison with the empty circumsphere criterion, the latter procedure of retriangulation was consistently used in the presented mesh generator.

5.3 Triangulation of Boundary Nodes

The first phase of the volume mesh generation is the unconstrained triangulation of the set of NP_B boundary nodes, based on a Delaunay incremental insertion method. The proposed modification of this algorithm for the task of mesh generation problem includes an additional systematic refinement of the mesh during the whole process of boundary triangulation (Alg. 5.2). The coefficient q_{Bt} controls the level of refinement (higher value of q_{Bt} increases the number of additional nodes, for $q_{Bt} = 0$ the algorithm returns to standard version).

```
create boundary surface mesh T_h^S
create initial volume mesh T_h
if q_Bt > 0 then
    insert all V ∈ T_h^S into auxiliary octree structure
for i ← 1 to NP_B do
    select random V_i ∈ T_h^S (without repetitions)
    find tetrahedron K_c ∈ T_h containing V_i
    insert V_i into T_h with retriangulation
    if q_Bt > 0 then
        refine mesh T_h in vicinity of V_i according to q_Bt
```

Algorithm 5.2: Unconstrained 3D triangulation of boundary nodes

The unconstrained triangulation of boundary nodes includes the following steps:

1. *Preparation of boundary mesh.* Before the volume triangulation procedure can be started, the boundary description has to be created. This boundary is formed by a closed triangular mesh which is either given directly as input data or it has to be retrieved from the triangular meshes created for surface patches using algorithms described in Chapter 4. In the latter case the triangular discretizations of all surface patches defining each volume block have to be projected into 3D space and combined into a consistent boundary description.

2. *Creation of initial volume mesh.* From the set of boundary nodes the bounding box is calculated. This regular domain is slightly enlarged and used for creation of a simple initial mesh consisting of eight tetrahedra.

3. *Creation of octree of boundary nodes.* If the auxiliary refinement is to be applied, all boundary nodes are initially sift through the octree structure in order to facilitate their efficient localization in further steps.

4. *Insertion of boundary nodes.* The boundary nodes are inserted one by one, in randomized order. Inserting a node into the mesh requires localizing the tetrahedron containing the new point and the retriangulation procedure using the empty sphere criterion described earlier. After retriangulation an additional refinement step may be also performed.

5.3.1 Auxiliary Refinement for Boundary Triangulation

The efficiency and robustness of incremental retriangulation procedures is highly influenced by the geometrical quality of the intermediate mesh. In case of low quality of elements the efficiency of operations such as localization of containing tetrahedron or retriangulation may be substantially lower than for more regular mesh.

In the proposed modification the auxiliary nodes are inserted not only during the boundary constraining (which is commonly used in literature) but also during the procedure of triangulation of boundary nodes. After retriangulation of each inserted boundary node,

all tetrahedra adjacent to this node are inspected. If the quality of the worst element (calculated using *mean ratio* in metric space local for inserted node V_i) $\eta_{\mathcal{M}}$ is below some prescribed threshold value q_{Bt}, the mesh is refined locally by inserting an auxiliary node at the middle of the longest edge of this tetrahedron. The operation is repeated iteratively, until all tetrahedra adjacent to V_i have quality coefficient higher than q_B or no new auxiliary point can be inserted.

Since at this phase of mesh generation the boundary information is not fully represented in the created mesh, an additional check has to be made to avoid collision of some auxiliary node with any of not yet inserted boundary nodes. The insertion of the auxiliary node is cancelled if it is too close to one of such boundary node. In order to decrease the number of required distance calculations, an octree structure of boundary nodes is used, which is initialized at the beginning of boundary nodes triangulation.

The cost of auxiliary nodes insertion includes the initial preparation of an additional octree structure, retrieving the set of tetrahedra adjacent to vertex V_i, calculating their geometrical quality and actual insertion of new nodes into the mesh (with retriangulation). However, in most cases the benefits of increased efficiency of retriangulation operations are higher than these costs.

The difference between the standard algorithm and the proposed modification is depicted on Fig. 5.3. The presented charts show the (100-running) average of retriangulation cost expressed in terms of number of necessary in-sphere tests (i.e. checked elements), size of initial and final cavity and number of replaced elements per each inserted node. The results are gathered from generation of three different meshes: graded but isotropic cubic domain CUBE2 (Fig. A.10), anisotropic CYLINDER2 (Fig. A.16) and less anisotropic but more complex COMP1 (Fig. A.21). The charts include all phases of mesh generation requiring retriangulation: boundary triangulation, constraining and insertion of inner nodes.

These examples illustrate the main effects of the described modification. There are no visible improvements for isotropic mesh, since in this case the quality of the intermediate mesh is sufficient even without the additional refinement. However, for anisotropic meshes the retriangulation cost of boundary triangulation is visibly reduced. Additionally, the auxiliary refinement reduces the amount of missing entities which have to be recovered during the constraining phase and hence the smaller number of retriangulation operations (resulting from the points inserted near the missing entities in order to promote their recovery) in this variant of algorithm. Table. 5.1 presents the average statistics for the whole meshing process.

Table 5.1: Average cost of retriangulation for volume meshing, $\overline{N}_{\mathrm{CS}}$ – number of CS calls, $\overline{N}_{\mathrm{sc}}$ – average number of in-sphere checks per each inserted point, $\overline{N}_{\mathrm{ci}}$ and $\overline{N}_{\mathrm{cf}}$ – average number of tetrahedra in initial and final cavity per each inserted point, $\overline{N}_{\mathrm{nt}}$ – average number of new tetrahedra created per each inserted point

mesh	$\overline{N}_{\mathrm{CS}}$	$\overline{N}_{\mathrm{sc}}$	$\overline{N}_{\mathrm{ci}}$	$\overline{N}_{\mathrm{cf}}$	$\overline{N}_{\mathrm{nt}}$	$\overline{N}_{\mathrm{CS}}$	$\overline{N}_{\mathrm{sc}}$	$\overline{N}_{\mathrm{ci}}$	$\overline{N}_{\mathrm{cf}}$	$\overline{N}_{\mathrm{nt}}$
	$q_{Bt} = 0$					$q_{Bt} = 0.15$				
CUBE2	87.5	34.8	15.1	15.1	21.3	87.6	34.4	14.8	14.8	21.0
CYLINDER2	180.0	592.6	299.6	20.6	26.8	99.9	47.4	21.5	17.4	23.5
COMP1	89.9	296.1	134.5	18.6	25.1	81.3	45.6	20.4	16.4	22.6

The selection of the threshold value q_{Bt} directly influences the efficiency of this method. Increasing its value results in a larger number of the additional nodes. The actual computational effect may depend on the given mesh characteristics, the value of this coefficient should be higher for more anisotropic meshes. In Table 5.2 there are gathered results of generation of a number of example meshes with different value of q_{Bt}. The value $q_{Bt} = 0.15$ was selected as default in this work.

5.3.2 Localization of Containing Tetrahedron

In order to find the tetrahedron K_c containing the new vertex V a procedure of oriented traversing through the mesh elements is applied. Starting with some element $K_0 = K_s$, each subsequent tetrahedron K_{i+1} is selected as one of the tetrahedra adjacent to K_i. The selection is guided by direction of vertex V, i.e. the chosen tetrahedron is adjacent through face F_j for which

$$\max_{j=1,\ldots,4} \langle \mathbf{n}_j, \mathbf{x}_j \rangle \ , \tag{5.1}$$

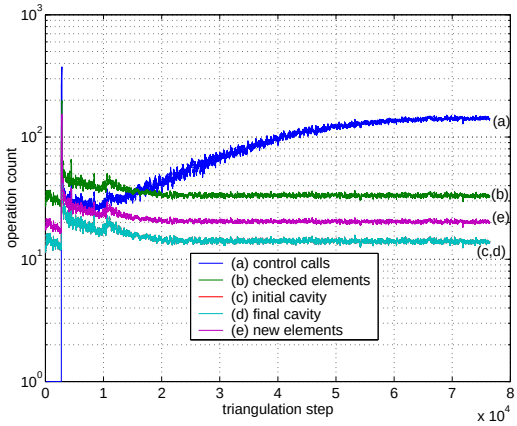
where $\mathbf{n}_j$ is a normal vector for F_j and $\mathbf{x}_j$ is a normalized vector $\overrightarrow{P_j V_i}$ with P_j being the middle of F_j. In general case such localization procedure visits on average $O(\sqrt{n})$ elements. In order to reduce this cost an additional octree structure is maintained responsible for selection of good starting tetrahedron K_s.

Each leaf of this octree contains a reference to tetrahedron which is used as a starting element for each vertex falling into this leaf. Each tetrahedron created during the procedure of boundary triangulation is sift through this octree and if it is better than current starting tetrahedron for some leaf (i.e. is closer to the middle of the leaf), the reference is accordingly updated. Each sift tetrahedron increases counters in all visited leaves of the octree. If the counter for some leaf reaches the predefined threshold value, the leaf is split. The updating of octree is necessary also for removed tetrahedra, which have to be removed from the 'best element' references in leaves. However, the counters are not decreased.

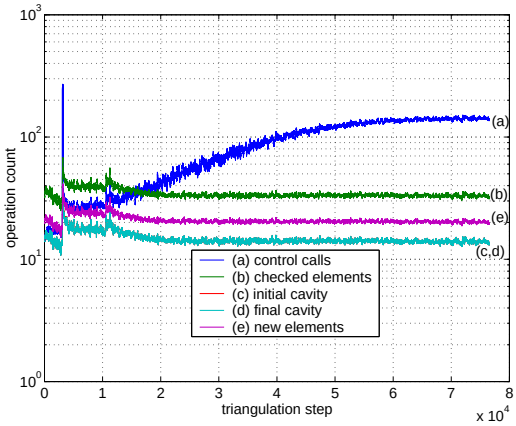
The tetrahedron is sift through the octree by selecting one or more points representing this element. The following sets of representative points for a tetrahedron were inspected: (a) middle of the element, (b) all vertices of element, (c) middle points of all edges of the element, (d) middle points of all faces of the element, and (e) the complete set of points (a)–(d). The average numbers of traversed tetrahedra per each retriangulated point are illustrated on Fig. 5.4 for example meshes CUBE2, CYLINDER2 and COMP1.

The average cost of traversing is significantly lower when using the auxiliary refinement modification of boundary nodes insertion. The influence of selected set of representative points for a tetrahedron is consistent for all presented meshes. The more points are used to represent the element, the better starting tetrahedra can be found for leaves of octree and the less elements are traversed per average node inserted into mesh. However, enriching the set of representative points increases also the size of the octree structure. The update of the octree takes also more time, since each representative point has to be sift through the octree.

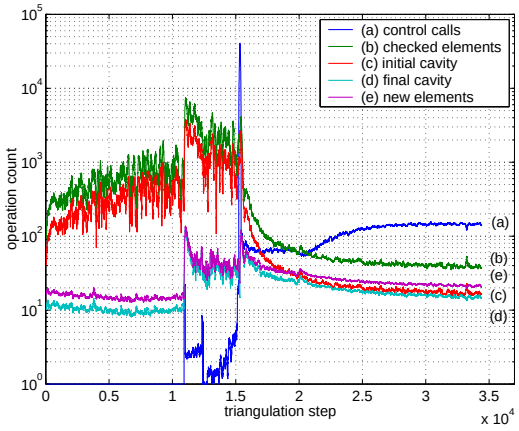
Another parameter influencing the efficiency of the overall procedure is the threshold for splitting of octree leaves. Higher value reduces size of the octree structure, but may also



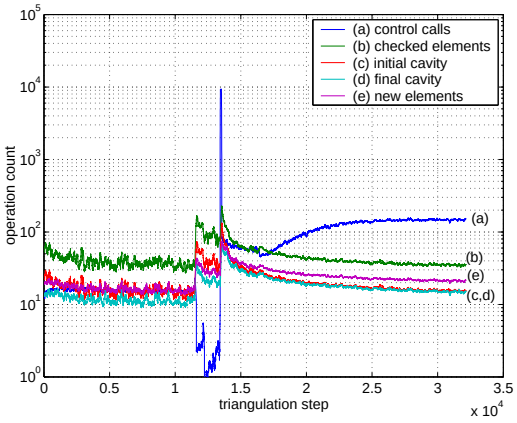
(a) CUBE2, $q_{Bt} = 0$



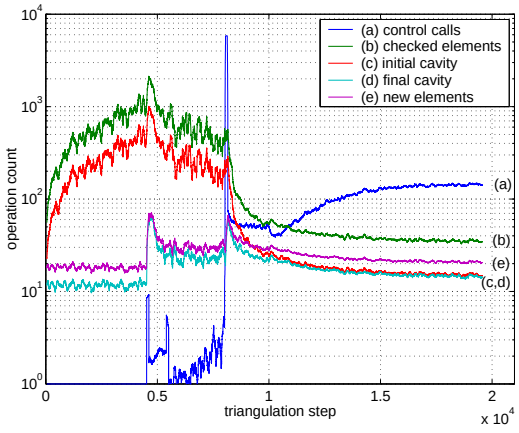
(b) CUBE2, $q_{Bt} = 0.15$



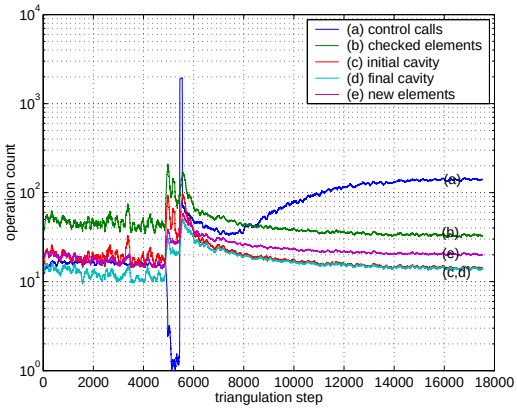
(c) CYLINDER2, $q_{Bt} = 0$



(d) CYLINDER2, $q_{Bt} = 0.15$



(e) COMP1, $q_{Bt} = 0$

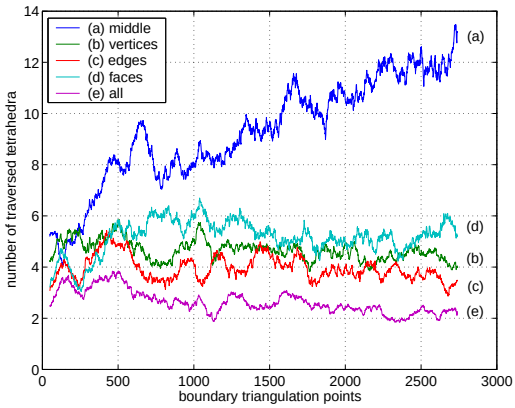


(f) COMP1, $q_{Bt} = 0.15$

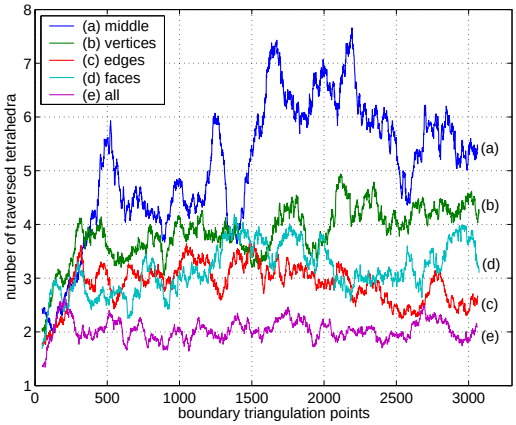
Figure 5.3: Statistics for retriangulation operation (100-moving average, with and without auxiliary boundary refinement)

Table 5.2: Statistics of boundary triangulation with refinement (q_{Bt} is the refined tetrahedra quality threshold, NP_r – the number of auxiliary nodes, NP – total number of nodes in created mesh, E_m,F_m – percent of missing edges/facets, t_b – time of unconstrained boundary triangulation, t_c – time of constraining, and t_{3d} – total time of 3D mesh generation)

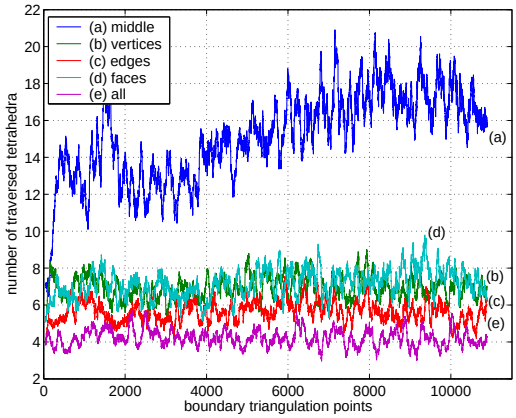
mesh	q_{Bt}	NP_r	NP $[10^3]$	$E_m[\%]$	$F_m[\%]$	$t_b[s]$	$t_c[s]$	$t_{3d}[s]$	$\overline{\mu}_{\mathcal{M}}$
CONE1	0	0	3.2	2.0	5.1	0.25	3.35	4.7	0.994
	0.1	48	3.2	1.6	4.6	**0.19**	0.32	1.6	0.997
	0.2	68	3.1	1.6	4.3	0.20	0.13	**1.4**	1.005
	0.3	87	3.1	**1.1**	**3.1**	0.20	**0.10**	**1.4**	**1.002**
ELLIPSOID	0	0	2.0	**2.5**	7.1	0.19	2.01	3.0	0.981
	0.1	52	1.9	2.9	7.9	0.16	1.59	2.5	0.993
	0.2	63	2.0	2.6	**6.9**	0.14	**0.99**	**1.9**	0.989
	0.3	80	1.9	2.8	7.6	**0.10**	1.46	2.3	**1.001**
SPIRAL	0	0	20.4	4.3	10.4	5.07	75.15	91.4	0.979
	0.1	402	20.1	2.5	6.5	2.44	28.25	41.3	0.986
	0.2	511	20.0	**2.0**	**5.3**	**1.70**	28.01	39.8	0.987
	0.3	525	20.0	2.3	6.0	1.80	**25.06**	**37.2**	**0.990**
TORUS2	0	0	10.1	3.5	9.0	1.19	23.87	29.5	0.988
	0.1	230	10.0	2.6	7.1	0.64	**8.07**	**12.7**	0.994
	0.2	290	10.0	2.9	7.6	**0.57**	12.68	17.3	0.993
	0.3	327	9.9	**2.0**	**5.5**	0.58	8.83	13.4	**0.996**
COMP2	0	0	8.0	3.0	7.9	0.90	10.55	14.9	**1.000**
	0.1	153	7.9	**1.9**	**4.9**	0.51	**4.69**	**8.4**	1.011
	0.2	190	7.8	**1.9**	**4.9**	**0.46**	7.38	10.9	1.008
	0.3	213	7.9	2.1	5.4	0.49	5.68	9.3	1.006
CUBE2	0	0	76.4	0.0	0.0	**0.38**	**0.10**	**35.0**	**1.024**
	0.1	243	76.5	0.0	0.0	0.45	0.11	38.1	1.027
	0.2	373	76.4	0.0	0.0	0.47	0.11	38.1	1.026
	0.3	466	76.3	0.0	0.0	0.49	0.13	37.9	1.026
CYLINDER2	0	0	31.8	4.2	9.6	7.82	420.27	450.9	**0.984**
	0.1	516	30.6	2.0	5.1	**2.08**	42.98	61.3	0.980
	0.2	680	30.6	1.8	4.6	2.10	41.90	60.3	0.982
	0.3	743	30.5	**1.7**	**4.5**	2.09	**21.54**	**39.9**	0.981
COMP1	0	0	17.8	3.9	9.7	2.16	54.41	65.6	**1.005**
	0.1	302	17.1	1.3	3.6	0.80	7.01	15.2	1.017
	0.2	377	17.1	1.0	2.8	**0.79**	7.67	15.9	1.016
	0.3	431	17.1	**0.9**	**2.6**	0.80	**3.19**	**11.3**	1.018



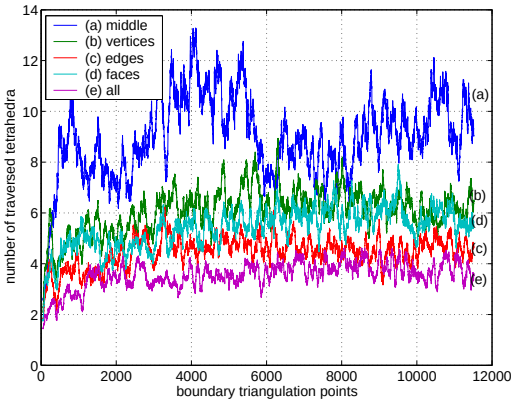
(a) CUBE2, $q_{Bt} = 0$



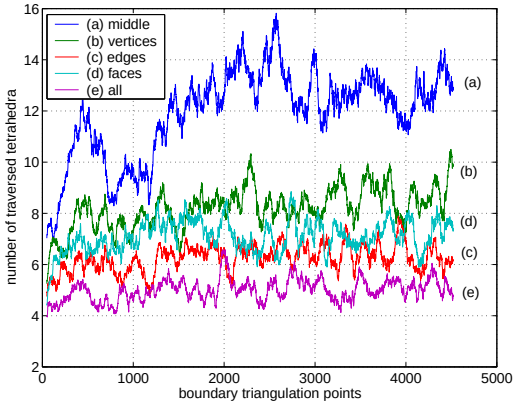
(b) CUBE2, $q_{Bt} = 0.15$



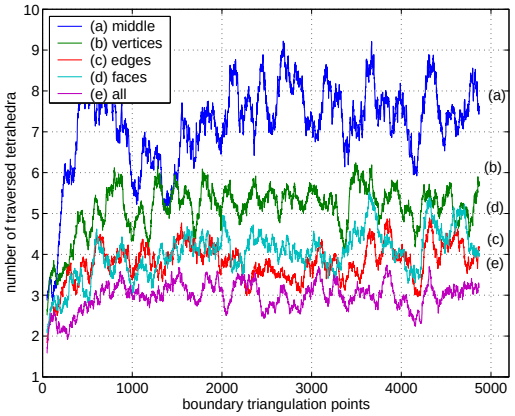
(c) CYLINDER2, $q_{Bt} = 0$



(d) CYLINDER2, $q_{Bt} = 0.15$



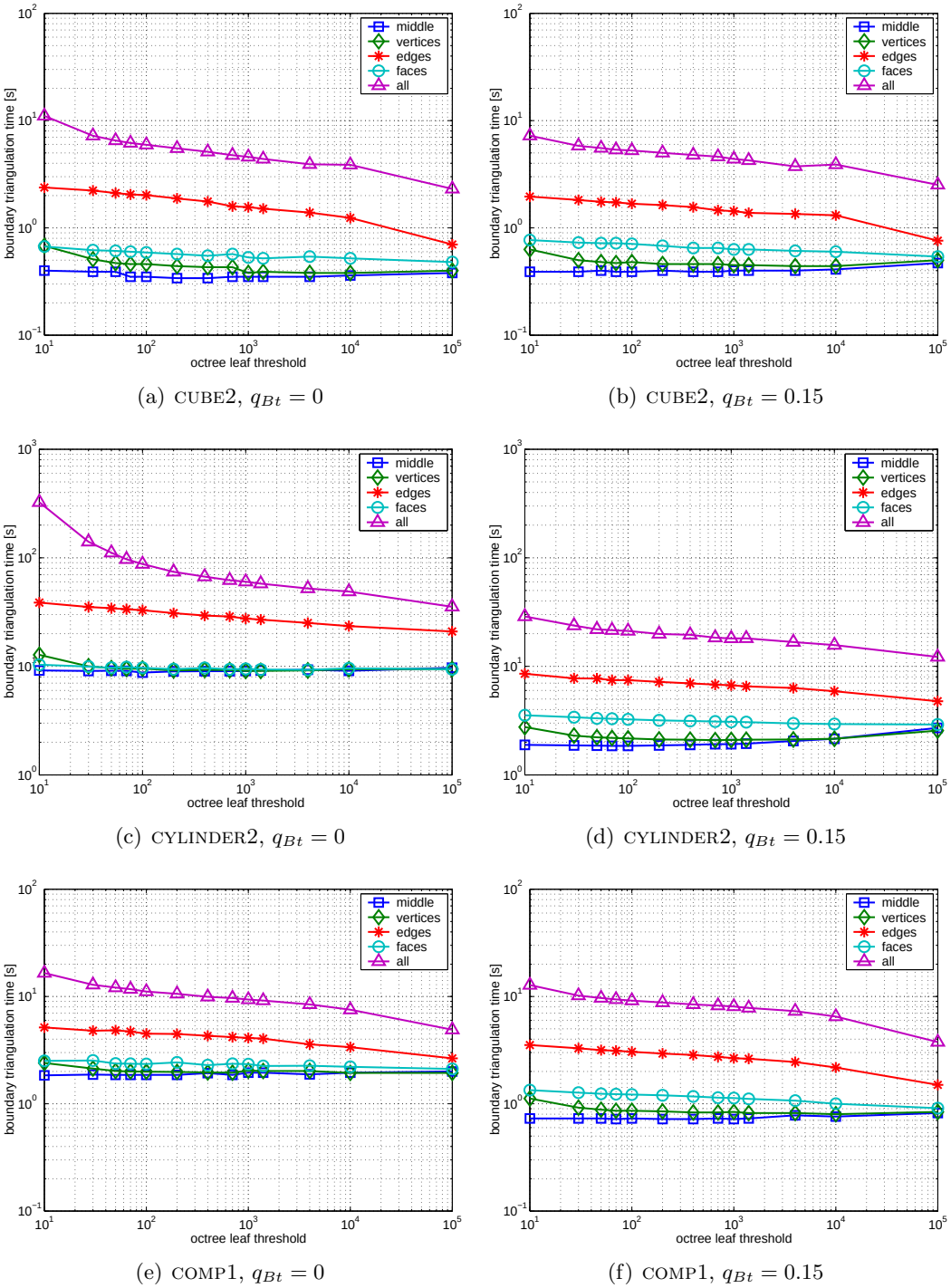
(e) COMP1, $q_{Bt} = 0$



(f) COMP1, $q_{Bt} = 0.15$

Figure 5.4: Statistics for localization of containing tetrahedron (100-moving average of number of the traversed tetrahedra)

increase the expected number of traversed tetrahedra (since each leaf covers more elements). Figure 5.5 shows the measured time of boundary nodes triangulation for different sets of representative points per element and different values of octree leaf threshold.



(a) CUBE2, $q_{Bt} = 0$ (b) CUBE2, $q_{Bt} = 0.15$

(c) CYLINDER2, $q_{Bt} = 0$ (d) CYLINDER2, $q_{Bt} = 0.15$

(e) COMP1, $q_{Bt} = 0$ (f) COMP1, $q_{Bt} = 0.15$

Figure 5.5: Influence of containing tetrahedron localization on boundary triangulation time

The best results were obtained for representing tetrahedra with a single point (middle

of the element) and this method has been selected as default in this work together with the default octree leaf threshold value equal to 1000.

5.4 Constraining of Boundary Triangulation

After all points are inserted into the mesh, the constraining procedure is used for enforcing all edges and facets from the boundary description in the created mesh. At the same time elements adjacent to boundary facets are identified as belonging to meshed domain or not. After constraining this information is spread to other elements (through adjacent elements but not crossing the boundary faces). Finally, the unmarked tetrahedra are removed from the mesh together with all unconnected points which may have been left by this operation.

The constraining procedure starts with checking if all edges and faces from the original boundary surface mesh are properly represented in the created mesh. All existing entities have ascribed the proper boundary conditions and volume data for adjacent blocks. The other entities are gathered for further postprocessing in two lists – for missing edges and facets, separately.

In this work an algorithm is proposed to recover the missing entities, which combines several heuristics. The algorithm starts with some basic mesh transformations, which can solve many of the cases. Subsequent phases introduce additional (usually more time consuming or intrusive) operations, which allow to recover more difficult cases. Each phase starts with recovering edges, and then faces. The phase is repeated until there are no further changes respecting the number of recovered edges and faces. Table 5.3 presents the order, in which the described methods of boundary recovery are applied.

Table 5.3: Recovery methods

phase	edge recovery	face recovery
1	PIPE-REDUCTION (Sec. 5.4.1)	PIPE-REDUCTION VICINITY-1 (Sec. 5.4.3)
2	PIPE-REDUCTION	PIPE-REDUCTION VICINITY-2 (Sec. 5.4.3)
3	PIPE-REDUCTION SPECIAL-METRIC-1 (Sec. 5.4.2)	PIPE-REDUCTION VICINITY-3 (Sec. 5.4.3)
4	PIPE-REDUCTION SPECIAL-METRIC-2 (Sec. 5.4.2)	PIPE-REDUCTION SPECIAL-METRIC-1
5	PIPE-REDUCTION BOUNDARY-POINTS (Sec. 5.4.4)	PIPE-REDUCTION BOUNDARY-POINTS

5.4.1 Pipe Reduction

For each missing edge a set of entities (edges, faces or blocks) crossing the missing edge (i.e. a pipe) is constructed (Fig. 5.6). All geometrical calculations for building the pipe are based on the *orient3d* and *orient2d* tests [178] in metric space. The metric transformation allows to use a single threshold value $\epsilon_{\mathcal{M}}$ for differentiating, whether the missing edge running through some tetrahedron is crossing one of its faces or one of its edges.

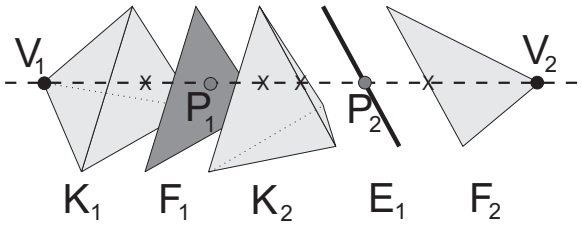


Figure 5.6: Recovery pipe for missing edge V_1V_2 (crossing through tetrahedron K_1 to face F_1, through tetrahedron K_2 to edge E_1 and through face F_2)

For each neighboring pair of crossed entities in the pipe the basic transformations: face-flip *T23*, edge-flip *T32* (Fig. 5.2) or co-planar edge-flip *T44* (Fig. 5.7) are used, if possible. Each successful transformation removes one of the crossing entities from the pipe.

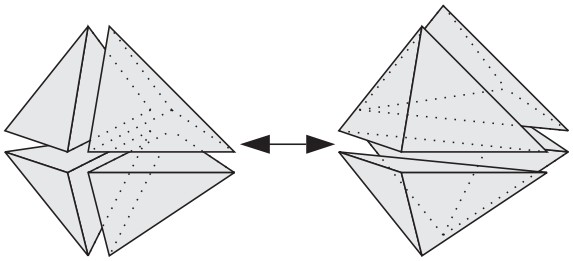


Figure 5.7: Co-planar edge-flip (T44)

5.4.2 Mesh Transformation with Special Metric

In this procedure all mesh blocks can be subjected to transformation in order to recover the missing entity (face or edge). The transformation process is guided by a special metric, artificially created for each missing entity in a way, which supports reconstruction of this entity (Fig. 5.8). In case of a missing edge E, such edge becomes the first eigenvector e_1 of the metric tensor with eigenvalue $\lambda_1 = ||E||$. The other two eigenvectors e_2, e_3 are selected as orthogonal to e_1 and to each other. The eigenvalues λ_2 and λ_3 are set as much smaller than λ_1 ($\lambda_2 = \lambda_3 = k\lambda_1$, where $k << 1$).

(a) edge (b) face

Figure 5.8: Special metric definition for recovery of missing mesh entity

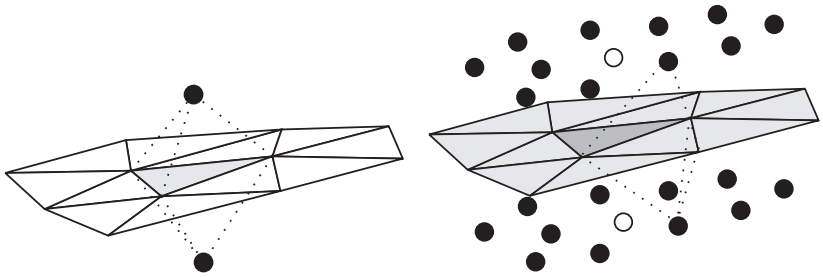
For face recovery. the two-dimensional metric is calculated from the face simplex, and the third direction e_3 is calculated as orthogonal to e_1 and e_2 with length $\lambda_3 = k\min(\lambda_1, \lambda_2)$.

The procedure starts with checking all tetrahedra adjacent to any vertex of the missing entity. Using the combination of edge- and face-flips, this fragment of the mesh is transformed according to the special metric. If the complete transformation of the selected mesh fragment will not result in recovering the missing entity, the mesh fragment is extended by the next layer of nodes and adjacent blocks. The procedure stops if the missing entity is successfully recovered, the maximum layer is reached or no more blocks can be transformed.

The difference between the SPECIAL-METRIC-1 and SPECIAL-METRIC-2 methods (Tab. 5.3) is the value of the maximum layer threshold, which is increased for SPECIAL-METRIC-2.

5.4.3 Vicinity Mesh Reconstruction

The mesh created for boundary points only may contain many tetrahedra of a rather low quality. Such tetrahedra would be removed after the constraining phase (exterior blocks) or would be refined during the phase of insertion of inner points (inner blocks). However, such tetrahedra may obstruct the recovery process, so this method tries to locally restore good quality of mesh near missing entities.



(a) missing face (b) adjacent faces

Figure 5.9: Inserting additional points in the vicinity of missing face

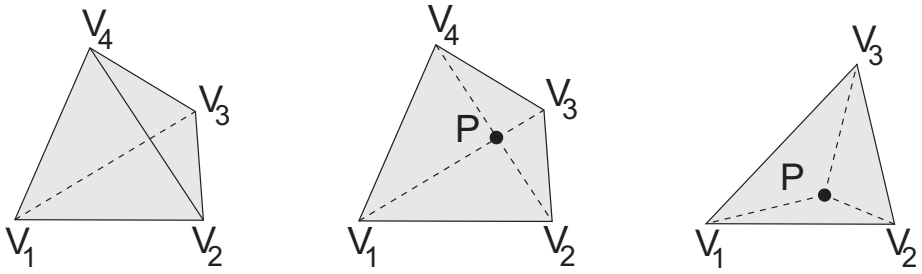
For each missing face, which can not be recovered because one of its edges is also missing, several additional points are inserted. The placement of such points is calculated using the metric (from CS) relevant for the given face. Insertion is cancelled if the new point is too close (in metric space) to some vertex already existing in the mesh or one of the missing faces or edges still to be recovered. The new points are inserted on both sides of the missing face (VICINITY-1, Fig. 5.9(a)) or all faces adjacent to the vertices of the missing face (VICINITY-2, Fig. 5.9(b)). Additional points can be also inserted at edges adjacent to vertices of the missing facet (VICINITY-3), found to be too long according to the local metric.

After the constraining phase is completed and all missing edges and facets are successfully recovered, these additional nodes are either removed (if placed outside the model) or repositioned (if they are placed within the domain).

5.4.4 Boundary Point Insertion

For edges and faces which could not be recovered by any other method, additional boundary points are inserted in places where the missing edges (faces) are crossing other mesh entities

(Fig. 5.10). If thus calculated point would be too close to one of the vertices of the missing entity, it is moved appropriately towards the middle of the entity.



(a) missing edge V_1V_3 (b) recovery of edge (c) recovery of face

Figure 5.10: Inserting additional boundary points

In case of edge recovery, after insertion of the new point P (and local retriangulation) the missing edge V_1V_3 (Fig. 5.10) is replaced by two edges V_1P and PV_3. Insertion of point P does not guarantee that these edges will be recovered directly. Instead, both edges are appended to the list of missing entities and set to be recovered in further steps. Additionally, each missing face adjacent to the edge V_1V_3 (e.g. face $V_1V_2V_3$) is also replaced by two new faces (e.g. V_1V_2P and V_2V_3P) and new edge (e.g. V_2P).

Similar procedure is used for recovery of faces, where after inserting a new point P within the face $V_1V_2V_3$ three new faces (V_1V_2P, V_2V_3P and V_3V_1P) and three edges (V_1P, V_2P and V_3P) have to be recovered.

Insertion of points at the surface boundary results in creation of mesh entities smaller than defined by the control space. For parallel mesh generation through the decomposition of the domain[179], surface meshes between two blocks meshed separately can not be modified at all. For this reason, an additional postprocessing procedure was implemented, which tries to move all additional boundary points out of the boundary in order to restore its original structure. For each additional boundary point there is made an attempt to move such point out of the boundary, creating empty cavity which can be then filled with additional tetrahedra.

5.4.5 Results

Table 5.4 presents statistics for efficiency of subsequent phases of boundary recovery for example meshes POLYHEDRON1 (Fig. A.9), CONE1 (Fig. A.12), CONE2 (Fig. A.13), CYLINDER1 (Fig. A.15), ELLIPSOID (Fig. A.14), TORUS1 (Fig. A.17), TORUS2 (Fig. A.18), TORUS3 (Fig. A.19), COMP1 (Fig. A.21), COMP2 (Fig. A.22), COMP3 (Fig. A.23), and COMP4 (Fig. A.24).

On average about 40% of missing edges and 60% of missing faces can be recovered in the first step of simple transformations. About 55% of missing edges and 35% of missing faces are recovered after insertion of additional points in the vicinity of the missing entities. The last, most time consuming and intrusive operations of recovery with special metric and insertion of points on the boundary is required for 5% or less of the missing entities. The

Table 5.4: Recovery statistics

mesh		entities		recovered at phase [%]				
		total	missing [%]	1	2	3	4	5
POLYHEDRON1	E	7104	1.7	28.3	60.8	9.2	0.0	1.7
	F	4736	4.7	90.1	4.1	4.5	0.0	1.4
CONE1	E	3090	2.6	32.5	25.0	32.5	2.5	7.5
	F	2060	6.2	59.8	6.3	25.2	2.4	6.3
CONE2	E	3999	5.2	42.2	34.0	21.4	0.0	2.4
	F	2666	13.1	68.9	12.3	16.0	0.0	2.9
CYLINDER1	E	1788	1.5	57.7	26.9	3.8	0.0	11.5
	F	1192	4.1	83.7	2.0	4.1	0.0	10.2
ELLIPSOID	E	2370	0.9	36.4	18.2	40.9	0.0	4.5
	F	1580	2.5	52.5	15.0	27.5	0.0	5.0
TORUS1	E	6426	3.5	38.3	21.2	36.5	0.5	3.6
	F	4284	8.7	52.2	17.2	25.8	0.5	4.3
TORUS2	E	8466	2.7	28.4	19.0	46.6	0.0	6.0
	F	5644	6.6	42.1	13.3	38.4	0.0	6.1
TORUS3	E	13401	4.2	39.6	25.3	33.2	1.1	0.9
	F	8934	10.5	59.5	16.9	21.3	1.3	1.1
COMP1	E	5829	3.8	29.7	21.6	43.7	0.9	4.1
	F	3886	8.9	46.4	16.5	31.3	1.2	4.6
COMP2	E	7335	2.7	40.6	21.3	34.0	1.0	3.0
	F	4890	7.0	56.2	14.4	24.7	1.2	3.5
COMP3	E	12147	4.0	40.3	26.5	30.2	0.6	2.3
	F	8098	10.5	60.0	12.8	23.9	0.7	2.6
COMP4	E	11115	4.5	39.3	25.3	29.9	1.4	4.2
	F	7410	11.1	57.1	12.9	24.8	1.7	3.5

actual cost of boundary recovery operation strongly depends on the geometric shape of the boundary, and also on quality, gradation and anisotropy of the surface mesh.

5.5 Insertion of Inner Nodes

After the boundary mesh is successfully constrained, it needs to be further refined according to the element sizing defined in the control space. The successive inner nodes are inserted in or near the tetrahedra which are too large or badly shaped, starting with the worst ones (Alg. 5.3). At the beginning of this procedure, there is calculated a quality coefficient for all tetrahedra in the mesh. The quality coefficient is used for guiding the order of refinement.

5.5.1 Evaluation of Tetrahedra for Refinement

The quality of tetrahedron K is calculated as:

$$q(K) = \frac{S_{\circ_I}}{S_{\circ_K}} \ , \tag{5.2}$$

```
1 calculate quality coefficients for all elements in T_h
2 organize all elements into heap structure with element K_0 with worst quality on top
3 while q(K_0) < q_t do
4 |   create new point P for refining of K_0
5 |   insert P into T_h with retriangulation
6 |   update heap structure
```

Algorithm 5.3: Refinement of tetrahedral mesh with inner nodes

where $S_{\circ_K}$ is the volume of the sphere circumscribed on the evaluated tetrahedron and $S_{\circ_I}$ is the volume of the circumsphere of the ideal tetrahedron (according to the local metric). In ideal *unit mesh* quality of all tetrahedra should be equal to 1. If $q(K) < 1$ then the tetrahedron requires refining by inserting an additional node in its vicinity. Calculation of circumsphere volume instead of taking the tetrahedron volume directly makes the quality coefficient sensitive for both size and shape discrepancy.

The procedure of mesh refinement stops when the quality of the worst element in the heap structure is already better than the selected quality threshold. In the current implementation the default value of quality threshold $q_t = 0.57$ is used for volume meshes, which produces final meshes with an average metric length of edges close to 1.

The local metric is retrieved from CS for barycenter of the tetrahedron selected for refinement. This metric is used for transformation of vertices for calculation of new node and for reverse-transformation of this node before insertion into mesh. Similarly as in the two-dimensional case, an additional condition is applied. If the quality $q(K)$ calculated in local metric from the barycenter of the tetrahedron K is higher than the quality threshold q_t (which would block further refinement of this element) the length of all edges is also evaluated. If the metric length of any edge (with metric calculated for the middle of an edge) is too high ($l_{\mathcal{M}}(E) > 1.5$) the quality coefficient $q(K)$ is set to $0.95q_t$ in order to enforce the refinement of this tetrahedron.

5.5.2 Placement of Refinement Points

The selected mesh element K is refined by inserting of a new node in its vicinity. Two methods of inner nodes placement for refined tetrahedron were inspected:

1. *Middle of the longest edge* (MLE). The length of edges is calculated in metric space and only non-boundary edges are considered. This method is very fast, since the containing element of the new node is always known directly. Unfortunately, the quality of mesh created using purely this method of determining coordinates of the new nodes is rather low.

2. *Center of circumscribed sphere* (CCS). This technique provides better quality of the final mesh but it requires a number of additional operations. The circumcenter coordinates are calculated in metric space, from transformed vertices of the tetrahedron and have to be transformed back for insertion. Since the circumcenter of a tetrahedron does not necessary lie within this element (especially at the beginning phase of

triangulation, where elements have rather low geometrical quality), the localization procedure of the containing tetrahedron may be required. At this phase of mesh generation the octree for starting tetrahedra is no longer used. Instead, the traversing of mesh begins from the element K being refined, which should be sufficiently near to the containing element. Since the triangulation at this phase is constrained, the searching for containing tetrahedron may fail (the refined and containing elements may be divided by some boundary face). In such case, the refinement procedure for this element is abandoned. The quality coefficient of K is adequately increased in order to move this element further down the heap structure and try again later, when local quality of the mesh will be improved. The insertion of new node may be also cancelled if one of the following conditions is true:

- the new node lies outside the meshing domain,
- the new node lies too close to one of mesh vertices,
- metric retrieved from CS for the new node is too different from the metric local to tetrahedron K (i.e. the refined tetrahedron would not be included into the retriangulation cavity if the metric at the new node is used).

In the proposed approach both methods of determination of inner nodes placement are used alternatively, depending on the quality of refined elements. At the initial phase of mesh refinement procedure, the MLE method is favored, since it allows to avoid a large number of unnecessary calculation and cancellation of insertions. After some threshold value q_{It} of improved tetrahedron quality is reached, the insertion method is switched to CCS in order to obtain better quality of the mesh elements at the final steps of inner nodes refinement.

In Table 5.5 there are gathered statistics of meshes (CUBE2, CYLINDER2 and COMP1) generated with different value of q_{It}. The meshes are compared with respect to number of inserted inner nodes, triangulation time, quality of obtained meshes and number of cancelled insertions. If $q_{It} \leq 0.0$ all inner nodes are inserted using CCS, for $q_{It} \geq q_t$ all nodes are inserted with MLE. For $0.0 < q_{It} < q_t$ both methods are used in some proportions.

Figure 5.11 illustrates the measured speed of the phase of inner nodes insertion for the example meshes and different values of the threshold value q_{It}. For isotropic mesh CUBE2 introduction of mixed method of determining placement of inner nodes does not give any advantages, since the best results are obtained with inserting new nodes using only method CCS. However, for anisotropic meshes CYLINDER2 and COMP1 the increase of meshing speed gained with using MLE in the early phases of inner nodes refinement is clearly visible. The actual value of threshold q_{It} is not that important for anisotropic cases, as long as it does not equals to extreme values (i.e. if both method of inner nodes placement determination are used).

The influence of the selected method of inner nodes placement on the quality of the obtained meshes is shown on Fig. 5.12. For pure MLE method the quality is clearly inferior for all tested meshes. Using mixed MLE+CCS method produces meshes with similar quality as using only CCS method.

Table 5.5: Inner nodes threshold q_{It} statistics, NPi – number of inner nodes, τ_i – speed of inner nodes triangulation, NPco – number of nodes outside the mesh bounding box, NPcc – number of nodes with no containing tetrahedron found, NPcm – number of nodes with too large metric difference, NPcp – number of nodes too close to other mesh vertices, and NPc – total number of cancelled nodes insertion

q_{It}	NPi $[10^3]$	$\tau_i[10^3/s]$	$\overline{\mu}_{\mathcal{M}}$	$\overline{\eta}_{\mathcal{M}}$	NPco	NPcc	NPcm	NPcp	NPc
CUBE2									
0.0	61.0	4.92	1.026	0.673	3 945	0	0	240	4 185
0.1	61.3	4.91	1.024	0.675	3 753	0	0	219	3 972
0.2	60.8	4.84	1.027	0.674	3 385	0	0	247	3 632
0.3	61.5	4.72	1.023	0.677	2 715	0	0	271	2 986
0.4	62.4	4.64	1.018	0.679	1 838	0	0	315	2 153
0.5	63.5	4.50	1.011	0.674	821	0	0	224	1 045
0.6	31.2	4.47	1.311	0.606	0	0	0	0	0
CYLINDER2									
0.0	15.5	2.07	0.976	0.753	5 881	1 384	9 935	10 069	27 269
0.1	15.5	2.88	0.978	0.747	1 926	882	2 728	6 306	11 842
0.2	15.2	2.92	0.984	0.744	1 364	724	1 148	4 813	8 049
0.3	15.1	2.96	0.985	0.745	953	581	483	3 559	5 576
0.4	15.0	2.94	0.986	0.744	583	433	181	2 258	3 455
0.5	15.0	2.90	0.989	0.740	270	187	46	998	1 501
0.6	7.5	2.19	1.147	0.640	0	0	0	0	0
COMP1									
0.0	9.9	2.52	1.019	0.824	1 501	2 900	2 026	2 835	9 262
0.1	9.9	3.91	1.017	0.826	368	1 508	274	1 487	3 637
0.2	9.9	3.90	1.018	0.824	296	1 342	145	1 165	2 948
0.3	9.9	3.93	1.018	0.826	212	1 091	80	847	2 230
0.4	9.9	3.88	1.017	0.826	130	768	31	611	1 540
0.5	10.2	3.81	1.011	0.823	58	396	7	251	712
0.6	4.6	3.02	1.219	0.765	0	0	0	0	0

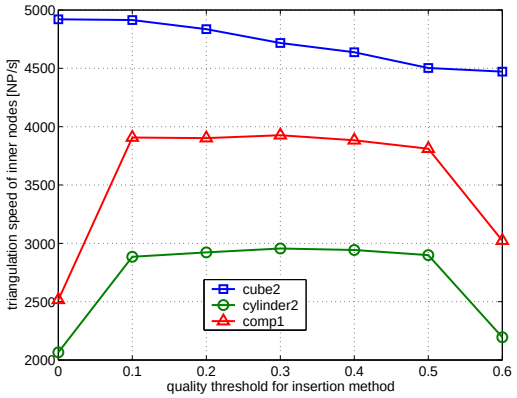
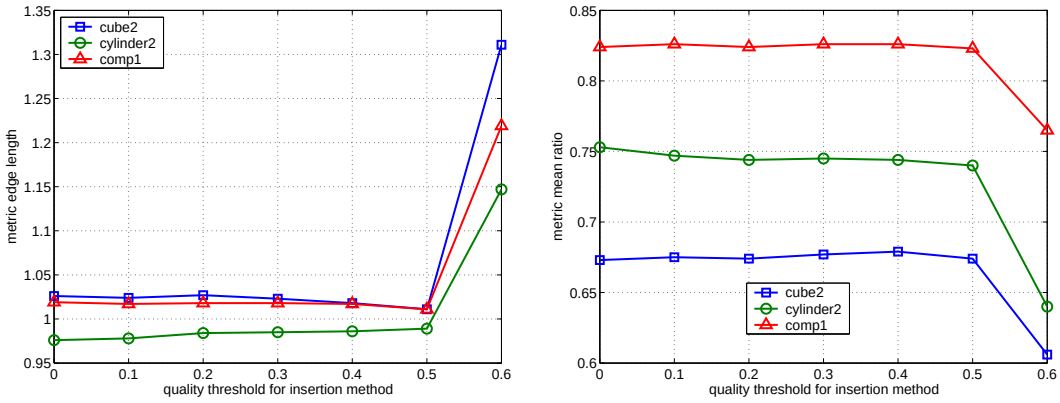


Figure 5.11: Influence of inner nodes insertion method on meshing time



(a) metric edge length (b) metric mean ratio

Figure 5.12: Influence of inner nodes insertion method on mesh quality

5.6 Improvement of Tetrahedral Mesh

After construction of the mesh, some postprocessing procedures can be applied in order to improve the quality of elements in the mesh. The most common techniques include topological (e.g. edge/face swapping, insertion of nodes, removal of nodes or elements) and geometric (relocation of mesh nodes by Laplace smoothing, optimization based or physics based) operations[47].

```
1 for i ← 1 to N_imp do
2     Laplace smoothing
3     geometric flipping of edges and faces
```

Algorithm 5.4: Tetrahedral mesh improvement

5.6.1 Laplace Smoothing

A three-dimensional version of Laplace smoothing (4.2) is used. All nodes from the mesh interior are inspected at an arbitrary order and each node is moved to barycenter of the set of incident nodes. The modified version of method (4.3) with metric calculated for each incident node separately is applied for smoothing of nodes, where the local gradation of metric is high.

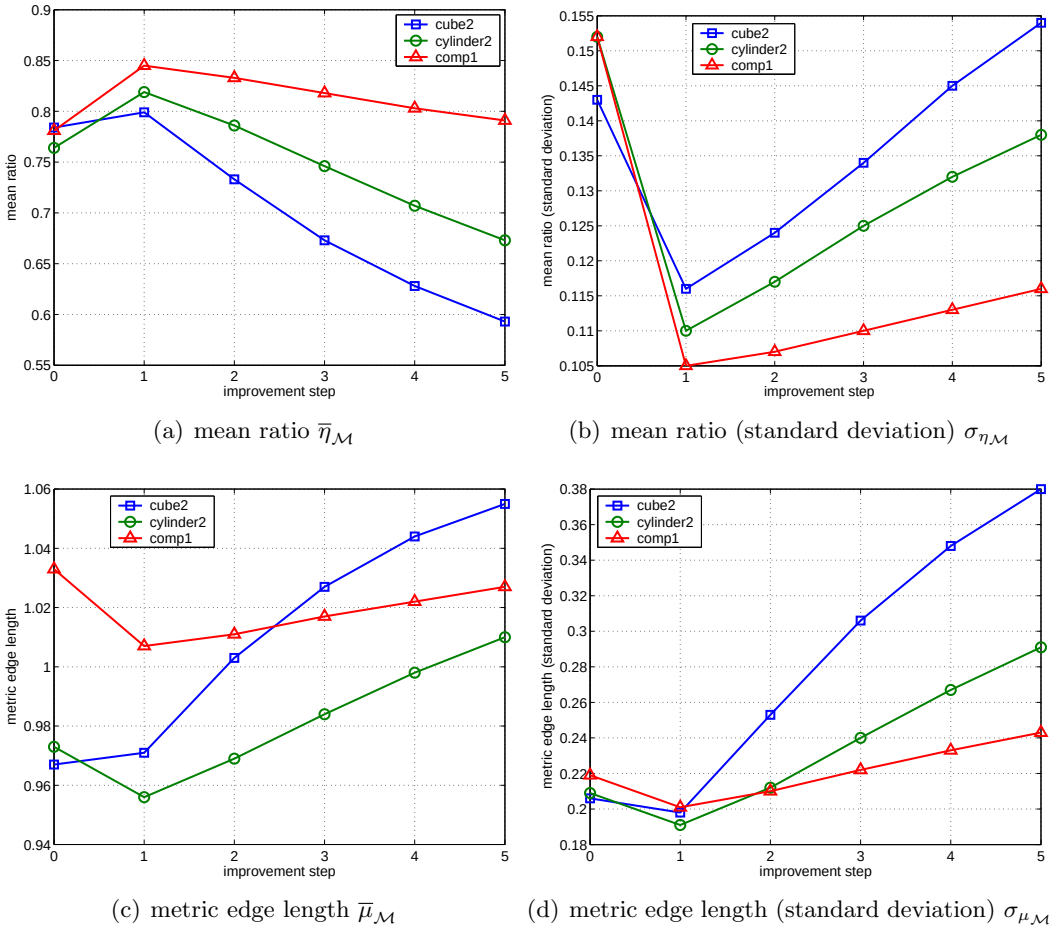
5.6.2 Geometric Flipping of Edges and Faces

All non-boundary edges and faces in the mesh are inspected and local reconfiguration of the mesh is applied whenever it would have an positive effect on the local quality of elements. The iterative procedure of edge- and face-flipping is based on Alg. 5.1. The only difference is the initial contents of the set of active vertices S_V, which in this case contains all vertices from the mesh. The condition of both edge- and face-flip transformations is based on mean ratio geometric quality of elements in metric space $\eta_{\mathcal{M}}(K)$. The transformation is applied

if the minimum quality of the set of tetrahedra created by transformation is higher than the minimum quality of initial tetrahedra. The number of edge-flips is typically much higher than the number of face-flips, which has a side effect of reducing the total number of tetrahedra in the mesh.

5.6.3 Results

Table 5.6 presents the results of improvement process for several iterations steps ($N_{imp} = 5$) for example meshes CUBE2, CYLINDER2 and COMP1 (Fig. 5.13).



(a) mean ratio $\overline{\eta}_{\mathcal{M}}$

(b) mean ratio (standard deviation) $\sigma_{\eta_{\mathcal{M}}}$

(c) metric edge length $\overline{\mu}_{\mathcal{M}}$

(d) metric edge length (standard deviation) $\sigma_{\mu_{\mathcal{M}}}$

Figure 5.13: Quality improvement of example meshes

The first step of the implemented local heuristics produces the desired effect of increasing the quality of the mesh. However, further application of these methods for graded and anisotropic tetrahedral meshes not only fails to improve but may actually degrade the average quality of elements in the mesh.

Table 5.6: Influence of successive improvement iterations on tetrahedral mesh quality, NT – number of tetrahedra, t_i – time of single improvement iteration

step	NT $[10^3]$	$\overline{\eta}_{\mathcal{M}}$	$\sigma_{\eta_{\mathcal{M}}}$	$\overline{\mu}_{\mathcal{M}}$	$\sigma_{\mu_{\mathcal{M}}}$	$t_i[s]$
CUBE2						
–	393.1	0.784	0.143	0.967	0.206	–
1	366.0	0.799	0.116	0.971	0.198	8.6
2	365.9	0.733	0.124	1.003	0.253	7.4
3	366.2	0.673	0.134	1.027	0.306	7.4
4	366.9	0.628	0.145	1.044	0.348	7.5
5	367.8	0.593	0.154	1.055	0.380	7.5
CYLINDER2						
–	138.0	0.764	0.152	0.973	0.209	–
1	128.5	0.819	0.110	0.956	0.191	2.7
2	128.4	0.786	0.117	0.969	0.212	2.4
3	128.4	0.746	0.125	0.984	0.240	2.4
4	128.4	0.707	0.132	0.998	0.267	2.4
5	128.5	0.673	0.138	1.010	0.291	2.4
COMP1						
–	79.4	0.781	0.152	1.033	0.219	–
1	74.2	0.845	0.105	1.007	0.201	1.2
2	74.1	0.833	0.107	1.011	0.210	1.1
3	74.1	0.818	0.110	1.017	0.222	1.1
4	74.1	0.803	0.113	1.022	0.233	1.1
5	74.1	0.791	0.116	1.027	0.243	1.1

5.7 Chapter Summary

The generation of three-dimensional tetrahedral meshes was described. The general algorithm is similar to the two-dimensional case, the incremental insertion algorithm is also used for volume meshing. However, the extension of this algorithm is not straightforward – some aspects like boundary recovery, which can be easily done in 2D, are becoming a serious challenge in three dimensions. A multi-step procedure coping with this problem was proposed in order to avoid the necessity of inserting additional nodes at the domain boundary. A concept of systematic introduction of auxiliary nodes during the boundary mesh nodes triangulation was proposed, which allowed to significantly increase the efficiency of the triangulation process.

Chapter 6

Mesh Generator Architecture

6.1 Introduction

When creating meshes containing very large number of elements, the computational complexity and the memory requirements of the generator can prove to be a very important factor. Several data structures to represent and manipulate unstructured meshes have been published over the years[180, 181]. Such structures usually offer efficient solutions to represent and recover mesh topology and modify its geometric parameters. In this chapter some aspects regarding the organization of the unstructured meshes for the developed mesh generator are described.

In recent years some approaches to object-oriented design of general meshing framework have been introduced. The Mesh Toolkit (MSTK) [182] utilizes many object-oriented programming principles but is implemented in C for efficiency. The MSTK is designed to allow general combinations of entities and adjacencies, some commonly used representations are encoded directly. The Algorithm Oriented Mesh Database (AOMD) [183] also provides the ability to build any set of mesh adjacencies, starting from a well defined minimum set, together with access iterators for mesh entities and their adjacencies. The Sandia Mesh Database Component (MDB)* allows flexibility only in the types of stored adjacencies. There are also available a number of other mesh representation libraries oriented on specific types of meshes (mostly triangular surfaces) e.g. GNU Triangulated Surface Library (GTS)†, OpenMesh‡ or Grid Algorithms Library (GrAL)§.

6.1.1 Contents

Sec. 6.2 provides the description of geometric and topological representation of the modeled domain. The hierarchy of control space structures and interfaces is presented in Sec. 6.3. The selected mesh representation (used mesh entities and theirs interconnections) and its

*http://sass1693.sandia.gov/cubit/mdb.htm
†http://gts.sourceforge.net/
‡http://www.openmesh.org/
§http://www.math.tu-cottbus.de/~berti/gral/

management during the discretization process is described in Sec. 6.4 together with a set of elementary accessing and modifying operations. The prediction of final mesh size is presented in Sec. 6.5. The statistics and analysis of computational and memory complexity of the implemented mesh generation procedures is included in Sec.6.6.

6.2 Description of Model Geometry and Topology

The domain is defined as following the BREP (Boundary Representation) model. A hierarchical approach is used, where surface patches are defined by an underlying surface description and a number of segments defining contours of each patch. Closed set of patches defines a volume block. The modeled domain can consist of many sub-domains, some of them may be declared as 'empty' in order to define cavities.

A valid domain specification has to conform to few topological constraints:

1. Entities (edges and patches) can not cross themselves.
2. Each edge has to be incident to two different vertices (no loops allowed).
3. Any two nodes can be connected by at most one edge.

The validity of the domain has to be verified in the mesh generator before the meshing process can be started.

A number of parametric representations of surfaces and curves has been implemented, which can be used in BREP description of the modeled domain.

6.2.1 Parametric Surfaces

The modeled domain is defined by a set of surface patches $\{\mathbf{p}_i\}$ with parametric formulation (2.2). For other surface representation (e.g. facet, voxel or implicit forms) an additional layer of parameterization is required.

The parametric surface can be defined using an analytic formulation. They can be also modeled using surfaces of revolution, surfaces of extrusion or free-form surfaces (e.g. NURBS).

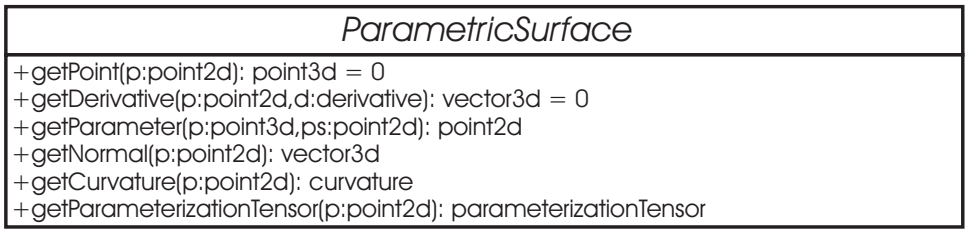


Figure 6.1: Parametric surface representation

The base class `ParametricSurface` (Fig. 6.1) defines the common interface for all implementations of parametric surfaces. Two abstract methods have to be provided in derived classes:

- *getPoint(point2d)* – returns three-dimensional point on surface in any point $P \in \mathcal{S}$ within the parametric space domain.
- *getDerivative(point2d, derivative)* – returns three-dimensional vector of the requested partial derivatives in any point $P \in \mathcal{S}$. The first order derivatives are required for the meshing procedures. The second order derivatives are optional and they are utilized only by auto-sizing methods based on calculation of surface curvature.

Additional set of basic methods is also defined on a general level, the selected operations may be overridden in derived classes in cases, where a more efficient procedures are available for specific surface representation (e.g. for plane).

- *getParameter(point3d, point2d)* – returns parametric coordinates of a three-dimensional point projected onto the surface. In all applications of this method in the generator a good starting point can be provided. The solution is found using a conjugate gradient method.
- *getNormal(point2d)* – returns normal vector at the given point $P \in \mathcal{S}$ of the surface (calculated as orthonormal to first derivatives at this point).
- *getCurvature(point2d)* – calculates main curvatures of surface (assuming availability of second order derivatives) at the given point $P \in \mathcal{S}$, using procedure described in Sec. 2.4.2.
- *getParameterizationTensor(point2d)* – returns the parameterization tensor at the given point $P \in \mathcal{S}$ using procedure described in Sec. 2.2.1.

6.2.2 Parametric Curves

Each surface patch of the modeled domain is constrained by a set of curvilinear contours $\{\mathbf{c}_{\mathbf{p}_i}\}$ with parametric formulation (2.44). All curves are defined in parametric space of some surface.

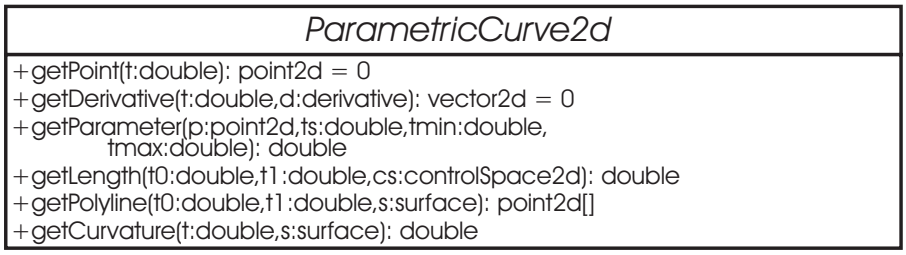


Figure 6.2: Representation of parametric curves on surfaces

The base class `ParametricCurve2d` (Fig. 6.2) defines the common interface for all implementations of parametric curves on surface. Two abstract methods have to be provided in derived classes:

- *getPoint(t)* – returns the two-dimensional point at curve for the given parameter $t \in \mathcal{C}$.

- *getDerivative(t, derivative)* – returns two-dimensional vector of the requested partial derivatives for any parameter $t \in \mathcal{C}$. The first order derivatives are required for the meshing procedures. The second order derivatives are optional and they are utilized only by auto-sizing methods based on calculation of contour curvature.

Additional set of basic methods is also defined on a general level, the selected operations may be overridden in derived classes in cases, where a more efficient procedures are available for specific representation (e.g. for straight line).

- *getParameter(point2d, ts, tmin, max)* – returns parameter for a two-dimensional point projected onto the curve. A combination of bracketing, bisection and Newton method is used to find the solution. In all applications of this method in the generator a good starting point can be provided. In some cases the limiting range $[t_{\min}, t_{\max}]$ is also helpful.

- *getPolyline(t0, t1, surface)* – returns the polyline approximation of curve segment between parameters t_0 and t_1, The approximation may also take into account the underlying surface.

- *getLength(t0, t1, cs)* – returns approximated length of a cuver segment between parameters t_0 and t_1. If the control space parameter is provided, the length is calculated in metric space.

- *getCurvature(t, surface)* – calculates curvature of three-dimensional contour (assuming availability of second order derivatives) for the given parameter t, using procedure described in Sec. 2.4.3.

During volume meshing a three-dimensional curve representation is also used with parametric formulation (2.43).

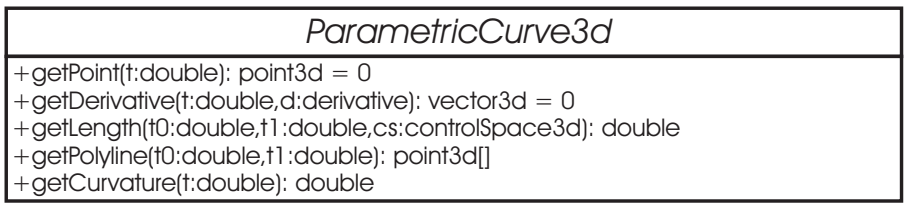


Figure 6.3: Representation of 3D parametric curves

The base class `ParametricCurve3d` (Fig. 6.3) defines the common interface for all implementations of three-dimensional parametric curves. Two abstract methods have to be provided in derived classes:

- *getPoint(t)* – returns the three-dimensional point at curve for the given parameter $t \in \mathcal{C}$.

- *getDerivative(t, derivative)* – returns three-dimensional vector of the requested partial derivatives for any parameter $t \in \mathcal{C}$.

The additional set of methods is defined similar to those described in two-dimensional version of parametric curves. The 3D curves can be defined through direct formulation or indirectly, using a two-dimensional curve together with parametric surface description.

6.3 Control Space Hierarchy

The adaptive control space concept described in Chapter 3 is implemented using a hierarchy of classes for representation of different control space structures.

6.3.1 Surface Meshing

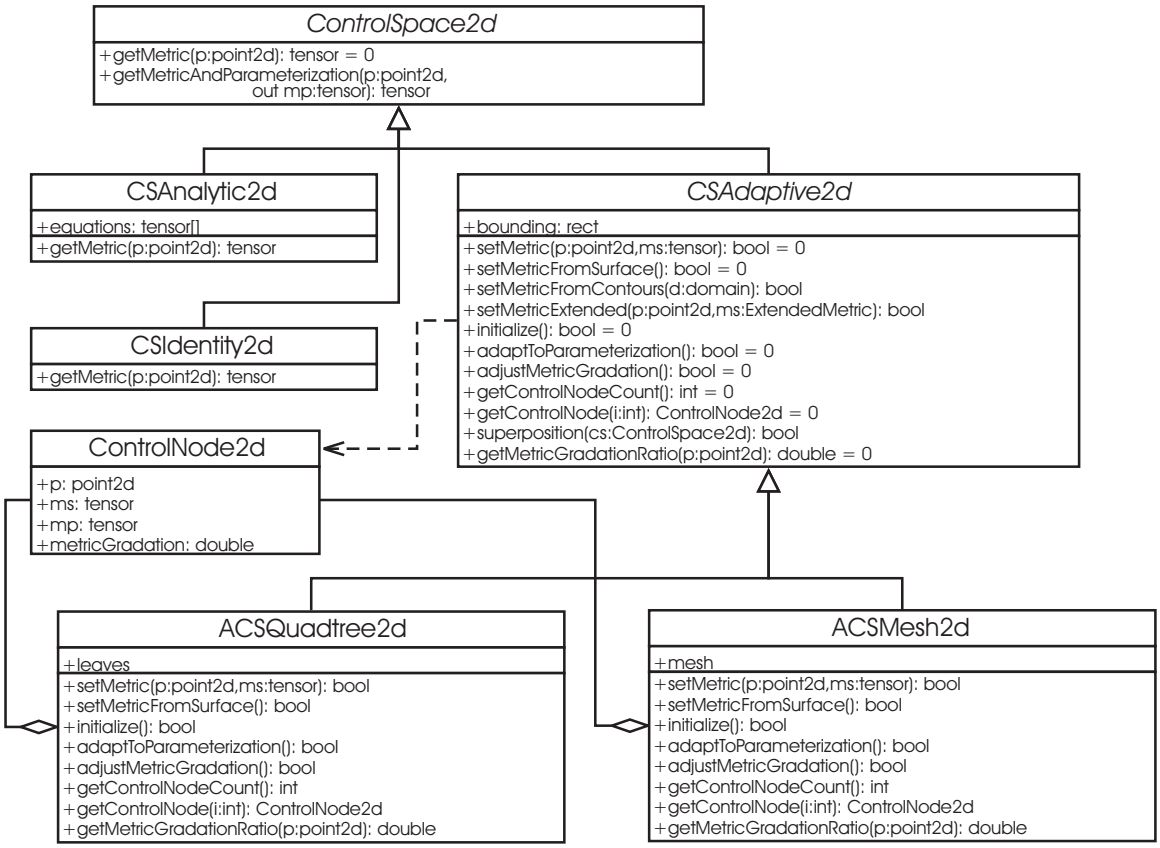


Figure 6.4: Control space hierarchy for surface meshing

Most basic of the hierarchy (Fig. 6.4), abstract `ControlSpace2d` class defines the essential functionality of a control space, which is to return the required metric transformation matrix at any given point within the domain being discretized. This functionality must be provided by each derived class.

Analytic formulation

Two simple non-adaptive classes are directly derived from `ControlSpace2d`:

CSAnalytic2d – storing metric data prescribed as one or more functions f_i within defined domain. This class is used for representation of metric description given by user in an analytical form. At least one global function f_0 is required, defining metric in the whole meshing domain. Additional functions f_i may be also defined to allow for more precise metric specification in some sub-domains. For each point within the valid domain the metric value can be evaluated directly, without any interpolation. The cost of its utilization depends on the complexity of given functions f_i. Since no further adaptation of this control space form is possible, it is usually used as an intermediate representation of some metric source, which is being superposed during the procedure of control space adaptation.

CSIdentity2d – defining simple control space used internally. This control space, returning Euclidean metric tensor throughout the whole domain is used for operations on any auxiliary mesh (like the background mesh control space structure) which is necessary during the generation of the actual mesh.

Abstract adaptive control space

In order to facilitate creation and utilization of more complex structures an abstract class `CSAdaptive2d` was created, defining a set of required methods. Any derived class should be able to initialize the control space from either continuous (e.g. curvature of surface or analytical equations) or discrete sources (e.g. discrete nodes or segments with metric), interpolate or extrapolate metric in case of insufficient data, adjust metric in any subarea of initialized structure and adapt its structure to variation of metric or parameterization.

Adaptive structures

Two adaptive structures of control space were implemented (Fig. 3.1):

CSQuadtree2d – using balanced quadtree grid (Sec. 3.3),

CSMesh2d – storing metric in nodes of a background mesh (Sec. 3.4).

6.3.2 Volume Meshing

General hierarchy of control space classes for volume meshing is very similar to the two-dimensional case (Fig. 6.5). The main differences are the dimensions of node coordinates and stored metric transformation tensor. Also the parameterization matrix is no longer required here, since the three-dimensional mesh generation does not use any parametric space.

6.4 Mesh Representation

The mesh is represented as a structure of topological entities (vertices, edges, faces and elements) and topological adjacencies (entity-interconnections, e.g. edge-vertex). The selection of the represented entities and theirs interconnection is motivated by the diversity

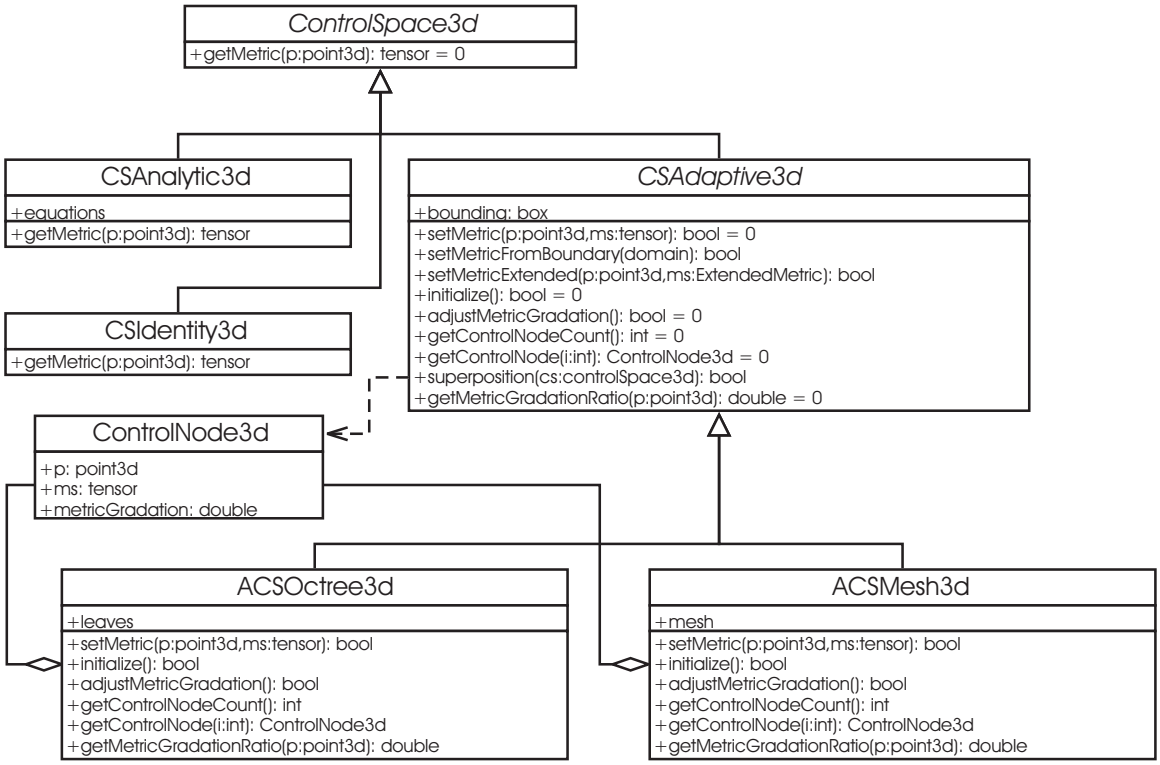
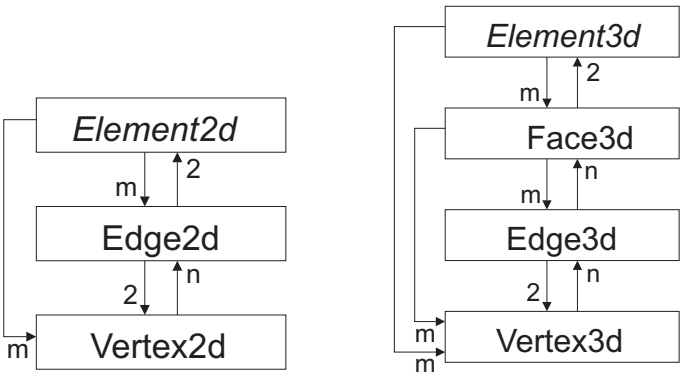


Figure 6.5: Control space hierarchy for volume meshing

of employed procedures, which can treat the mesh as a graph of vertices and edges, a graph of connected elements, etc. [181]. Some simple applications use only a minimal representation consisting of elements (triangles, tetrahedra, etc.) defined by points. In more complex applications the number of connections is greatly increased including e.g. adjacencies like element-element.

Figure 6.6 presents interconnections model selected in this work as a compromise between the efficiency of access and modification operations [184].



(a) surface mesh (b) three-dimensional mesh

Figure 6.6: Selected mesh representation

6.4.1 Mesh Entities

The implementation of main classes for mesh entities representation is briefly described. The enumeration of typical constructors and accessor/mutator methods is skipped for clarity.

Vertex2d

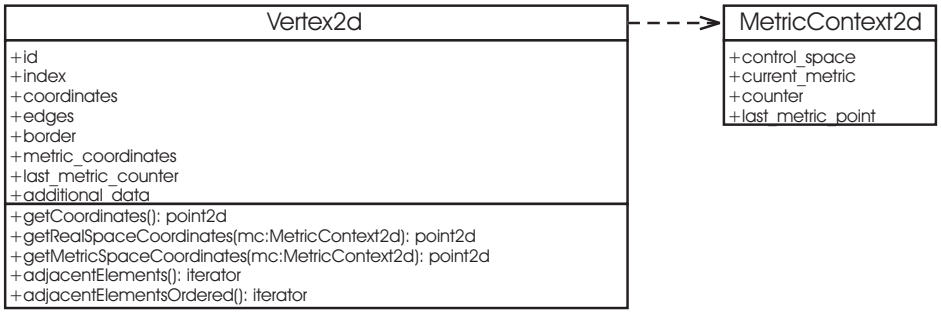


Figure 6.7: Two-dimensional mesh vertex representation

The class `Vertex2d` (Fig. 6.7) defines a mesh vertex for two-dimensional parametric meshing.

Main attributes and operations:

- `id` – a number used for identification of vertices (often following the information from model description),
- `index` – due to frequent operations of sequential scanning of vertices in the mesh, all vertices are stored in an indexed container,
- `coordinates` – two-dimensional geometric coordinates of the vertex in the parametric space,
- `edges` – list of all mesh edges (pointers) adjacent to this vertex (in arbitrary order),
- `border` – logical flag for differentiating vertices lying on boundary of the discretized model,
- `metric_coordinates` – in order to increase the efficiency of meshing operations and avoid unnecessary transformation of coordinates from parametric to metric space, in each vertex there are stored its coordinates transformed to metric space,
- `last_metric_counter` – information (counter of the recently used `MetricContext2d` reference), which allows to decide, whether the cached coordinates of this vertex transformed to metric space are valid, or if they need to be recalculated,
- `additional_data` – some meshing operations require one or more additional data to be stored in vertices (e.g. usage marks, references to projected three-dimensional vertices, weights, etc.),

- `getCoordinates()`, `getRealSpaceCoordinates(mc)`, `getMetricSpaceCoordinates(mc)` – this set of operations returns coordinates of the vertex in parametric space $\mathcal{S}$, transformed to $\mathcal{S}^{\mathcal{R}}$ space or transformed to metric space $\mathcal{S}^{\mathcal{M}}$ (using the provided `MetricContext2d` reference),
- `adjacentElements()` – returns an iterator for sequential accessing all elements adjacent to this vertex,
- `adjacentElementsOrdered()` – returns an iterator for accessing all elements and edges adjacent to this vertex in a given sequence (clockwise or counterclockwise).

The MetricContext2d reference is used throughout the meshing process to provide the information about currently used metric. After each update (recalculation of metric for some other point within the discretization domain) the current metric in metric transformation form is available in this object and the counter is increased.

Edge2d

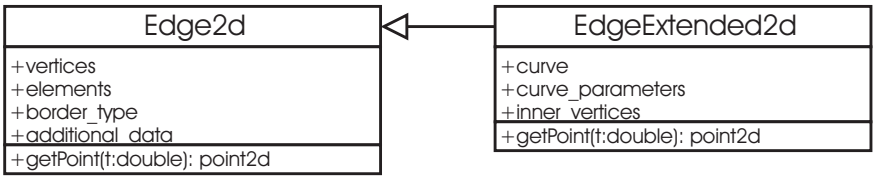


Figure 6.8: Two-dimensional mesh edge representation

The class `Edge2d` (Fig. 6.8) defines a basic mesh edge for two-dimensional parametric meshing. The derived class `EdgeExtended2d` is also provided for representing curvilinear or higher interpolation edges.

Main attributes and operations:

- `vertices` – an ordered pair of vertices (defining the direction of edge),
- `elements` – adjacent elements (at most two) also following the direction of edge,
- `border_type` – whether the edge is part of outer/inner boundary or a part of inner mesh,
- `additional_data` – depending on meshing operations, one or more additional data may have to be stored (e.g. front level, pointer to associated three-dimensional edge, usage side marks, etc.),
- `curve` – (`EdgeExtended2d`) pointer to parametric curve providing the shape information for this edge,
- `curve_parameters` – (`EdgeExtended2d`) parametric representation of both vertices of this edge for the underlying curve,

- `inner_vertices` – for higher-interpolation edges, additional mesh vertices may be added,
- `getPoint(`ξ`)` – returns the coordinates of point on this edge for the given parameter (where $\xi \in [0, 1]$).

Element2d

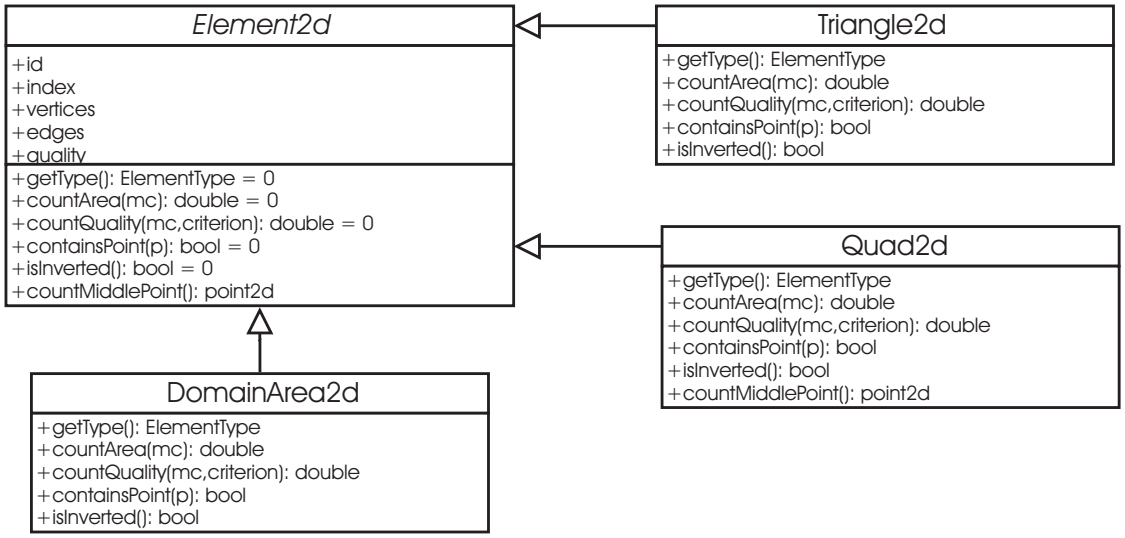


Figure 6.9: Two-dimensional mesh elements representation

The class `Element2d` (Fig. 6.9) defines an abstract mesh element for two-dimensional parametric meshing. Two main mesh element types (`Triangle2d` and `Quad2d`) were defined. Other types could be easily included if required. The `DomainArea2d` class is used to represent the initial domain area of meshed surface patch.

Main attributes and operations:

- `id` – elements from some sub-domains of the discretized model may have ascribed different area identifier,
- `index` – all elements are stored in an indexed way in order to facilitate efficient accessing of elements,
- `vertices` – an ordered (counterclockwise) list of vertices,
- `edges` – an ordered list of edges (following the sequence of vertices),
- `quality` – a quality coefficient, also used for ordering of elements in the heap structure,
- `getType()` – returns type of element (e.g. triangle, quadrilateral),
- `countArea(mc)` – calculates area of the element in metric space, using the provided `MetricContext2d` reference,

- `countQuality(mc, criterion)` – evaluates the quality of the element in metric space according to the selected criterion, using the provided `MetricContext2d` reference,
- `containsPoint(p)` – whether the given point is inside this element,
- `isInverted()` – checks the validity of the element, an element becomes inverted if the orientation of its vertices is no longer counterclockwise (for non-simplex elements the crossing of edges has to be also checked),
- `countMiddlePoint()` – returns middle (representative) point for element calculated as a barycenter of all vertices (for concave quadrilaterals the formula is slightly adjusted in order to assure that the middle point lies within the element).

Vertex3d

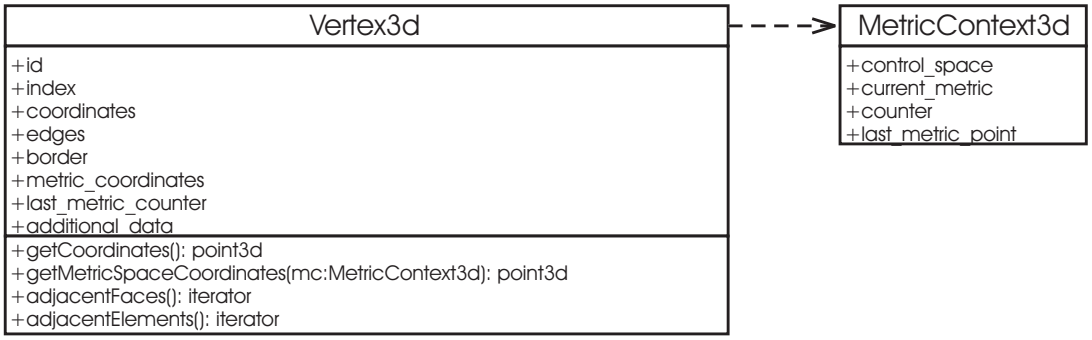


Figure 6.10: Three-dimensional mesh vertex representation

The class `Vertex3d` (Fig. 6.10) defines a mesh vertex for three-dimensional meshing. Main attributes and operations:

- `id` – a number used for identification of vertices (often following the information from model description),
- `index` – all vertices are stored in an indexed container,
- `coordinates` – three-dimensional geometric coordinates of the vertex,
- `edges` – list of references to all mesh edges adjacent to this vertex (in arbitrary order),
- `border` – logical flag for differentiating vertices lying on boundary of the discretized model,
- `metric_coordinates` – each vertex stores its coordinates transformed to metric space in order to increase the efficiency of meshing operations,
- `last_metric_counter` – information (counter of the recently used `MetricContext3d` reference), which allows to decide, whether the cached coordinates of this vertex transformed to metric space are valid, or if they need to be recalculated,

- `additional_data` – some meshing operations require one or more additional data to be stored in vertices (e.g. usage marks, references to projected three-dimensional vertices, weights, etc.),
- `getCoordinates()`, `getMetricSpaceCoordinates(mc)` – this set of operations returns the current coordinates of the vertex or the coordinates transformed to metric space (using the provided `MetricContext3d` reference),
- `adjacentFaces()` – returns an iterator for sequential accessing all faces adjacent to this vertex,
- `adjacentElements()` – returns an iterator for sequential accessing all elements adjacent to this vertex.

Edge3d

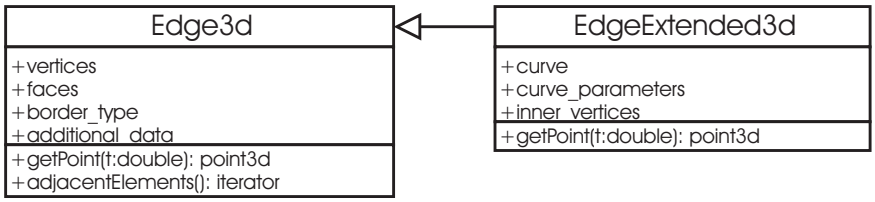


Figure 6.11: Three-dimensional mesh edge representation

The class `Edge3d` (Fig. 6.11) defines a basic mesh edge for three-dimensional meshing. The derived class `EdgeExtended3d` is also provided for representing curvilinear or higher interpolation edges.

Main attributes and operations:

- `vertices` – an ordered pair of vertices (defining the direction of edge),
- `faces` – list of adjacent faces in arbitrary order,
- `border_type` – whether the edge is part of outer/inner boundary or a part of inner mesh,
- `additional_data` – depending on meshing operations, one or more additional data may have to be stored (e.g. usage marks),
- `curve` – (`EdgeExtended3d`) pointer to parametric curve providing the shape information for this edge,
- `curve_parameters` – (`EdgeExtended3d`) parametric representation of both vertices of this edge for the underlying curve,
- `inner_vertices` – for higher-interpolation edges, additional mesh vertices may be added,

- `getPoint(ξ)` – returns the coordinates of point on this edge for the given parameter (where $\xi \in [0, 1]$).

Face3d

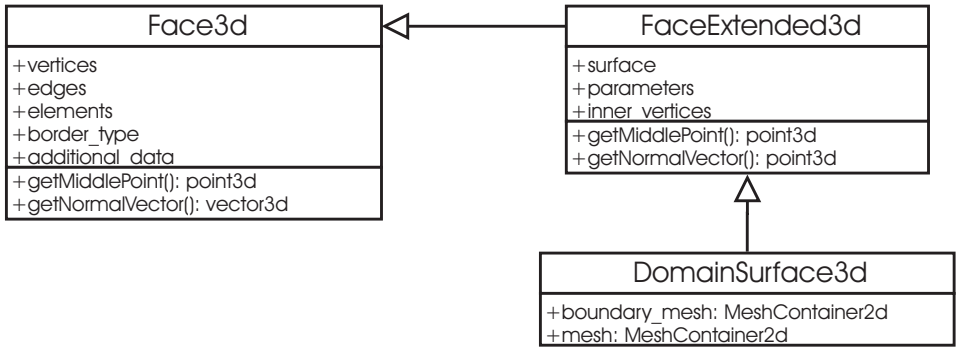


Figure 6.12: Three-dimensional mesh face representation

The class `Face3d` (Fig. 6.12) defines a basic mesh face for three-dimensional meshing. The derived class `FaceExtended3d` is also provided for representing non-planar or higher interpolation faces. The class `DomainSurface3d` is used to represent the domain faces of the discretized model.

Main attributes and operations:

- `vertices` – ordered (counterclockwise) list of vertices,
- `edges` – ordered list of edges (following the sequence of vertices),
- `elements` – adjacent elements (at most two) following the (counterclockwise) orientation of face,
- `border_type` – whether the edge is part of outer/inner boundary or a part of inner mesh,
- `additional_data` – depending on meshing operations, one or more additional data may have to be stored (e.g. usage marks),
- `surface` – (`FaceExtended3d`) pointer to parametric surface providing the shape information for this face,
- `parameters` – (`FaceExtended3d`) parametric representation of all vertices of this face for the underlying surface,
- `inner_vertices` – for higher-interpolation faces, additional mesh vertices may be added,
- `getMiddlePoint()` – returns the coordinates of middle point,
- `getNormalVector()` – returns the normal vector for this face (for extended face the normal is calculated in the middle point of the face).

Element3d

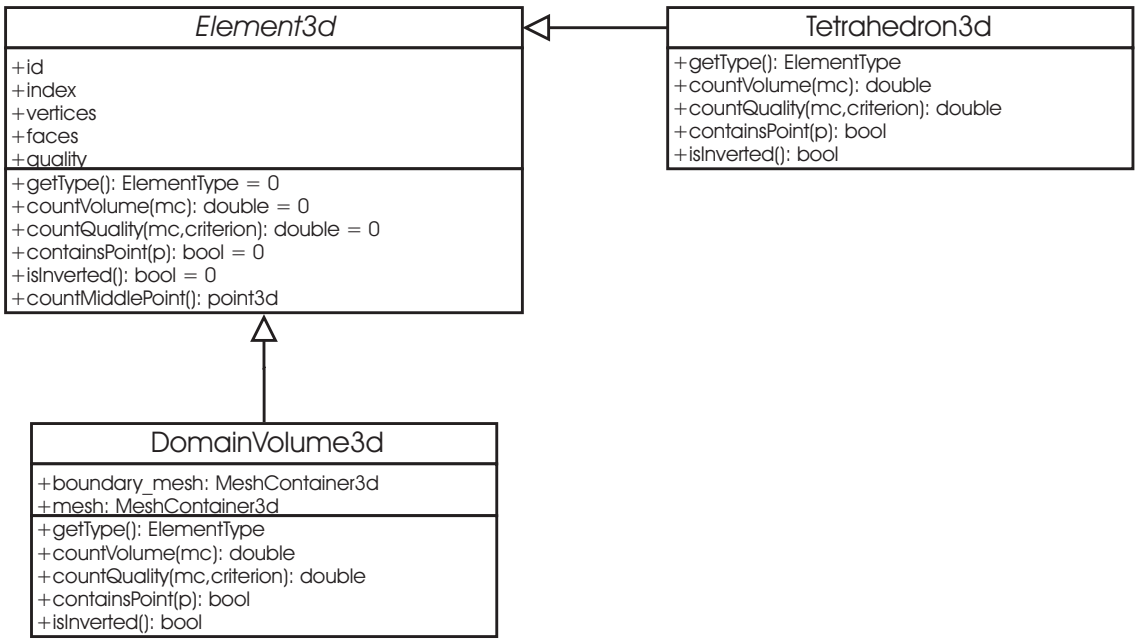


Figure 6.13: Three-dimensional mesh elements representation

The class `Element3d` (Fig. 6.13) defines an abstract mesh element for three-dimensional meshing. The class `Tetrahedron3d` is an implementation of a simple tetrahedron mesh element type. Other types could be easily included if required. The `DomainVolume3d` class is used to represent the domain volume of the meshed model.

Main attributes and operations:

- `id` – elements from some sub-domains of the discretized model may be ascribed with different volume identifiers,
- `index` – all elements are stored with indices in order to facilitate efficient accessing of elements,
- `vertices` – an ordered list of vertices (the orientation is determined by the volume of element, which should be positive),
- `faces` – an ordered list of faces following the sequence of vertices (e.g. for tetrahedron i-th face is opposite to i-th vertex),
- `quality` – a quality coefficient, also used for ordering of elements in the heap structure,
- `getType()` – returns type of element (e.g. tetrahedron),
- `countVolume(mc)` – calculates volume of the volume in metric space, using the provided `MetricContext3d` reference,

- `countQuality(mc, criterion)` – evaluates the quality of the element in metric space according to the selected criterion, using the provided `MetricContext3d` reference,
- `containsPoint(p)` – whether the given point is inside this element,
- `isInverted()` – checks the validity of the element (i.e. whether it has a negative volume).

6.4.2 Mesh Containers

The meshes are stored as a set of interconnected entities. The list of operations includes a set of accessors/mutators for browsing all types of entities (either available directly or gathered using adjacency information from other entities) and some methods extending the properties of mesh entities (e.g. the quality of mesh elements) for the whole mesh.

MeshContainer2d

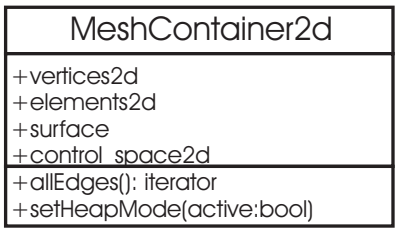


Figure 6.14: Container for two-dimensional mesh

The class `Container2d` (Fig. 6.14) defines a container for two-dimensional mesh (at any phase of its construction).

Main attributes and operations:

- `vertices2d` – an indexed set of mesh vertices stored in a dynamic two-dimensional array of pointers,
- `elements2d` – an indexed set of mesh elements which can be also reorganized into heap order (required during the phase of triangular mesh refinement),
- `surface` – a reference to the underlying parametric surface description for this surface patch discretization,
- `control_space2d` – a reference to a two-dimensional (adaptive) control space structure associated with this mesh,
- `allEdges()` – the list of all mesh edges is not maintained directly and is retrieved from the adjacency lists in mesh vertices whenever required,
- `setHeapMode()` – in heap mode all operations of insertion or removing of mesh element require some additional adjustments to preserve this ordering, enabling of this mode requires an initial reorganization of all elements.

MeshContainer3d

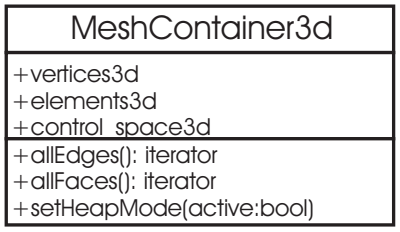


Figure 6.15: Container for three-dimensional mesh

The class `Container3d` (Fig. 6.15) defines a container for three-dimensional mesh (at any phase of its construction).

Main attributes and operations:

- `vertices3d` – an indexed set of mesh vertices stored in a dynamic two-dimensional array of pointers,
- `elements2d` – an indexed set of mesh elements which can be also reorganized into heap order (required during the phase of tetrahedral mesh refinement),
- `control_space3d` – a reference to a three-dimensional (adaptive) control space structure associated with this mesh,
- `allEdges()` – the list of all mesh edges is not maintained directly and is retrieved from the adjacency lists in mesh vertices whenever required,
- `allFaces()` – the list of all mesh faces is not maintained directly and is retrieved from the adjacency lists in mesh elements whenever required,
- `setHeapMode()` – if the heap mode is enabled, all modifications of mesh element trigger additional adjustments to preserve this ordering, enabling of this mode requires also an initial reorganization of all elements.

6.4.3 Memory Management

The process of mesh generation consists of a sequence of mesh transformation (mostly local), associated with such operations as insertion of the new node into the triangulation, or with conversion of two triangles into a single quadrilateral element. Depending upon the phase of the generation process, the number of various mesh entities may gradually increase, decrease or be practically constant.

In order to optimize algorithms which relies on the removing/inserting of mesh entities (e.g. edges or triangles), there is being used an adequately adjusted procedure of allocation and deallocation of mesh elements. Instead of many single memory operations, a set of entities is allocated at once. Additionally, during the routines of mesh generation, where the quantity of elements has an inclination to grow, removed entities are not deallocated, but they are put aside for further use.

6.4.4 Visualization

Due to large sizes of created meshes a proper methods of visualization of a selected parts of the mesh are necessary. In order to visualize the mesh (or some part of it) a snapshot of selected mesh entities is gathered and rendered using OpenGL in a separate thread.

6.5 Prediction of Final Mesh Size

In general case, the final size of the mesh is not known a priori. However, such information can be very useful for better establishing of the sizes of the dynamically created structures used for storing the mesh entities, increasing the efficiency of memory management – and the whole generation procedure. In the presented approach the final size of the mesh is approximated during the process of creation of the mesh, basing upon the property of the incremental triangulation algorithm. During this procedure, all mesh elements are organized in the heap-structure, according to specified quality criterion (the root of the heap contains the element with the lowest quality). The selected quality criterion (4.1) and (5.2) are responsible for assessing how well the given simplex conforms to the prescribed control space (according both to geometrical shape and element size).

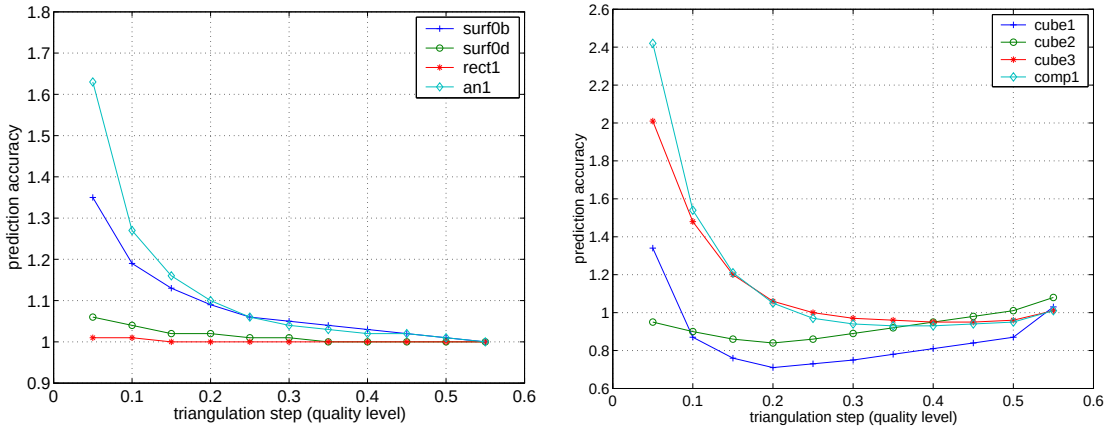
At the subsequent steps of the triangulation algorithm, elements with the worst quality (i.e. too large or badly shaped) are taken from the root of the heap structure. For each taken element a refinement procedure is performed, which consists of insertion of a new node in the circumsphere of the element and local reconfiguration of the mesh according to the selected criterion. The quality criterion is mainly used for determining the order of elements selected for the refinement operation. However, this formula can be additionally used for predicting the final number of elements in the mesh.

The prediction of the final size of the mesh is calculated from the current number of elements in the mesh and the assessed number of elements, which should be created instead of some average element from the current mesh:

$$\mathrm{NT}_p = \mathrm{NT}_c \frac{q_t}{q(K)} \tag{6.1}$$

where NT_p is the predicted number of elements in the final mesh, NT_c is the current number of elements, q_t is the quality threshold for refinement procedure and $q(K)$ is the quality coefficient for the selected element.

The predicted number is updated at regular steps, measured by the quality of the worst triangle in the refinement heap. Figure 6.16 presents the prediction accuracy for some typical mesh generation cases. The two-dimensional examples include the surface mesh SURF0 (NT=$8.7 \cdot 10^4$ and NT=$2.0 \cdot 10^6$, Fig. A.4) with different levels of curvature approximation (and resulting quantity of mesh elements), uniform isotropic mesh RECT1 (NT=$1.2 \cdot 10^6$) and planar strongly anisotropic mesh AN1 (NT=$2.1 \cdot 10^5$, Fig. A.2). For volume meshing there are shown examples of rather uniform mesh CUBE1 (NT=$9.5 \cdot 10^4$, Fig. A.8), irregular isotropic CUBE2 (NT=$3.7 \cdot 10^5$, Fig. A.10), irregular anisotropic CUBE3 (NT=$1.3 \cdot 10^5$, Fig. A.11) and anisotropic mesh COMP1 with complex boundary (NT=$7.9 \cdot 10^5$, Fig. A.21).



(a) two-dimensional (b) three-dimensional

Figure 6.16: Prediction of final element count in successive refinement steps

For two-dimensional examples the accuracy of prediction (measured as the ratio of predicted number of elements at the given meshing step to the actual final quantity) is quite good, depending slightly on the gradation and the size of created meshes. For three-dimensional meshing the results are less accurate, although the precision is still sufficient for proper managing of data structures for storing the mesh entities during the meshing process.

A rough estimate of the final mesh element number can be also obtained at the very beginning of the mesh refinement case. During the initial evaluation of quality coefficients for all elements in the mesh (which is the required step before the refinement loop can begin) the total metric area (or volume) of all elements can be calculated and used as follows:

$$\mathrm{NT}_p = \frac{1}{S_{\mathcal{M}}(K_I)} \sum_{i=1}^{\mathrm{NT}_c} S_{\mathcal{M}}(K_i) \tag{6.2}$$

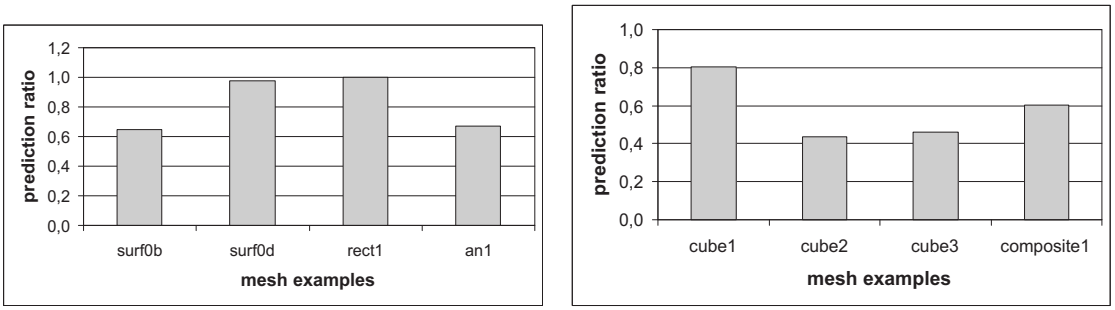
where NT_p is the predicted number of elements in the final mesh, NT_c is the current number of elements, $S_{\mathcal{M}}(K_I)$ is the metric area (volume) of the ideal simplex and $S_{\mathcal{M}}(K_i)$ is the metric area (volume) of simplex K_i.

The results of initial prediction are usually under-estimated (Fig. 6.17), depending on the gradation level of the mesh being created. However, in some cases the initial approximations can be also larger than reality. Usually the accuracy of the initial prediction is not very precise and in the current form may be used only as a rough estimate. Future works should include additional information (e.g. correction for the metric gradation in control space) in order to increase the accuracy.

The described prediction statistics for example meshes are gathered in Table 6.1.

6.6 Computational Complexity

The experimental results were obtained on a single Dual Core AMD Opteron (2x2GHz) machine with 4GB memory.



(a) two-dimensional (b) three-dimensional

Figure 6.17: Initial prediction of final element count

Table 6.1: Prediction accuracy, $q(K_0)$ – quality of the worst element in current mesh

$q(K_0)$	SURF0B	SURF0D	RECT1	AN1	CUBE1	CUBE2	CUBE3	COMP1
0.05	1.35	1.06	1.01	1.63	1.34	0.95	2.01	2.42
0.10	1.19	1.04	1.01	1.27	0.87	0.90	1.48	1.54
0.15	1.13	1.02	1.00	1.16	0.76	0.86	1.20	1.21
0.20	1.09	1.02	1.00	1.10	0.71	0.84	1.06	1.05
0.25	1.06	1.01	1.00	1.06	0.73	0.86	1.00	0.97
0.30	1.05	1.01	1.00	1.04	0.75	0.89	0.97	0.94
0.35	1.04	1.00	1.00	1.03	0.78	0.92	0.96	0.93
0.40	1.03	1.00	1.00	1.02	0.81	0.95	0.95	0.93
0.45	1.02	1.00	1.00	1.02	0.84	0.98	0.95	0.94
0.50	1.01	1.00	1.00	1.01	0.87	1.01	0.96	0.95
0.55	1.00	1.00	1.00	1.00	1.03	1.08	1.01	1.01

6.6.1 Triangular Surface Meshing

The presented time and memory statistics of the meshing process were gathered for a number of meshes with different characteristics: the planar uniform mesh RECT1 (Fig. A.1), planar anisotropic AN1 (Fig. A.2) with analytically specified stretched element sizing and more or less anisotropic surface meshes SURF0 (Fig. A.4), SURF1 (Fig. A.5) and SURF2 (Fig. A.6). For all modeled domains appropriate meshing parameters were adjusted in order to obtain meshes with different final number of triangles. The curvature ratio coefficient was ranging from 0.5 (smallest mesh) to 0.004 (largest mesh) for successive version of SURF0, from 0.1 to 0.006 for SURF1 and from 0.1 to 0.0015 for SURF2. The element sizing in the AN1 mesh is prescribed by analytic formula, where element lengths where proportionally scaled in order to obtain meshes with increasingly larger number of elements. The largest size of generated meshes was limited by the amount of available memory, with the maximum number of about 9.5 millions of triangles for 4GB of memory.

Initialization of control space structures The initial ACS are created basing on the metric sources available directly (i.e. given by user or retrieved from geometry of surfaces and contours). The cost of this operation depends on the number and form of available metric

sources. In the presented examples (Fig. 6.18) the main cost of ACS initialization is associated with retrieving of curvature information from surfaces and contours. Since larger versions of meshes are obtained by decreasing the curvature ratio, the adaptive recognition of surface curvature requires more refined ACS structure, hence the increased cost.

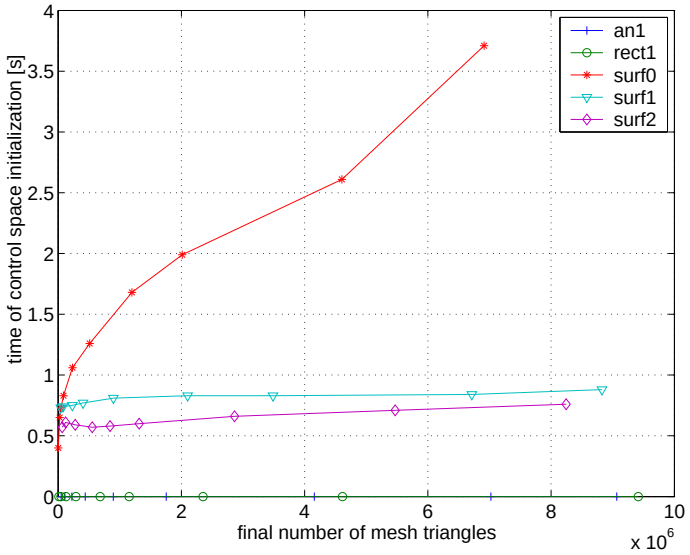


Figure 6.18: Time of creation of initial ACS for surface patches

Triangulation of boundary nodes The boundary triangulation cost depends on the geometric description of the domain (e.g. ratio of boundary nodes to total nodes in the final mesh – Fig. 6.19).

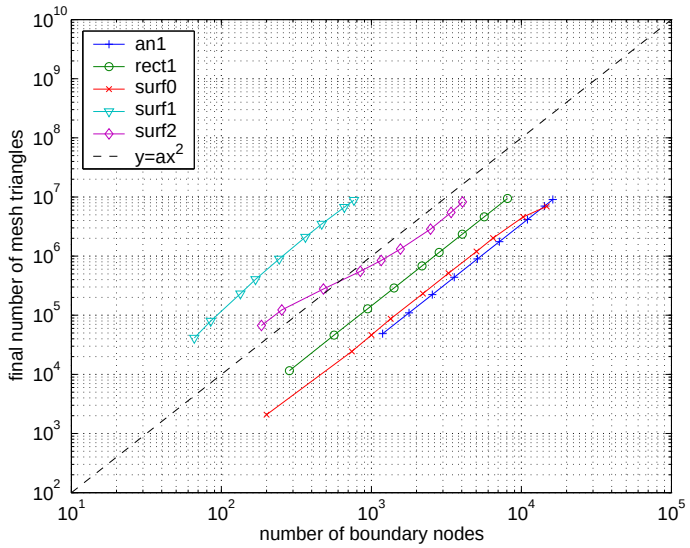


Figure 6.19: Ratio of number of boundary to surface vertices

Figure 6.20 presents the time required for generation and triangulation of boundary nodes.

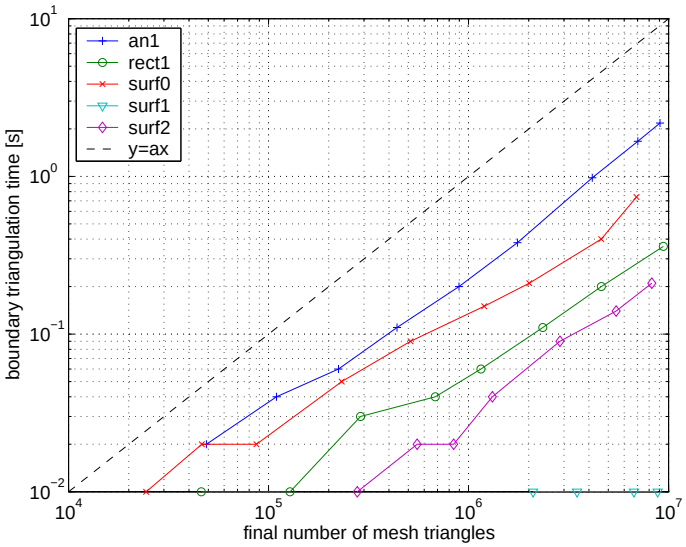


Figure 6.20: Time complexity for surface meshing (boundary nodes))

This phase includes the following main steps:

1. Discretization of boundary edges. The boundary nodes are placed along the contours of the patch using an iterative procedure (with limited number of steps per each node). The total cost of this operation is linear with respect to number of boundary nodes. An additional cost of retrieving metric from control space depends only on the selected control space structure and its size (resulting from the contents and diversity of the stored sizing field). Since the control space structure is adapted to the sizing field (influencing the final number of mesh elements) the size of this structure (number of control nodes) is usually correlated with the size (number of elements) of the created mesh.

2. Projection of three-dimensional boundary nodes onto patch surface. An iterative procedure (with limited number of steps) is used and the actual cost may depend on the specific parametric surface representation.

3. Control space update. The procedure is linear with respect to the number of boundary points. It also depends on the size of the control space structure.

4. Triangulation of boundary nodes. All boundary nodes are inserted into the triangulation one by one, and the final complexity of this operation depends on the average cost of a single point insertion. For each inserted boundary point the following main operations have to be performed (assuming the utilization of Delaunay empty sphere criterion): localization of the containing triangle, identification of triangles forming the cavity and replacing them with new set of elements. The implemented procedure of mesh traversing with auxiliary structures allows to limit the containing triangle

localization to $O(\ln n)$ where n is the number of triangles in the current mesh. The cost of both operations of identifying and replacing of cavity triangles depends on the local characteristics of mesh and may differ at subsequent meshing steps. However, the average cost can be treated as constant and it depends mostly on the variation of mesh element sizing.

The obtained results show linear complexity of boundary triangulation time with respect to the final number of triangles for all tested meshes.

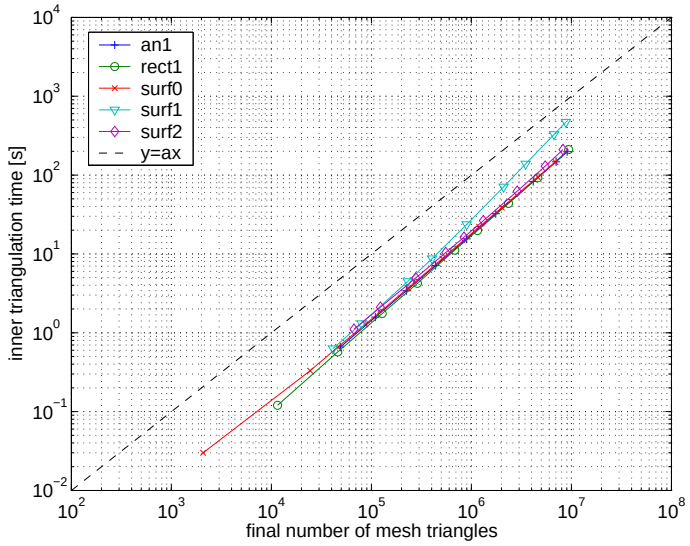


Figure 6.21: Running time statistics for surface meshing (inner nodes)

Triangulation of inner nodes The incremental refinement of the mesh through insertion of inner nodes (Fig. 6.21) includes the three main tasks:

1. Management of the heap ordered list of mesh elements. All mesh elements are organized in the heap structure with the worst element (i.e. element with the lowest quality coefficient) placed on the top. The initial ordering of all mesh elements (obtained from the previous phase of the boundary nodes triangulation) can be achieved in time $O(n \ln n)$ where n is the number of elements in boundary triangulation. Each subsequent alteration (insertion, removing or updating) of any mesh element requires proper update of the heap at the cost of $O(\ln n)$.

2. Determination of coordinates of the new mesh node. Although the refinement point coordinates for the selected triangle is calculated in constant time, an additional number of checks have to be performed in order to validate the insertion according to the (variable) metric space. For typical meshes that additional cost is negligible. However, for highly graded meshes it may increase slightly the overall complexity.

3. Local retriangulation. The retriangulation cost is similar as in boundary triangulation procedure, with some exceptions. The localization of the containing triangle is much

faster, since the refined triangle either contains the new point or is very close. There is also included the additional cost of quality evaluation and heap order update for all altered mesh elements. On the other hand, the octree for containing triangle localization is no longer used and does not need updating for modified elements.

The experimental results show the complexity close to linear. The increased cost of the SURF1 case is caused by a higher gradation of the sizing field resulting in less efficient determination of refinement points coordinates.

Mesh improvement The mesh improvement is performed by iteratively applying some improvement procedures:

1. Laplace smoothing. The cost of this operation depends on the number of mesh points and their average rank (which can be assumed as constant for typical meshes).

2. Topological edge swap. All edges are inspected once and in case of the diagonal swap, the operation is performed in constant time.

3. Geometric edge swap. Unlike in the previous method, this operation checks the mesh edges for geometric quality until this condition is fulfilled in all mesh. The unmodified entities are specially marked in order to reduce unnecessary calculations, and the swap process is guarded against the infinite loops (which may be caused by variation of metric space). The complexity of this operation depends on the initial quality of the mesh, in typical cases it is slightly above linear.

4. The postprocessing operation additionally checks all mesh elements (in constant time).

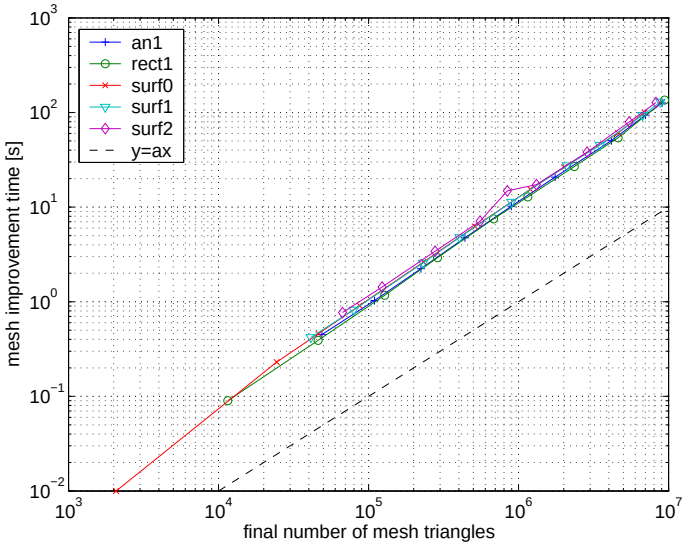


Figure 6.22: Running time statistics for surface meshing (mesh improvement)

The experimental results (Fig. 6.22) show the computational complexity close to linear.

Total meshing time The summary surface meshing time (Fig. 6.23) is mostly influenced by two last operations (inner nodes triangulation and mesh improvement).

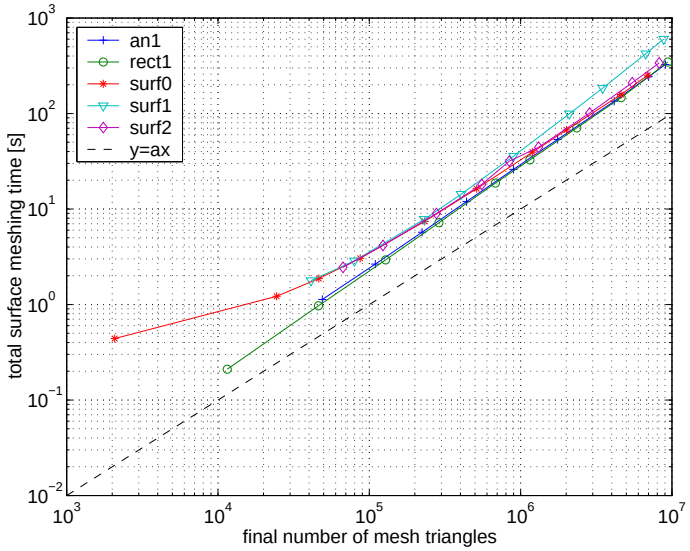


Figure 6.23: Surface meshing total running time statistics

Memory requirements The maximum amount of memory required during the meshing process (Fig. 6.24) is mostly determined by the size of the generated mesh and the created control space structure. The total mesh requirement are usually close to linear, although it may slightly depend on the variation of the sizing field (which may increase the size of the control space structure).

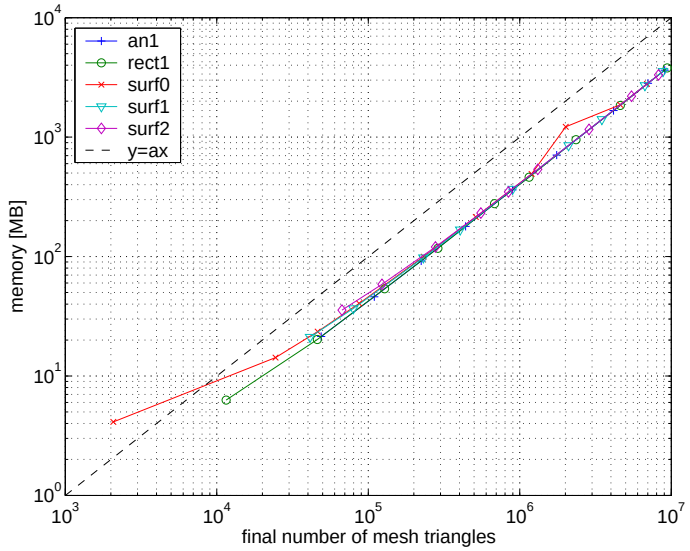


Figure 6.24: Mesh memory requirements statistics

The detailed results for all presented meshes are gathered in Table 6.2.

Table 6.2: Time and memory statistics for surface mesh generation, NP_B – number of boundary nodes, NT – number of triangles in the final mesh, t_c – initialization of ACS structures, t_b – creation and triangulation of boundary nodes, t_i – mesh refinement with inner nodes, t_s – mesh improvement, t_t – total meshing time, and mem – required memory in MB

$NP_B[10^3]$	NT $[10^3]$	$t_c[s]$	$t_b[s]$	$t_i[s]$	$t_s[s]$	$t_t[s]$	mem [MB]
				AN1			
1.2	48.9	0.0	0.02	0.66	0.45	1.13	21.4
2.5	223.6	0.0	0.06	3.41	2.24	5.71	91.9
3.6	439.2	0.0	0.11	7.11	4.75	11.97	178.8
5.1	896.1	0.0	0.20	15.55	10.24	25.99	362.0
7.1	1 754.8	0.0	0.38	32.36	20.74	53.48	707.1
11.0	4 157.3	0.0	0.98	82.86	50.43	134.27	1 671.0
14.3	7 023.3	0.0	1.67	147.41	93.80	242.88	2 819.9
16.2	9 065.6	0.0	2.18	197.72	126.23	326.13	3 637.6
				RECT1			
0.3	11.5	0.0	0.00	0.12	0.09	0.21	6.3
0.6	46.1	0.0	0.01	0.57	0.39	0.97	20.2
1.4	288.6	0.0	0.03	4.23	2.92	7.18	117.3
2.2	682.2	0.0	0.04	11.16	7.51	18.71	277.5
2.8	1 152.9	0.0	0.06	19.79	12.84	32.69	461.7
4.0	2 351.1	0.0	0.11	43.66	26.83	70.60	951.5
5.7	4 614.8	0.0	0.20	92.24	54.07	146.51	1 841.2
8.1	9 415.1	0.0	0.36	210.36	135.35	346.07	3 796.1
				SURF0			
0.2	2.1	0.40	0.00	0.03	0.01	0.44	4.1
0.7	24.4	0.65	0.01	0.33	0.23	1.22	14.3
1.3	87.0	0.73	0.02	1.28	0.91	3.04	40.5
2.2	232.3	1.06	0.05	3.68	2.63	7.42	99.8
3.3	513.0	1.26	0.09	8.71	6.25	16.31	214.5
5.0	1 197.4	1.68	0.15	22.05	15.43	39.31	492.2
6.5	2 013.3	1.99	0.21	38.74	26.39	67.33	1 218.8
14.7	6 912.9	3.71	0.74	146.57	100.45	251.47	2 793.1
				SURF1			
0.1	41.2	0.73	0.00	0.63	0.42	1.78	21.1
0.1	230.9	0.75	0.00	4.47	2.57	7.79	97.9
0.2	404.2	0.77	0.00	8.73	4.78	14.28	167.6
0.2	897.9	0.81	0.00	23.63	11.30	35.74	367.0
0.4	2 102.0	0.83	0.01	70.86	27.52	99.22	851.5
0.5	3 486.3	0.83	0.01	138.22	45.26	184.32	1 409.2
0.7	6 712.4	0.84	0.01	327.96	93.70	422.51	2 706.1
0.8	8 823.8	0.88	0.01	470.71	129.67	601.27	3 554.2
				SURF2			
0.2	67.1	0.57	0.00	1.11	0.77	2.45	35.6
0.5	277.7	0.59	0.01	4.96	3.41	8.97	120.4
0.8	553.8	0.57	0.02	10.41	7.11	18.11	231.4
1.2	842.4	0.58	0.02	16.25	14.87	31.72	347.6
1.6	1 316.2	0.60	0.04	26.41	17.31	44.36	538.1
2.5	2 863.8	0.66	0.09	62.30	38.12	101.17	1 159.5
3.4	5 470.0	0.71	0.14	129.83	79.49	210.17	2 203.3
4.1	8 243.2	0.76	0.21	210.62	127.27	338.86	3 314.6

6.6.2 Volume Meshing

The following set of typical geometries was used for experimental comparison of meshing process complexity: CUBE1 (isotropic and rather regular discretization of a cube – Fig. A.8), CUBE2 (isotropic and graded discretization of a cube – Fig. A.10), CUBE3 (similar case as the previous one, but anisotropic – Fig. A.11), COMP1 (graded model with composite boundary – Fig. A.21), CYLINDER1 (isotropic and graded discretization of cylinder – Fig. A.15) and CYLINDER2 (similar as the previous one, but anisotropic – Fig. A.16). For all tested geometries some meshing parameters (e.g. curvature ratio) or sizing prescription were successively adjusted in order to obtain for each geometry a sequence of meshes with similar characteristic but with incrementally larger number of elements.

Figure 6.25 shows the number of tetrahedra in the final mesh compared to the number of triangles in surface mesh. This comparison gives some estimation regarding the surface to volume ratio of each model, which has an impact on the complexity of further meshing procedures.

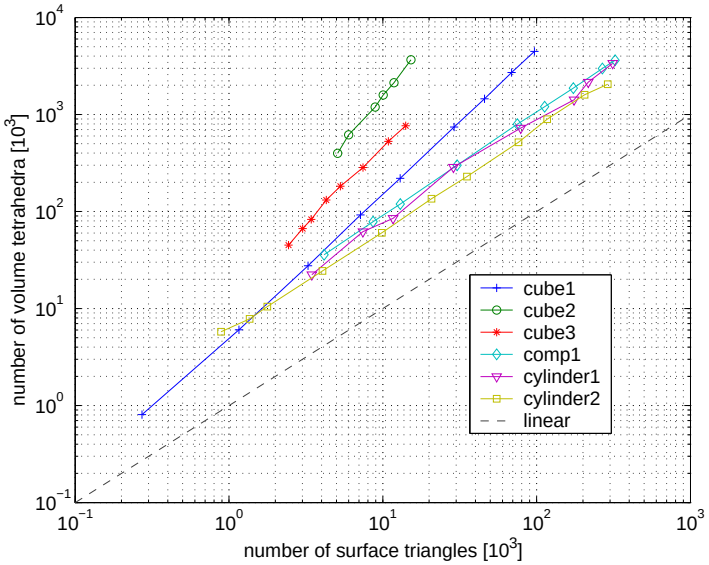


Figure 6.25: Ratio of number of boundary to volume vertices

Creation of boundary meshes The preparation of boundary meshes consists of the following steps:

1. Discretization of all surface patches in the modeled domain using the surface meshing procedures (Sec. 6.6.1).

2. Creation of surface boundary. The mesh elements from all surface patches defining the given domain volume have to be combined into proper boundary description for volume meshing. This includes projection of mesh entities from parametric space of patches to three-dimensional space and association of vertices and edges adjacent to

more than one surface patch. Both operations can be performed in linear time with respect to number of boundary vertices, edges and elements.

3. Creation of control space. The 3D control space is initialized and updated with information gathered from control spaces of each of the surface patches. The procedure is linear with respect to number of boundary vertices, although it may be influenced by the variation of volume control space.

The summary computation cost of this phase is presented on Fig. 6.26.

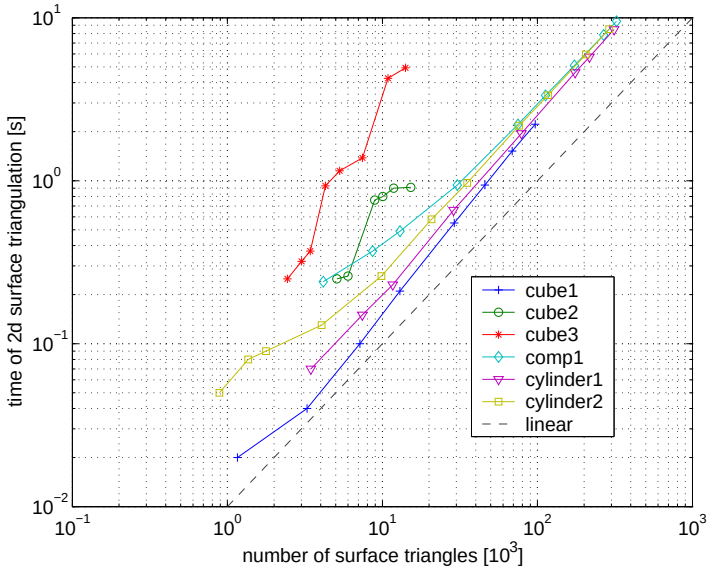


Figure 6.26: Statistics for volume mesh generation (triangulation of surfaces)

Triangulation of boundary nodes During this phase of volume mesh generation all boundary vertices are incrementally inserted into the tetrahedral mesh (in randomized order), starting with an initial mesh (a cubicoid split into six tetrahedra). The total computational complexity of this phase depends on the average cost of a single mesh point insertion, which consists of the following steps:

1. Localization of the containing tetrahedron. A starting tetrahedron is selected from an octree structure ($O(\ln n)$ where n is the current number of elements in the mesh) and then the directed mesh traversing is used. The average number of traversed tetrahedra per each inserted point depends on the octree leaf threshold (i.e. maximum number of tetrahedra in the mesh represented by single leaf) and does not depend on the number of elements in the mesh.

2. Identification of tetrahedra forming the empty cavity and replacing the cavity by new tetrahedra. Starting from the containing tetrahedron, subsequent adjacent tetrahedra containing the new node in circumsphere are found and included into an initial cavity. This initial cavity is then pruned by removing some elements in order to guarantee

validity of new tetrahedra. The number of checked tetrahedra and the initial cavity size increases with the size of the mesh. However, the cavity size after pruning (also the number of tetrahedra removed and inserted per each inserted point) remains roughly constant for all inserted boundary vertices. For each removed or created tetrahedron an update of the octree structure for localization of containing tetrahedra has to be performed, which requires additional computation cost $O(\ln n)$ (where n is the current number of elements in the mesh).

3. Auxiliary points. For each boundary vertex some additional points may be added into triangulation in its vicinity in order to increase the quality of mesh and thus the efficiency of the meshing process. This operation relies on local information, the additional introduced cost depends on the local quality of the mesh and number of tetrahedra adjacent to the inserted boundary node.

The measured times of boundary triangulation are shown on Fig. 6.27. In most cases the complexity of this procedure is close to linear with exception of highly anisotropic cases, where the time cost is higher.

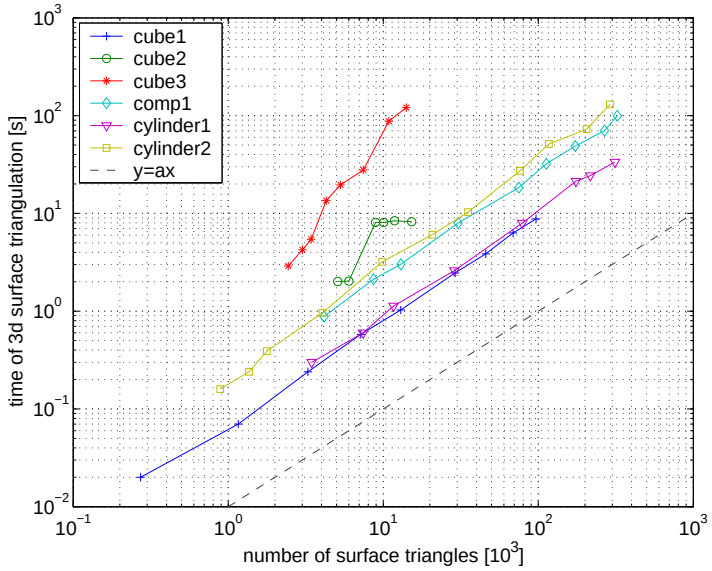


Figure 6.27: Statistics for volume mesh generation (triangulation of boundary nodes)

Constraining of boundary triangulation The constraining consists of the following steps:

1. Initial inspection of boundary. All boundary edges and faces are inspected and any missing entity is included in a special list. The existing entities are marked as unmodifiable. The cost of this operation is linear with respect to the number of boundary vertices.

2. Recovery of missing edges and faces. An iterative procedure of mesh transformation in the vicinity of missing entities is used with a number of heuristics (including swapping

of edges and faces, insertion of additional nodes and special metric transformation). The cost of these operations (as well as the number of initially missing entities) is highly dependent on the complexity of geometric boundary description and local mesh quality.

3. Removing of obsolete tetrahedra. After all boundary faces are recovered, the tetrahedra adjacent to them on proper side are marked as valid. This information is propagated to other elements through adjacency connections, avoiding crossing of boundary faces and already marked tetrahedra. Then, all unmarked tetrahedra can be removed. This operation is linear with respect to number of faces and elements in the mesh.

The procedure of mesh constraining is the least consistent with regard to measured time complexity (Fig. 6.28). The reason for this situation is the fact, that although the cost of boundary recovery depends on the mesh size, it is very sensitive to the actual complexity and quality of the boundary mesh.

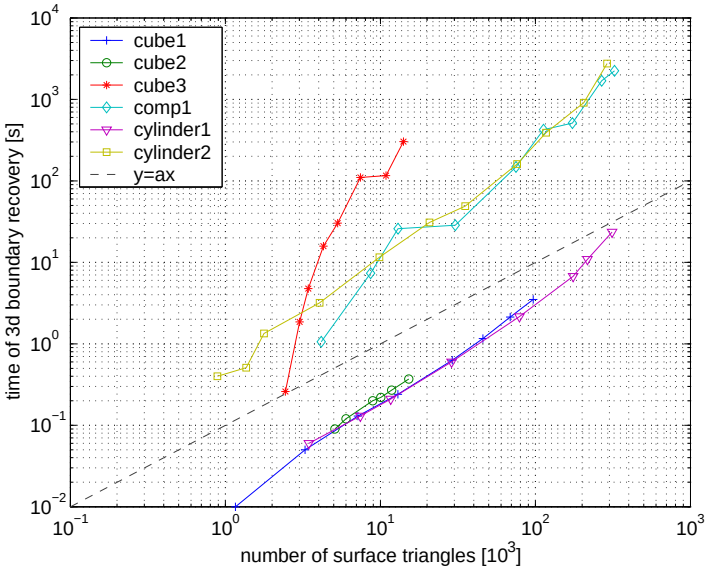


Figure 6.28: Statistics for volume mesh generation (boundary recovery)

Triangulation of inner nodes The incremental mesh refinement consists of the three main tasks:

1. Management of the heap ordered list of mesh elements. The heap structure is used for quick selection of the worst element in a manner similar to the two-dimensional meshing case. The initial organization of all mesh elements requires an average $O(n \ln n)$ operations, where n is the number of elements in boundary triangulation. Each alteration (insertion, removing or updating) of any mesh element requires update of the heap at the cost of $O(\ln n)$ operations.

2. Determination of coordinates of the new mesh node. The new point is inserted in the circumcenter of the selected tetrahedron. An additional check has to be performed in order to validate the insertion according to some quality conditions. Although for typical mesh that additional cost is negligible, it may increase slightly the overall complexity for highly graded or anisotropic meshes.

3. Local retriangulation. The retriangulation cost is similar as in boundary triangulation procedure. The containing tetrahedron can be localized faster, since the improved tetrahedron either already contains the new point or is very near. The octree structure for containing element localization is no longer used (and there are no updates for modified elements). However, there is the additional cost of quality evaluation and heap order update for all altered (removed, inserted or modified) elements.

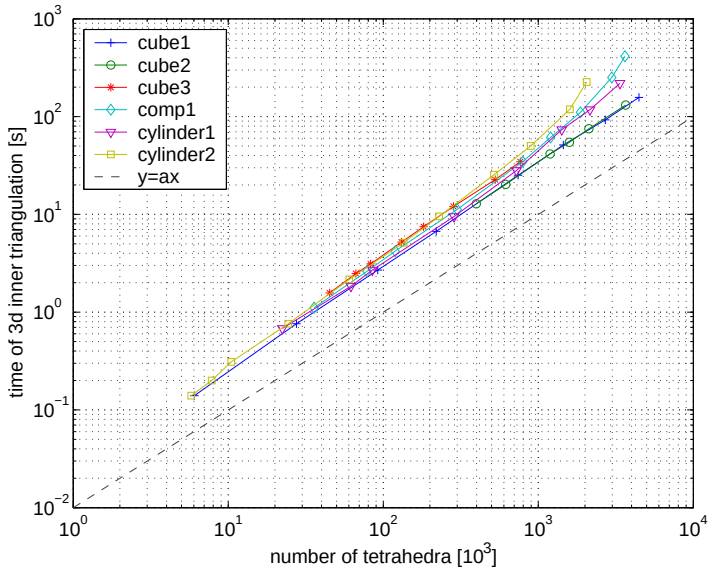


Figure 6.29: Statistics for volume mesh generation (mesh refinement)

The experimental results (Fig. 6.29) show the complexity close to linear with slightly increased cost for meshes with higher gradation and anisotropy.

Mesh improvement The mesh improvement is performed by iteratively applying some improvement procedures:

1. Laplace smoothing. The cost of this operation depends on the number of mesh points and their average rank.

2. Edge and face swap. A number of transformation templates are used in order to increase the quality of the mesh. The cost of each transformation is constant, the number of transformations depends on the number of mesh elements and the current mesh quality.

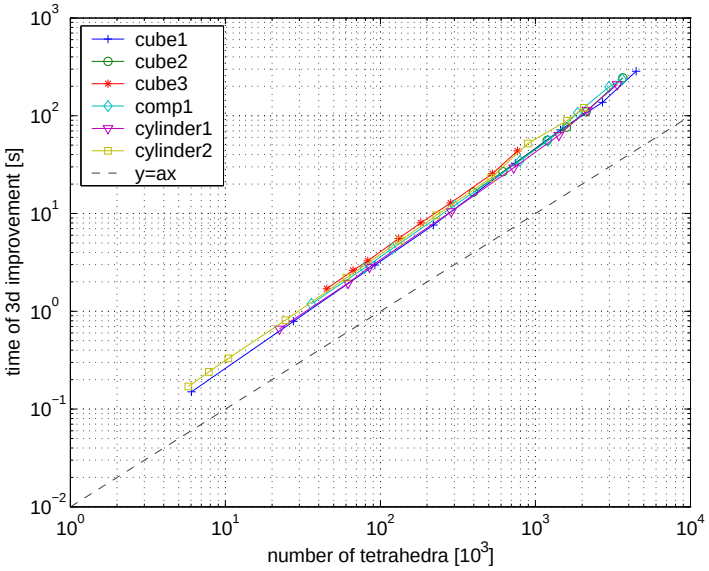


Figure 6.30: Statistics for volume mesh generation (mesh improvement)

The operation of mesh improvement (Fig. 6.30) shows very similar and close to linear computational complexity for all tested geometries.

Total meshing time The total 3D triangulation time (Fig. 6.31) is almost linear for regular meshes and is slightly higher for cases with high anisotropy or complex boundary description.

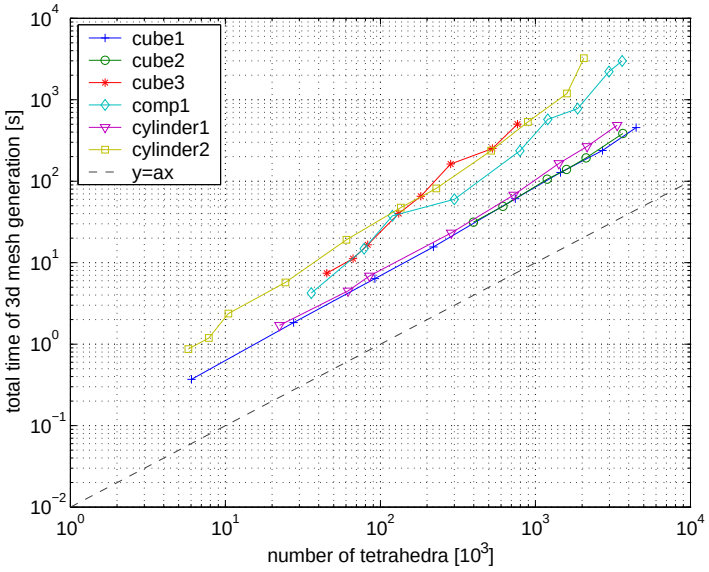


Figure 6.31: Statistics for volume mesh generation (total meshing time)

Memory requirements Memory requirements (Fig. 6.32) are also consistent for all tested examples. The higher memory cost for CUBE3 model is caused by an increased precision (and size) of the control space structure required for accurate representation of high gradation of elements.

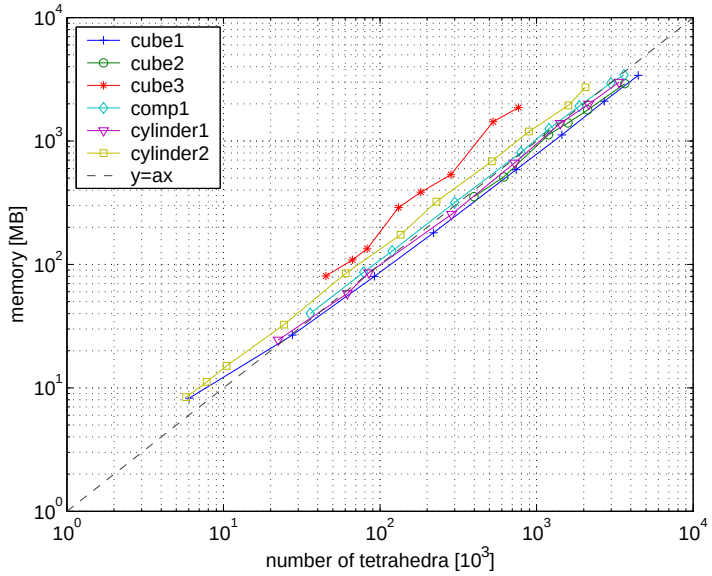


Figure 6.32: Statistics for memory requirements during volume mesh generation

The detailed statistics are gathered in Table 6.3.

6.7 Chapter Summary

In order to assure the extendability of the created mesh generation system an object oriented approach was used. The specifications of the three main class hierarchies: for domain geometry representation, control space structure and mesh representation were described. Some optimization methods oriented on increasing the efficiency of mesh management were also proposed. The computational and memory complexity of the developed meshing algorithms were tested and analyzed for a number of example cases.

Table 6.3: Time and memory statistics for volume mesh generation, NT_S – number of boundary faces, NT – number of tetrahedra in the final mesh, t^{2d} – generation of boundary surface meshes, t_b^{3d} – triangulation of boundary nodes, t_c^{3d} – constraining of boundary mesh, t_i^{3d} – mesh refinement with inner nodes, t_s^{3d} – mesh improvement, t_t^{3d} – total meshing time, and mem – required memory in MB

$NT_S[10^3]$	NT $[10^3]$	$t^{2d}[s]$	$t_b^{3d}[s]$	$t_c^{3d}[s]$	$t_i^{3d}[s]$	$t_s^{3d}[s]$	$t_t^{3d}[s]$	mem [MB]
				CUBE1				
3.3	27.6	0.04	0.24	0.05	0.76	0.79	1.84	26.8
13.0	219.8	0.21	1.03	0.24	6.70	7.63	15.60	180.7
29.1	742.2	0.55	2.46	0.63	25.13	32.62	60.85	587.1
45.8	1 450.8	0.94	3.86	1.16	51.31	71.68	128.02	1 121.6
68.8	2 706.6	1.52	6.32	2.13	92.65	137.01	238.13	2 102.1
96.8	4 467.2	2.22	8.79	3.50	157.16	285.64	455.11	3 403.2
				CUBE2				
5.1	398.0	0.25	2.01	0.09	12.81	16.50	31.42	354.1
6.0	616.9	0.26	2.03	0.12	20.27	26.75	49.17	511.7
8.9	1 191.7	0.76	8.10	0.20	41.51	56.03	105.84	1 123.8
10.1	1 587.1	0.80	8.07	0.22	54.68	76.36	139.34	1 403.3
11.8	2 123.6	0.90	8.40	0.27	75.11	109.67	193.45	1 787.4
15.2	3 662.9	0.91	8.23	0.37	131.33	244.53	384.46	2 917.1
				CUBE3				
2.4	45.1	0.25	2.90	0.26	1.57	1.70	7.43	80.7
3.4	82.9	0.37	5.44	4.76	3.11	3.28	16.59	133.9
5.3	182.1	1.15	19.56	30.30	7.48	8.03	65.37	385.6
7.4	284.1	1.38	27.80	110.04	12.03	12.84	162.71	534.5
10.9	528.0	4.25	87.62	116.13	22.68	25.56	251.99	1 433.3
14.1	766.2	4.93	121.08	303.12	34.58	43.66	502.45	1 864.5
				COMP1				
8.6	78.6	0.37	2.12	7.35	2.61	2.76	14.85	86.5
13.0	119.5	0.49	3.02	25.92	4.21	4.39	37.56	128.3
74.9	794.8	2.21	18.36	148.12	34.64	34.56	235.70	809.0
173.3	1 870.4	5.10	48.84	511.63	110.57	107.01	778.11	1 918.5
267.4	2 983.1	7.85	70.33	1 692.30	251.18	196.71	2 210.63	2 951.7
323.7	3 623.9	9.51	99.67	2 243.78	413.91	235.48	2 993.03	3 431.8
				CYLINDER1				
7.4	61.9	0.15	0.60	0.13	1.84	1.94	4.51	58.2
28.8	286.1	0.66	2.60	0.60	9.51	10.49	23.20	254.5
79.0	724.1	1.95	7.96	2.16	28.08	29.34	67.56	663.6
175.1	1 418.3	4.61	21.34	6.74	73.51	63.29	164.91	1 391.6
216.6	2 153.9	5.75	24.29	10.88	117.76	113.69	266.66	1 992.2
312.4	3 368.4	8.47	33.46	23.57	219.06	208.67	484.84	2 999.8
				CYLINDER2				
4.1	24.4	0.13	0.96	3.19	0.76	0.81	5.72	32.6
20.7	135.3	0.58	6.06	30.93	5.14	5.30	47.43	173.9
35.2	229.3	0.97	10.30	48.99	9.52	9.58	81.40	323.2
117.1	894.5	3.35	51.31	390.26	50.11	52.02	533.73	1 194.9
205.0	1 599.3	5.95	72.74	910.85	118.75	89.02	1 191.42	1 943.2
289.9	2 052.5	8.51	130.35	2 752.84	225.49	120.37	3 229.16	2 727.0

Chapter 7

Conclusions

7.1 Research Contributions

Anisotropic Metric Transformation The definition of anisotropic metric for mesh generation was presented for meshing of both the parametric surfaces and three-dimensional volumes. The selected metric form and a set of associated operations were used to create a transformed coordinate system, which allows for a fairly easy and elegant implementation of a wide range of meshing routines originally created for isotropic meshes.

Control Space Implementation An object-oriented specification of an abstract adaptive control space interface was proposed. The ACS interface contains a set of operations necessary for storing and management of a discrete metric data. Implementations of two different structures: quadtree/octree grid and background mesh were presented as examples. A number of optimization for control space utilization were proposed for both the general ACS interface and the specific implementations.

An iterative procedure of automated construction of control space for both surface and volume meshing has been developed and described. The additional information which becomes available at the successive steps of mesh generations is used to adaptively refine the sizing field as required.

A number of parameters have been supplied for controlling the process of ACS initialization and utilization during meshing. The selection of most important parameters has been listed and theirs influence on the created meshes has been shown in a number of examples for both two- and three-dimensional meshing problems.

Mesh Generator Architecture The specifications of the three main object-oriented hierarchies: for domain geometry representation, control space structure and mesh representation were described. Some optimization methods oriented on increasing the efficiency of mesh management were also proposed.

Anisotropic Mesh Generation on Parametric Surfaces The selected steps of the presented algorithm (e.g. criteria of retriangulation, placement of inner nodes, metric introduction, etc.) were inspected in detail and a number of improvements were proposed. The overall computational complexity was analyzed and the results of practical experiments were provided.

For generation of quadrilateral meshes an indirect frontal approach was inspected which iteratively converts mesh triangles into quadrilaterals. Two methods were implemented with necessary adjustments for anisotropic meshing with metric transformation tensor approach. As a result of this research a mixed method is proposed where both inspected conversion method may be used alternately, depending on the local metric characteristics. Finally, a number of examples are provided illustrating the performance of the implemented procedures.

Anisotropic Volume Mesh Generation The generation of three-dimensional tetrahedral meshes with anisotropic metric transformation was described. A multi-step procedure coping with boundary recovery problem was proposed in order to avoid the necessity of inserting additional nodes at the domain boundary. A concept of systematic introduction of auxiliary nodes during the boundary mesh nodes triangulation was proposed, which allowed to significantly increase the efficiency of the triangulation process. Finally, the computational complexity of the presented implementation was analyzed and a number of example results were shown.

7.2 Future Works

Further developments regarding the unstructured mesh generation problems will concentrate on two areas:

- Parallel mesh generation. Two categories of parallel techniques are considered: coarse-grained decomposition of a modeled domain[179, 185–187] and a fine-grained concurrent openMP implementation of selected algorithms.
- Meshing of non-parametric surfaces. Representations like sets of facets, point clouds or implicit surfaces are not directly available for meshing with the presented mesh generation. An introduction of a local parameterization layer is considered in order to extend the applicability of the created software.

Bibliography

[1] S.J. Owen. Meshing research corner. on the Web. http://www.andrew.cmu.edu/user/sowen/mesh.html.

[2] R. Schneiders. Mesh generation & grid generation. on the Web. http://www-users.informatik.rwth-aachen.de/~roberts/meshgeneration.html.

[3] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, Singapore, 1992.

[4] M. Bern and P. Plassmann. Mesh generation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 6. Elsevier Science, 1999.

[5] S.J. Owen. A survey of unstructured mesh generation technology. In *Proc.* 7^{th} *Int. Meshing Roundtable*, pages 239–267, Dearborn, MI, USA, Oct. 1998. Sandia National Laboratories.

[6] J. F. Thomson, B. K. Soni, and N. P. Weatherill, editors. *Handbook of Grid Generation*. CRC Press LLC, 1999.

[7] M.A. Yerry and M.S. Shephard. Three-dimensional mesh generation by modified octree technique. *International Journal for Numerical Methods in Engineering*, 20:1965–1990, 1984.

[8] P. Baehmann, L. Scott, L. Wittchen, M. Shephard, K. Grice, and M. Yerry. Robust geometrically based, automatic two-dimensional mesh generation. *Internat. J. Numer. Methods Engrg.*, 24:1043–1078, 1987.

[9] M. Shephard and M. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *Internat. J. Numer. Methods Engrg.*, 32:709–749, 1991.

[10] P. Frey and L. Marechal. Fast adaptive quadtree mesh generation. In *Proc.* 7^{th} *Int. Meshing Roundtable*, pages 211–222, Dearborn, MI, USA, Oct. 1998. Sandia National Laboratories.

[11] K. Tchon, K. Mohammed, G. Francois, and C. Ricardo. Constructing anisotropic geometric metrics using octrees and skeletons. In *Proc.* 12^{th} *Int. Meshing Roundtable*, pages 293–304, Santa Fe, New Mexico, USA, Sept. 2003. Sandia National Laboratories.

[12] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proc.* 12^{th} *Int. Meshing Roundtable*, pages 103–114, Santa Fe, New Mexico, USA, Sept. 2003. Sandia National Laboratories.

[13] J. Teran, N. Molino, R. Fedkiw, and R. Bridson. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, 21:2–18, 2005.

[14] W. Schroeder and M. Shephard. A combined octree/Delaunay method for fully automatic 3D mesh generation. *Internat. J. Numer. Methods Engrg.*, 29:37–55, 1990.

[15] N. Hitschfeld-Kahler. Generation of 3D mixed element meshes using a flexible refinement approach. *Engineering with Computers*, 21:101–114, 2005.

[16] Y. Zhang, Ch. Bajaj, and B.-S. Sohn. 3D finite element meshing from imaging data. *Comput. Methods Appl. Mech. Engrg.*, 194:5083–5106, 2005.

[17] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, USA, 2001.

[18] A. Bowyer. Computing Dirichlet tessellations. *Comput. J.*, 24(2):162–166, 1981.

[19] D.F. Watson. Computing the n-dimensional Delaunay tesselation with applications to Voronoï polytopes. *Comput. J.*, 24(2):167–172, 1981.

[20] C.L. Lawson. Properties of n-dimensional triangulations. *Computer Aided Geometric Design*, 3:231–246, 1986.

[21] P. Su and R.L.S. Drysdale. A comparison of sequential Delaunay triangulation algorithms. In *Proc.* 12^{th} *Symposium on Computational Geometry*, pages 61–70. ACM, 1996.

[22] B. Žalik. An efficient sweep-line Delaunay triangulation algorithm. *Computer-Aided Design*, 37:1027–1038, 2005.

[23] T.J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, 5:161–175, 1989.

[24] P. George, F. Hecht, and E. Saltel. Automatic mesh generation with specified boundary. *Comput. Methods Appl. Mech. Engrg.*, 92:169–188, 1991.

[25] N. Weatherill and O. Hassan. Efficient three dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *Internat. J. Numer. Methods Engrg.*, 37:2005–2039, 1994.

[26] H. Borouchaki and S.H. Lo. Fast Delaunay triangulation in three dimensions. *Comput. Methods Appl. Mech. Engrg.*, 128(1-2):153–167, 1995.

[27] P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing. Applications to Finite Elements*. Hermès, Paris, 1998.

[28] P.J. Frey and P.-L. George. *Mesh generation. Application to finite elements*. Hermès Science Publishing, Paris, Oxford, 2000.

[29] A. Rassineux. Generation and optimization of tetrahedral meshes by advancing front technique. *Int. J. Numer. Meth. Engrg*, 41(4):651–674, 1998.

[30] C.K. Lee and R.E. Hobbs. Automatic adaptive finite element mesh generation over arbitrary two-dimensional domain using advancing front technique. *Computers and Structures*, 71:9–34, 1999.

[31] R. Löhner and J. Cebral. Generation of non-isotropic unstructured grids via directional enrichment. *Int. J. Numer. Meth. Engng*, 49:219–232, 2000.

[32] D.L. Marcum. Efficient generation of high-quality unstructured surface and volume grids. *Engineering with Computers*, 17:211–233, 2001.

[33] R. Löhner. Progress in grid generation via the advancing front technique. *Engrg. Comput.*, 12:186–199, 1996.

[34] D.I. Holmes. Generalized method of decomposing solid geometry into hexahedron finite elements. In *Proc.* 4^{th} *Int. Meshing Roundtable*, pages 141–152, Albuquerque, NM, USA, Oct. 1995. Sandia National Labs.

[35] T. Blacker. The cooper tool. In *Proc.* 5^{th} *Int. Meshing Roundtable*, pages 13–30, Pittsburgh, PA, USA, Oct. 1996. Sandia National Labs.

[36] C.G. Armstrong, D.J. Robinson, R.M. McKeag, T.S. Li, S.J. Bridgett, R.J. Donaghy, and C.A. McGleenan. Medials for meshing and more. In *Proc.* 4^{th} *Int. Meshing Roundtable*, pages 277–288, Albuquerque, NM, USA, Oct. 1995. Sandia National Labs.

[37] T.D. Blacker. Paving: A new approach to automated quadrilateral mesh generation. *Int. J. Num. Meth. Eng.*, 32:811–847, 1991.

[38] D.R. White and P. Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. In *Proc.* 6^{th} *Int. Meshing Roundtable*, pages 323–335, Park City, UT, USA, Oct. 1997. Sandia National Laboratories.

[39] T.D. Blacker and R.J. Meyers. Seams and wedges in plastering: A 3D hexahedral mesh generation algorithm. *Engineering with Computers*, 9:83–93, 1993.

[40] C.K. Lee and S.H. Lo. A new scheme for the generation of a graded quadrilateral mesh. *Computers & Structures*, 52(5):847–857, 1994.

[41] Y.K. Lee and C.K. Lee. Automatic generation of anisotropic quadrilateral meshes on three-dimensional surfaces using metric specifications. *Int. J. Numer. Meth. Engng*, 53:2673–2700, 2002.

[42] S.J. Owen, M.L. Staten, S.A. Canann, and S. Saigal. Q-morph, an indirect approach to advancing front quad meshing. *Int. J. Numer. Meth. Engrg*, 44(9):1317–1340, 1999.

[43] K. Shimada, J.-H. Liao, and T. Itoh. Quadrilateral meshing with directionality control through the packing of square cells. In *Proc.* 7^{th} *Int. Meshing Roundtable*, pages 61–76, Dearborn, MI, USA, Oct. 1998. Sandia National Laboratories.

[44] M. Bern and D. Eppstein. Quadrilateral meshing by circle packing. *Int. J. Comput. Geom. Appl.*, 10(4):347–360, 2000.

[45] N. Viswanath, K. Shimada, and T. Itoh. Quadrilateral meshing with anisotropy and directionality control via close packing of rectangular cells. In *Proc.* 9^{th} *Int. Meshing Roundtable*, pages 217–225, New Orleans, CA, Oct. 2000. Sandia National Laboratories.

[46] S.J. Owen. *Non-Simplical Unstructured Mesh Generation.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA. USA, April 1999.

[47] L.A. Freitag and C. Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *Int. J. Numer. Meth. Engng*, 40:3979–4002, 1997.

[48] M.L. Staten and S.A. Canann. Post refinement element shape improvement for quadrilateral meshes. In *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pages 9–16. ASME, July 1997.

[49] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms*, 30:302–322, 1999.

[50] P.M. Knupp. Winslow smoothing on two-dimensional unstructured meshes. *Engineering with Computers*, 15:263–268, 1999.

[51] B. Balendran. A direct smoothing method for surface meshes. In *Proc.* 8^{th} *Int. Meshing Roundtable*, pages 189–193, South Lake Tahoe, CA, Oct. 1999. Sandia National Laboratories.

[52] H. Edelsbrunner, X.-Y. Li, G. Miller, A. Stathopoulos, D. Talmor, S.-H.Teng, A. Üngör, and N. Walkington. Smoothing and cleaning up slivers. In *Proc.* 32^{nd} *Annual Symposium on the Theory of Computing*, pages 273–278, Portland, Oregon, May 2000. Association for Computing Machinery.

[53] T. Zhou and K. Shimada. An angle-based approach to two-dimensional mesh smoothing. In *Proc.* 9^{th} *Int. Meshing Roundtable*, pages 373–384, New Orleans, CA, Oct. 2000. Sandia National Laboratories.

[54] T.S. Li, S.M. Wong, Y.C. Hon, C.G. Armstrong, and R.M. McKeag. Smoothing by optimization for a quadrilateral with invalid elements. *Finite Elements in Analysis and Design*, 34: 37–60, 2000.

[55] R.V. Garimella, M.J. Shashkov, and P.M. Knupp. Optimization of surface mesh quality using local parametrization. In *Proc.* 11^{th} *Int. Meshing Roundtable*, pages 41–52, Ithaca, NY, Sept. 2002. Sandia National Laboratories.

[56] J.R. Shewchuk. Two discrete optimization algorithms for the topological improvement of tetrahedral meshes. Unpublished manuscript, 2002.

[57] M. Aiffa and J.E. Flaherty. A geometrical approach to mesh smoothing. *Comput. Methods Appl. Mech. Engrg.*, 192:4497–4514, 2003.

[58] M. Brewer, L.A. Freitag-Diachin, P.M. Knupp, T. Leurent, and D. Melander. The mesquite mesh quality improvement toolkit. In *Proc.* 12^{th} *Int. Meshing Roundtable*, pages 239–250, Santa Fe, New Mexico, USA, 2003. Sandia National Laboratories.

[59] P.M. Knupp. A method for hexahedral mesh shape optimization. *Int. J. Numer. Meth. Engng*, 58:319–332, 2003.

[60] J.M. Escobar, E. Rodríguez, R. Montenegro, G. Montero, and J.M. González-Yuste. Simultaneous untangling and smoothing of tetrahedral meshes. *Comput. Methods Appl. Mech. Engrg.*, 192:2775–2787, 2003.

[61] R.V. Garimella, M.J. Shashkov, and P.M. Knupp. Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Comput. Methods Appl. Mech. Engrg.*, 193 (9-11):913–928, Mar. 2004.

[62] I.B. Semenova, V.V. Savchenko, and I. Hagiwara. Two techniques to improve mesh quality and preserve surface characteristics. In *Proc.* 13^{th} *Int. Meshing Roundtable*, pages 277–288, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories.

[63] J.M. Escobar, G. Montero, R. Montenegro, and E. Rodríguez. An algebraic method for smoothing surface triangulations on a local parametric space. *Int. J. Numer. Meth. Engng*, 66:740–760, 2006.

[64] S.K. Khattri. A new smoothing algorithm for quadrilateral and hexahedral meshes. *Lecture Notes in Computational Science*, 3992:239–246, 2006.

[65] M. Berzins. A solution-based triangular and tetrahedral mesh quality indicator. *SIAM J. Sci. Comput.*, 19:2051–2069, 1998.

[66] P.J. Frey and H. Borouchaki. Geometric evaluation of finite element surface meshes. *Finite Elements in Analysis and Design*, 31:33–53, 1998.

[67] J. Dompierre, P. Labbé, F. Guibault, and R. Camarero. Proposal of benchmarks for 3D unstructured tetrahedral mesh optimization. In *Proc.* 7^{th} *Int. Meshing Roundtable*, pages 459–478, Dearborn, MI, USA, 1998. Sandia National Laboratories.

[68] P.J. Frey and H. Borouchaki. Surface mesh quality evaluation. *Int. J. Numer. Meth. Engng*, 45:101–118, 1999.

[69] D.A. Field. Qualitative measures for initial meshes. *Int. J. Numer. Meth. Engng*, 47:887–906, 2000.

[70] P.M. Knupp. Algebraic mesh quality metrics. *SIAM J. Sci. Comput.*, 23(1):193–218, 2001.

[71] P.P. Pébay and T.J. Baker. A comparison of triangle quality measures. In *Proc.* 10^{th} *Int. Meshing Roundtable*, pages 327–340, Newport Beach, CA, Oct. 2001. Sandia National Laboratories.

[72] J.R. Shewchuk. What is a good linear finite element? interpolation, conditioning, anisotropy and quality measures. University of California at Berkeley, 2002.

[73] P.M. Knupp. Algebraic mesh quality metrics for unstructured initial meshes. *Finite Elements in Analysis and Design*, 39:217–241, 2003.

[74] P.P. Pébay. Planar quadrilateral quality measures. *Engineering with Computers*, 20:157–173, 2004.

[75] P. Labbé, J. Dompierre, M.-G. Vallet, F. Guibault, and J.-Y. Trepanier. A measure of the conformity of a mesh to an anisotropic metric. In *Proc.* 10^{th} *Int. Meshing Roundtable*, pages 319–326, Newport Beach, CA, 2001. Sandia National Laboratories.

[76] P. Labbé, J. Dompierre, M.-G. Vallet, F. Guibault, and J.-Y. Trépanier. A universal measure of the conformity of a mesh with respect to an anisotropic metric field. *Int. J. Numer. Meth. Engng*, 61:2675–2695, 2004.

[77] J. Dompierre, M.-G. Vallet, P. Labbé, and F. Guibault. An analysis of simplex shape measures for anisotropic meshes. *Comput. Methods Appl. Mech. Engrg.*, 194:4895–4914, 2005.

[78] Y. Sirois, J. Dompierre, M.-G. Vallet, and F. Guibault. Measuring the conformity of non-simplicial elements to an anisotropic metric field. *Int. J. Numer. Meth. Engng*, 64:1944–1958, 2005.

[79] S.A. Canann, J.R. Tristano, and M.L. Staten. An approach to combined laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *Proc. 7^{th} Int. Meshing Roundtable*, pages 479–494, Dearborn, MI, USA, Oct. 1998. Sandia National Laboratories.

[80] L.A. Freitag-Diachin, P.M. Knupp, T. Munson, and S. Shontz. A comparison of inexact newton and coordinate descent mesh optimization techniques. In *Proc. 13^{th} Int. Meshing Roundtable*, pages 243–254, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories.

[81] L.A. Freitag. On combining Laplacian and optimization-based mesh smoothing techniqes. In *AMD-vol. 220 Trends in Unstructured Mesh Generation*, pages 37–43, 1997.

[82] H.N. Djidjev. Force-directed methods for smoothing unstructured triangular and tetrahedral meshes. In *Proc. 9^{th} Int. Meshing Roundtable*, pages 395–406, New Orleans, CA, Oct. 2000. Sandia National Laboratories.

[83] K.E. Jansen, M.S. Shephard, and M.W. Beall. On anisotropic mesh generation and quality control in complex flow problems. In *Proc. 10^{th} Int. Meshing Roundtable*, pages 341–349, Newport Beach, CA, Oct. 2001. Sandia National Laboratories.

[84] N. Troyani, A. Pérez, and P. Baíz. Effect of finite element mesh orientation on solution accuracy for torsional problems. *Finite Elements in Analysis and Design*, 41:1377–1383, 2005.

[85] J. Peraire, J. Peiro, and K. Morgan. Adaptive remeshing for three dimensional compressible flow computation. *J. Comput. Phys.*, 103:269–285, 1992.

[86] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specifications. Part I. Algorithms. *Finite Elements in Analysis and Design*, 25:61–83, 1997.

[87] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specifications. Part II. Applications. *Finite Elements in Analysis and Design*, 25:85–109, 1997.

[88] J. Dompierre, M.-G. Vallet, Y. Bourgault, M. Fortin, and W.G. Habashi. Anisotropic mesh adaptation: Towards user-independent, mesh-independent and solver-independent CFD. Part III: Unstructured meshes,. *Int. J. Numer. Meth. Fluids*, 39(8):675–702, 2002.

[89] X. Li. *Mesh Modification Procedures for General 3-D Non-manifold Domains.* PhD thesis, Rensselear Polytechnic Institute, August 2003.

[90] X. Li, M.S. Shephard, and M.W. Beall. 3D anisotropic mesh adaptation by mesh modification. *Comput. Methods Appl. Mech. Engrg.*, 194:4915–4950, 2005.

[91] K. Shimada, A. Yamada, and T. Itoh. Anisotropic triangulation of parametric surfaces via close packing of ellipsoids. *Int. J. Comput. Geom. Appl.*, 10(4):417–440, 2000.

[92] S.H. Lo and W.X. Wang. Generation of anisotropic mesh by ellipse packing over an unbounded domain. *Engineering with Computers*, 20:372–383, 2005.

[93] M.-G. Vallet. *Génération de maillages éléments finis anisotropes et adaptatifs.* PhD thesis, Université Pierre et Marie Curie, Paris VI, France, 1992.

[94] H. Borouchaki and P.J. Frey. Adaptive triangular-quadrilateral mesh generation. *Int. J. Numer. Meth. Engrg*, 41:915–934, 1998.

[95] M.J. Castro Díaz, F. Hecht, B. Mohammad, and O. Pironneau. Anisotropic unstructured mesh adaptation for flow simulations. *Int. J. Numer. Meth. Fluids*, 25:475–491, 1997.

[96] M. Spivak. *Calculus on Manifolds.* W.A. Benjamin, Inc. New York, Amsterdam, 1965.

[97] H. Borouchaki, F. Hecht, and P.J. Frey. Mesh gradation control. *Int. J. Numer. Meth. Engrg*, 43:1143–1165, 1998.

[98] M. Vigo, N. Pla, and P. Brunet. Directional adaptive surface triangulation. *Computer Aided Geometric Design*, 16:107–126, 1999.

[99] M. Vigo and N. Pla. Computing directional constrained Delaunay triangulations. *Computers & Graphics : Technical Section*, 24:181–190, 2000.

[100] R.B. Simpson. Geometry independence for a meshing engine for 2D manifolds. *Int. J. Numer. Meth. Engng.*, 60:675–694, 2004.

[101] T. Jurczyk and B. Głut. Metric 3D surface mesh generation using coordinate transformation method. In *Proc. of Int. Conf. on Computer Methods and Systems CMS'05*, volume 1, pages 395–405, Kraków, Poland, 2005.

[102] P. Kiciak. *Podstawy modelowania krzywych i powierzchni.* WNT Warszawa, 2000.

[103] V. Dolejší. Anisotropic mesh adaptation for finite volume and element methods on triangular meshes. *Comput. Visual Sci.*, 1:165–178, 1998.

[104] F. Alauzet and P.J. Frey. Estimateur d'erreur géométrique et métriques anisotropes pour l'adaptation de maillage. Partie I: aspects théoriques. Technical Report RR-4759, INRIA Rocquencour, 2003.

[105] P. Labbé, J. Dompierre, M.-G. Vallet, F. Guibault, and J.-Y. Trepanier. Measuring the metric conformity of a mesh. In *Proc.* 7^{th} *Int. Conf. on Numerical Grid Generation in Computational Field Simulations*, pages 809–818, Whistler, BC, Sept. 2000.

[106] J.S. Owen and S. Saigal. Surface mesh sizing control. *Int. J. Numer. Mesh. Engng*, 47: 497–511, 2000.

[107] X. Li, J-F. Remacle, N. Chevaugeon, and M.S. Shephard. Anisotropic mesh gradation control. In *Proc.* 13^{th} *Int. Meshing Roundtable*, pages 401–412, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories.

[108] V.D. Liseikin. *A Computational Differential Geometry Approach to Grid Generation.* Springer-Verlag Berlin Heidelberg New York, 2004.

[109] A. Liu and B. Joe. Relationship between tetrahedron shape measures. *BIT*, 34(2):268–287, 1994.

[110] A. Cunha, S. Canann, and S. Saigal. Automatic boundary sizing for 2D and 3D meshes. In *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pages 65–72. ASME, July 1997.

[111] S.J. Owen and S. Saigal. Neighborhood-based element sizing control for finite element surface meshing. In *Proc.* 6^{th} *Int. Meshing Roundtable*, pages 143–154, Park City, UT, USA, Oct. 1997. Sandia National Laboratories.

[112] W.R. Quadros, S.J. Owen, and M. Brewer K. Shimada. Finite element mesh sizing for surfaces using skeleton. In *Proc.* 13^{th} *Int. Meshing Roundtable*, pages 389–400, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories.

[113] J. Zhu, T. Blacker, and R. Smith. Background overlay grid size functions. In *Proc.* 11^{th} *Int. Meshing Roundtable*, pages 65–74, Ithaca, NY, Sept. 2002. Sandia National Laboratories.

[114] J. Zhu. A new type of size function respecting premeshed entities. In *Proc.* 12^{th} *Int. Meshing Roundtable*, pages 403–413, Santa Fe, New Mexico, USA, Sept. 2003. Sandia National Laboratories.

[115] W.R. Quadros, K. Shimada, and S.J. Owen. Skeleton-based computational method for the generation of a 3D finite element mesh sizing function. *Engineering with Computers*, 20: 249–264, 2004.

[116] T. Jurczyk and B. Głut. Adaptive control space structure for anisotropic mesh generation. In *Proc. of ECCOMAS CFD 2006 European Conference on Computational Fluid Dynamics*, Egmond aan Zee, The Netherlands, 2006.

[117] B. Głut, T. Jurczyk, and M. Pietrzyk. Adaptacja siatek w modelowaniu metoda elementów skończonych procesów przepływu ciepła. *Informatyka w Technologii Materiałów*, 1(2):90–103, 2001.

[118] B. Głut, T. Jurczyk, and M. Pietrzyk. Zastosowanie siatek anizotropowych w modelowaniu procesów metoda elementów skończonych. In F. Grosman, A. Piela, J. Kusiak, and M. Pietrzyk, editors, *Proc. of VIII Conf. KomPlasTech 2001*, pages 35–42, Korbielów, Poland, 2001.

[119] B. Głut and T. Jurczyk. Generowanie siatek adaptacyjnych dla metody elementów skończonych. In *Proc. VIII Warsztaty Naukowe PTSK*, Gdańsk, Poland, 2001.

[120] B. Głut, T. Jurczyk, and M. Pietrzyk. Adaptacja siatki w symulacji procesów w metalurgii. In J. Kusiak, M. Pietrzyk, F. Grosman, and A. Piela, editors, *Proc. of IX Conf. KomPlasTech 2002*, pages 63–70, Szczawnica, Poland, 2002.

[121] B. Głut, T. Jurczyk, and M. Pietrzyk. Adaptive mesh generation for non-steady state heat transport problems. In H.A. Mang, F.G. Rammerstorfer, and J. Eberhardsteiner, editors, *Proc. Fifth World Congress on Computational Mechanics (WCCM V)*, Vienna, Austria, 2002. http://wccm.tuwien.ac.at.

[122] B. Głut, T. Jurczyk, and M. Pietrzyk. Adaptacja siatki w symulacji procesu przemiany fazowej. In *Proc. X Warsztaty Naukowe PTSK*, Zakopane, Poland, 2003.

[123] B. Głut and T. Jurczyk. Wielokryterialna adaptacja siatek dwuwymiarowych. In D. Szeliga, M. Pietrzyk, and J. Kusiak, editors, *Proc. of XIII Conf. KomPlasTech 2006*, pages 53–60, Szczawnica, Poland, 2006.

[124] B. Głut, T. Jurczyk, and M. Pietrzyk. Adaptive remeshing in the FE modeling of moving boundary problems. In N.-E. Wiberg and P. Diez, editors, *Proc. of ADMOS2003 Conf. on Adaptive Modeling and Simulation*, Goeteborg, Sweden, 2003.

[125] B. Głut and T. Jurczyk. Mesh adaptation based on discrete data. *Lecture Notes in Computer Science*, 3911:559–566, 2006.

[126] B. Głut and T. Jurczyk. Definition and interpolation of discrete metric for mesh generation on 3D surfaces. *Computer Science, Annual of University of Science and Technology*, 7:89–103, 2005.

[127] B. Głut, T. Jurczyk, and J. Kitowski. Anisotropic volume mesh generation controlled by adaptive metric space. In *Proc. of Int. Conf. NUMIFORM'07*, pages 233–238, Porto, Portugal, 2007.

[128] B. Głut and T. Jurczyk. Quality and efficiency of FEM mesh adaptation with respect to control space. In *Proc. SIAM Conf. on Computational Science and Engineering*, Orlando, Florida, USA, 10-15 February 2005.

[129] T. Jurczyk and B. Głut. Efficiency aspects of control space definition for the generation of surface meshes. In *Proc. SIAM Conf. on Computational Science and Engineering*, Orlando, Florida, USA, 10-15 February 2005.

[130] P.L. George and F. Hermeline. Delaunay's mesh of a convex polyhedron in dimension d - application to arbitrary polyhedra. *Int. J. Num. Meth. Eng.*, 33:975–995, 1992.

[131] B. Joe. Construction of k-dimensional Delaunay triangulations using local transformations. *SIAM J. Sci. Comput.*, 14(6):1415–1436, 1993.

[132] P.L. Chew. Guaranteed-quality triangular meshes. Technical Report TR 89-983, Department of Computer Science, Cornell University, Ithaca, NY, 1989.

[133] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. Technical Report TR UCB/CSD 92/694, University of California at Berkely, Berkely California, 1992.

[134] J.R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. *Lecture Notes in Computer Science*, 1148:203–222, May 1996.

[135] J.R. Shewchuk. Mesh generation for domains with small angles. In *Proc.* 16^{th} *Symposium on Computational Geometry*, pages 111–112. ACM, 2000.

[136] S. Rebay. Efficient unstructured mesh generation by means of Delaunay triangulation and bowyer-watson algorithm. *Journal Of Computational Physics*, 106:125–138, 1993.

[137] Q. Du and M. Gunzburger. Grid generation and optimization based on centroidal Voronoï tessellations. *Applied Mathematics and Computation*, 133:591–607, 2002.

[138] H. Borouchaki, F. Hecht, E. Saltel, and P.-L. George. Reasonably efficient Delaunay based mesh generator in 3 dimensions. In *Proc.* 4^{th} *Int. Meshing Roundtable*, pages 3–14, Albuquerque, NM, USA, Oct. 1995. Sandia National Labs.

[139] M.-C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *Int. J. Numer. Meth. Engng*, 40:3313–3324, 1997.

[140] D. Marcum and N. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA J.*, 33(9):1625, 1995.

[141] A. El-Hamalawi. A 2D combined advancing front-Delaunay mesh generation scheme. *Finite Elements in Analysis and Design*, 40:967–989, 2004.

[142] M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[143] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, May 1995.

[144] Ch. Boivin and C. Ollivier-Gooch. Guaranteed-quality triangular mesh generation for domains with curved boundaries. *Int. J. Numer. Meth. Engng*, 55:1185–1213, 2002.

[145] G.L. Miller, S.E. Pav, and N.J. Walkington. An incremental Delaunay meshing algorithm. Technical Report 02-CNA-011, Center for Nonlinear Analysis, Carnegie Mellon University, 2002.

[146] J.R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, May 2002.

[147] G.L. Miller, S.E. Pav, and N.J. Walkington. When and why Ruppert's algorithm works. In *Proc.* 12^{th} *Int. Meshing Roundtable*, pages 91–102, Santa Fe, New Mexico, USA, 2003. Sandia National Laboratories.

[148] S.E. Pav. *Delaunay Refinement Algorithms.* PhD thesis, Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 2003.

[149] S. Secchi and L. Simoni. An improved procedure for 2D unstructured Delaunay mesh generation. *Advances in Engineering Software*, 34:217–234, 2003.

[150] S.-W. Cheng, T.K. Dey, E. Ramos, and T. Ray. Quality meshing for polyhedra with small angles. In *Proc.* 20^{th} *Annu. ACM Sympos. Comput. Geom.*, pages 290–299, 2004.

[151] H. Borouchaki, P.-L. George, and H. Lo. Optimal Delaunay point insertion. *Int. J. Numer. Meth. Engng*, 39(20):3407–3437, 1996.

[152] H. Borouchaki and P.-L. George. Aspects of 2-D Delaunay mesh generation. *Int. J. Numer. Meth. Engng*, 40:1957–1975, 1997.

[153] O. Devillers. Improved incremental randomized Delaunay triangulation. In *Proc.* 14^{th} *Symposium on Computational Geometry*, pages 106–115. ACM, 1998.

[154] B. Simpson, N. Hitschfeld, and M.-C. Rivara. Approximate shape quality mesh generation. *Engineering with Computers*, 17:287–298, 2001.

[155] D. Nowottny. Quadrilateral mesh generation via geometrically optimized domain decomposition. In *Proc.* 6^{th} *Int. Meshing Roundtable*, pages 309–320, Park City, UT, USA, Oct. 1997. Sandia National Labs.

[156] S.-W. Chae and J.-H. Jeong. Unstructured surface meshing using operators. In *Proc.* 6^{th} *Int. Meshing Roundtable*, pages 281–291, Park City, UT, USA, Oct. 1997. Sandia National Labs.

[157] T. Jurczyk and B. Glut. Triangular and quadrilateral meshes on 3D surfaces. In H.A. Mang, F.G. Rammerstorfer, and J. Eberhardsteiner, editors, *Proc. Fifth World Congress on Computational Mechanics (WCCM V)*, Vienna, Austria, 2002. http://wccm.tuwien.ac.at.

[158] B. Głut, T. Jurczyk, P.J. Matuszyk, and M. Pietrzyk. Anisotropic 2D meshes in finite element modelling. In *Proc.* 2^{nd} *European Conf. on Computational Mechanics*, pages 1865–1872, Kraków, Poland, 2001.

[159] T. Jurczyk and B. Głut. Metric 3D surface mesh generation using Delaunay criteria. *Lecture Notes in Computer Science*, 3992:302–309, 2006.

[160] T. Jurczyk and B. Głut. Generation of triangular meshes for complex domains. *Computer Science, Annual of University of Mining and Metallurgy*, 3:71–93, 2001.

[161] B. Głut, K. Boryczko, T. Jurczyk, and W. Alda. Optymalny wybór wezłów w aproksymacji brzegu dwuwymiarowego obszaru. In R. Tadeusiewicz, A. Ligeza, and M. Szymkat, editors, *Proc. III Konferencji Metody i Systemy Komputerowe w Badaniach Naukowych i Projektowaniu Inżynierskim*, pages 75–80, Kraków, Poland, 2001.

[162] B. Głut, K. Boryczko, T. Jurczyk, and W. Alda. Comparison of algorithm efficiency and mesh quality in Delaunay triangulation for complex 2D domains. In B.K. Soni, J.F. Thompson, J. Hauser, and P.R. Eiseman, editors, *Proc. 7th Int. Conf. on Numerical Grid Generation in Computational Field Simulations*, pages 789–798, Whistler, British Columbia, Canada, 2000.

[163] B. Głut, K. Boryczko, T. Jurczyk, W. Alda, and J. Kitowski. High quality 2d mesh generation based on Delaunay triangulation. In M. Bubak, J. Mościński, and M. Noga, editors, *Proc. SGI Users' Int. Conf. SGI2000*, pages 334–343, Kraków, Poland, 2000.

[164] T. Jurczyk and B. Głut. Generation of good quality quadrilateral elements for anisotropic surface meshes. In N.-E. Wiberg and P. Diez, editors, *Proc. of ADMOS2003 Conf. on Adaptive Modeling and Simulation*, Goeteborg, Sweden, 2003.

[165] T. Jurczyk and B. Głut. Konstrukcja i optymalizacja anizotropowych siatek czworokatnych na powierzchniach trójwymiarowych. In *Proc. X Warsztaty Naukowe PTSK*, Zakopane, Poland, 2003.

[166] Y.K. Lee and C.K. Lee. A new indirect anisotropic quadrilateral mesh generation scheme with enhanced local mesh smoothing procedures. *Int. J. Numer. Meth. Engng*, 58:277–300, 2003.

[167] J.R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proc.* 4^{th} *Symposium on Computational Geometry*, pages 86–95. ACM, 1998.

[168] S.W. Cheng and S.H. Poon. Graded conforming Delaunay tetrahedralization with bounded radius edge ratio. In *Proc.* 14^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 295–304, Baltimore, MD, 2003. ACM.

[169] S.E. Pav and N.J. Walkington. Robust three dimensional Delaunay refinement. In *Proc.* 13^{th} *Int. Meshing Roundtable*, pages 145–156, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories.

[170] B.H.V. Topping, J. Muylle, P. Iványi, R. Putanowicz, and B. Cheng. *Finite Element Mesh Generation*. Saxe-Coburg Publications, Scotland, 2004.

[171] P.-L. George and H. Borouchaki. Premières expériences de maillage automatique par une méthode de Delaunay anisotrope en trois dimensions. Rapport Technique 272, INRIA, 2002.

[172] J.R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, May 1997.

[173] B. Kaan Karamete, M.W. Beall, and M.S. Shephard. Triangulation of arbitrary polyhedra to support automatic mesh generators. *Int. J. Numer. Meth. Engng*, 49:167–191, 2000.

[174] B. Joe. Construction of three-dimensional triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[175] L.P. Chew. Guaranteed-quality Delaunay meshing in 3D (short version). In *Proc.* 13^{th} *Symposium on Computational Geometry*, page 391–393. ACM, 1997.

[176] X.-Y. Li and S.-H. Teng. Generating well-shaped Delaunay meshes in 3D. In *Proc.* 12^{th}. *Annu. ACM-SIAM Sympos. Discrete Alg.*, pages 28–37, 2001.

[177] S.-W. Cheng, T.K. Dey, H. Edelsbrunner, M.A. Facello, and S.-H. Teng. Sliver exudation. *J. ACM*, 47:883–904, 2000.

[178] J.R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput Geom*, 18(3):305–363, 1997.

[179] T. Jurczyk, B. Głut, and P. Breitkopf. Parallel 3D mesh generation using geometry decomposition. In *Proc. of Int. Conf. NUMIFORM'07*, pages 1579–1584, Porto, Portugal, 2007.

[180] M.W. Beall and M.S. Shephard. A general topology-based mesh data structure. *Int. J. Numer. Meth. Engng*, 40:1573–1596, 1997.

[181] R.V. Garimella. Mesh data structure selection for mesh generation and FEA applications. *Int. J. Numer. Meth. Engng.*, 55(4):451–478, 2002.

[182] R.V. Garimella. MSTK - a flexible infrastructure library for developing mesh based applications. In *Proc.* 13^{th} *Int. Meshing Roundtable*, pages 203–212, Williamsburg, Virginia, USA, 2004. Sandia National Laboratories. http://math.lanl.gov/~rao/Meshing-Projects/MSTK/.

[183] J.-F. Remacle and M.S. Shephard. An algorithm oriented mesh database. *Int. J. Numer. Meth. Engng*, 58:349–374, 2003. http://www.scorec.rpi.edu/AOMD.

[184] T. Jurczyk and B. Głut. Organization of the mesh structure. *Lecture Notes in Computer Science*, 3037:646–649, 2004.

[185] T. Jurczyk, B. Głut, and J. Kitowski. An empirical comparison of decomposition algorithms for complex finite element meshes. *Lecture Notes in Computer Science*, 2328:493–501, 2001.

[186] T. Jurczyk and B. Głut. Domain decomposition coupled with Delaunay mesh generation. *Lecture Notes in Computer Science*, 2329:353–360, 2002.

[187] B. Głut, T. Jurczyk, P. Breitkopf, A. Rassineux, and P. Villon. Geometry decomposition strategies for parallel 3D mesh generation. In *Proc. of Int. Conf. on Computer Methods and Systems CMS'05*, volume 1, pages 443–450, Kraków, Poland, 2005.

Appendices

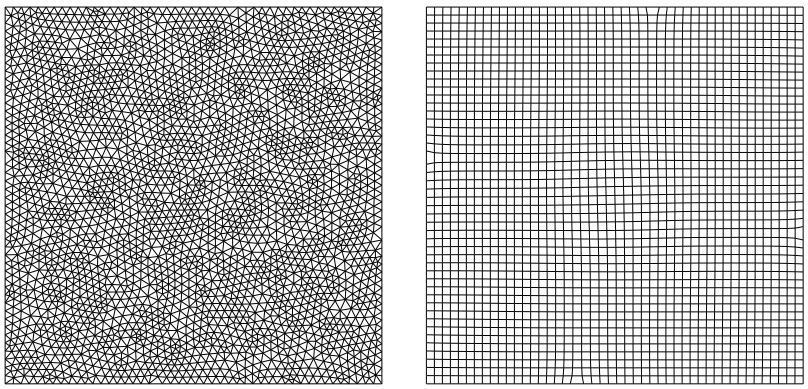
Appendix A

Examples

The following examples illustrate some typical meshes referenced in this thesis together with basic statistical description. The meshes are shown in simplified versions (with reduced number of elements) in order to show their structure while maintaining the necessary clarity of figures.

A.1 Surface Meshes

The mesh RECT1 (Fig. A.1) is an example of a simple, isotropic non-graded mesh for a rectangular domain.



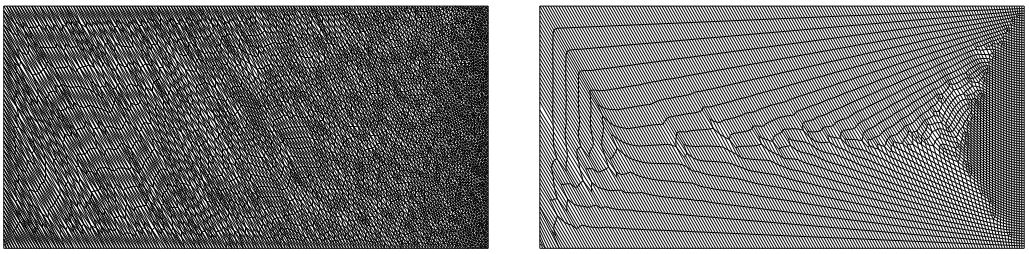
(a) triangular (b) quadrilateral

Figure A.1: Regular mesh RECT1

The anisotropic meshes AN1 (Fig. A.2) was generated on a plane with rectangular boundary $u \in [0, 2], v \in [0, 1]$ and sizing metric defined by function $f_m : (u, v) \rightarrow (\alpha, h_1, h_2)$:

$$f_m = (\pi/6, k/100, k(0.0101 - 0.005u)) \ , \tag{A.1}$$

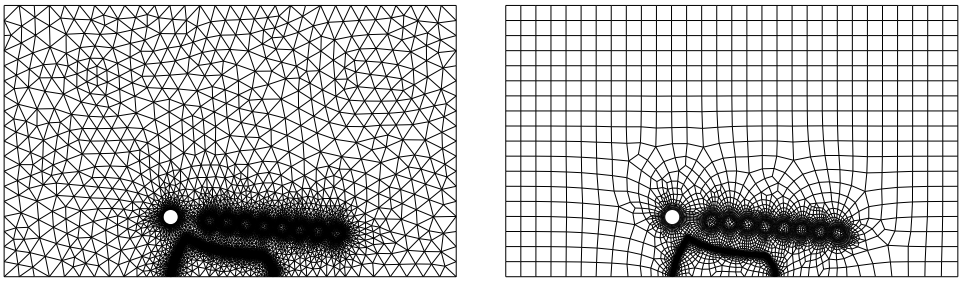
where coefficient k is used to control the size of created meshes.



(a) triangular (b) quadrilateral

Figure A.2: Anisotropic mesh AN1

The mesh NISA1 (Fig. A.3) is a discretization of a planar rectangular domain with sub-domains and specified element sizing along some contours.



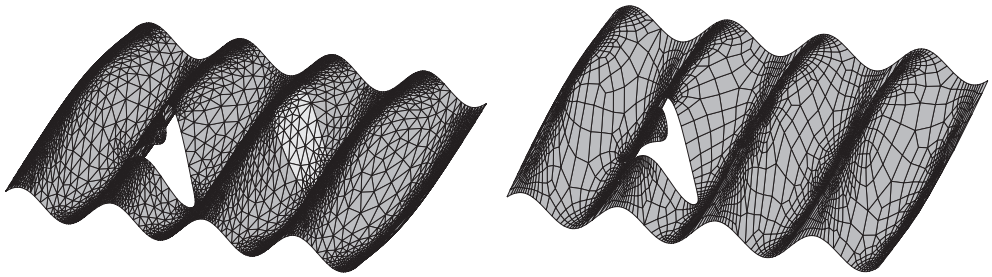
(a) triangular (b) quadrilateral

Figure A.3: Graded mesh NISA1

The meshes SURF0A (Fig. A.4), SURF0B and SURF0C were created on a surface patch

$$\mathbf{p} : (u, v) \rightarrow (u, v, 1.5e^{-0.001(u^2+v^2))\sin(2u)\cos(0.2v)}) \tag{A.2}$$

where $u, v \in [-6, 6]$, with additional subareas: circular (filled with other material) and defined by spline contour (empty). The curvature ratio for automatic geometry recognition was set for meshes SURF0A, SURF0B and SURF0C as $\gamma_c = 0.15, 0.05$ and 0.02.



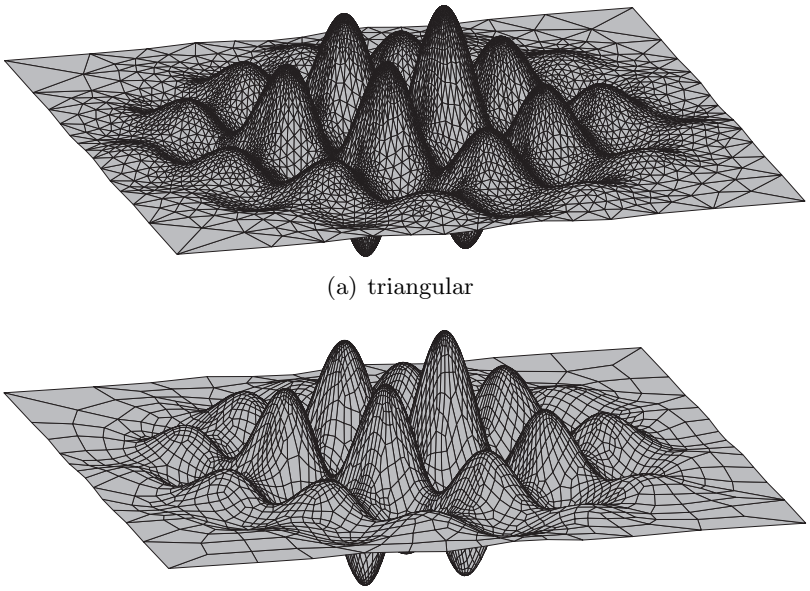
(a) triangular (b) quadrilateral

Figure A.4: Surface mesh SURF0

The mesh SURF1 (Fig. A.5) was created on a surface patch

$$\mathbf{p} : (u, v) \rightarrow (u, v, 2.5e^{-0.1(u^2+v^2))} \sin(2u) \cos(2v)) \tag{A.3}$$

where $u, v \in [-6, 6]$. The sizing metric was automatically recognized from the curvature of the surface and boundary with default parameters.



(a) triangular

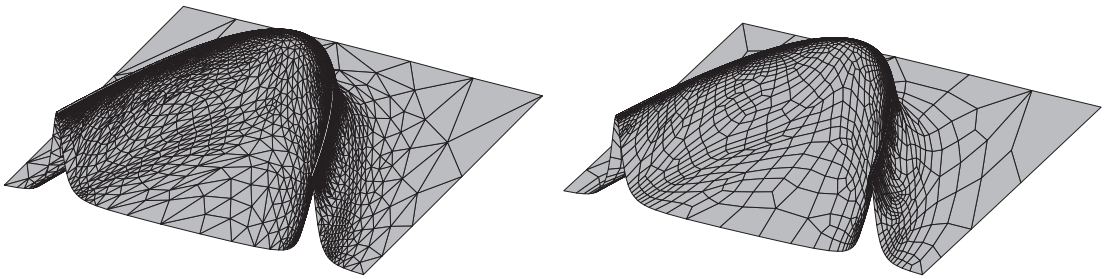
(b) quadrilateral

Figure A.5: Surface mesh SURF1

The mesh SURF2 (Fig. A.6) presents a discretization of a surface patch

$$\mathbf{p} : (u, v) \rightarrow (u, v, \frac{1}{0.4 + (\frac{u^2}{2} + v - \frac{5}{4})^2}) \tag{A.4}$$

where $u, v \in [-5, 5]$. The sizing metric was automatically recognized from the curvature of the surface and boundary with default parameters.

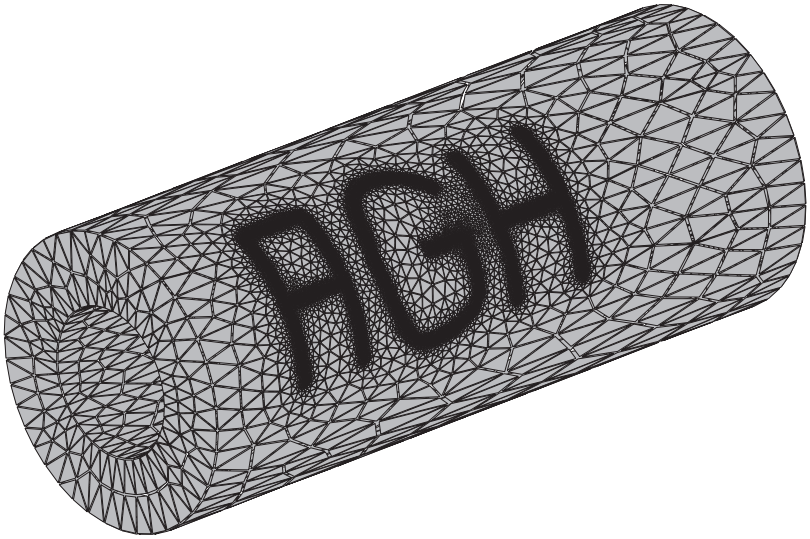


(a) triangular

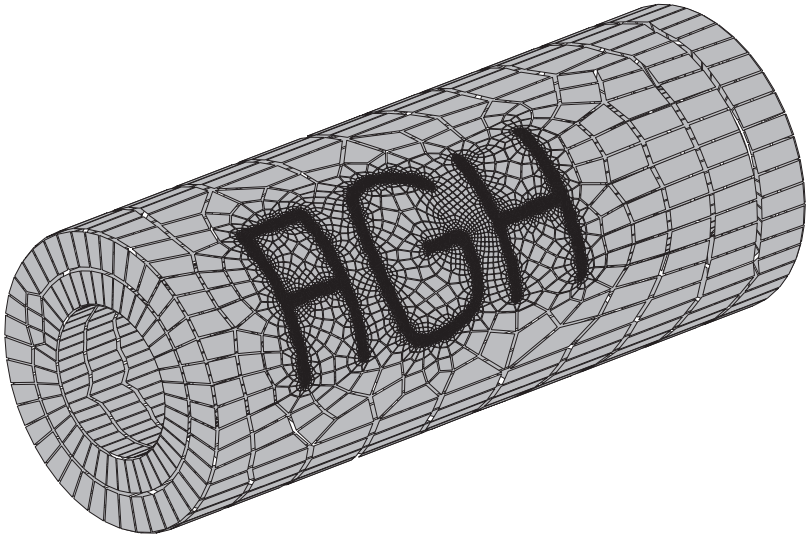
(b) quadrilateral

Figure A.6: Surface mesh SURF2

The mesh AGH1 (Fig. A.7) demonstrates the introduction of extended curvilinear metric sources on a cylindrical surface. These user-defined sources are combined with the information gathered from the geometry of the modeled domain.



(a) triangular



(b) quadrilateral

Figure A.7: Surface mesh AGH1

The numbers of elements and quality coefficients μ_e (average edge lengths in metric space) for both triangular and quadrilateral meshes are gathered in Table A.1.

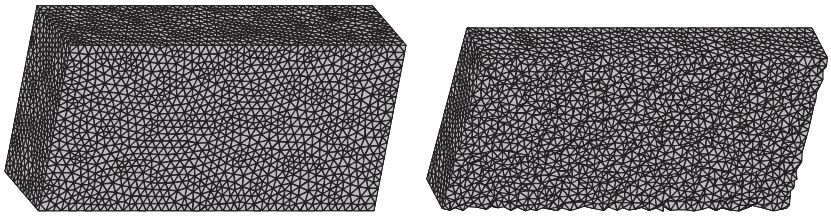
A.2 Volume Meshes

Figure A.8 presents a tetrahedral mesh CUBE1 created for cubicoidal domain with uniform element sizing. The presented cross-sections of volume meshes were created by showing

Table A.1: Surface mesh statistics (NT – number of triangles, μ_e, σ_e – average length and standard deviation of edges length in metric space for triangular (T) and quadrilateral (Q) meshes)

	NT	$\mu_e^{(T)}$	$\sigma_e^{(T)}$	NQ	$\mu_e^{(Q)}$	$\sigma_e^{(Q)}$
RECT1A	46312	1.001	0.139	19612	1.010	0.034
RECT1B	1152486	1.007	0.152	498278	1.002	0.021
AN1A	11098	1.002	0.141	5036	0.977	0.107
AN1B	217550	0.997	0.152	102603	0.948	0.186
NISA1	34574	0.976	0.165	16281	0.931	0.172
SURF0A	12262	0.959	0.175	5016	0.980	0.242
SURF0B	86830	0.986	0.154	36816	0.999	0.199
SURF0C	512300	0.997	0.144	216242	1.001	0.169
SURF1	20648	0.947	0.192	8466	0.976	0.261
SURF2	45436	0.950	0.167	17305	1.018	0.231
AGH1	59754	0.977	0.160	23949	1.015	0.231

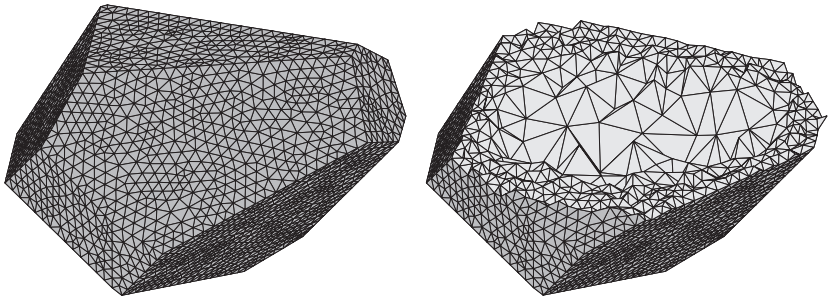
only tetrahedra placed on one side of the selected plane.



(a) mesh (b) cross-section

Figure A.8: Tetrahedral mesh CUBE1

The polyhedron (Fig. A.9) is also discretized with uniform sizing on boundary facets, but the size of tetrahedra in the mesh POLYHEDRON1 increases inside the domain, with the prescribed gradation ratio.

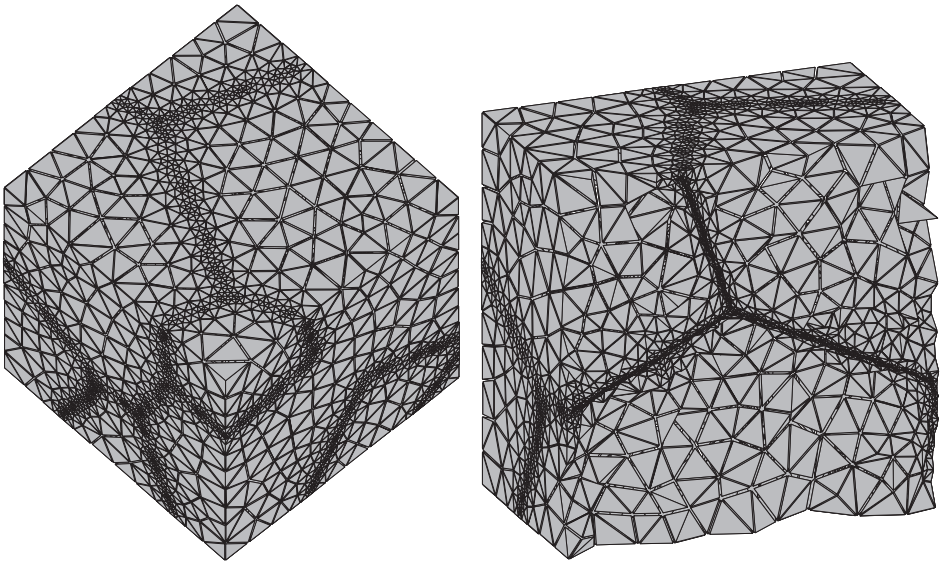


(a) mesh (b) cross-section

Figure A.9: Tetrahedral mesh POLYHEDRON1

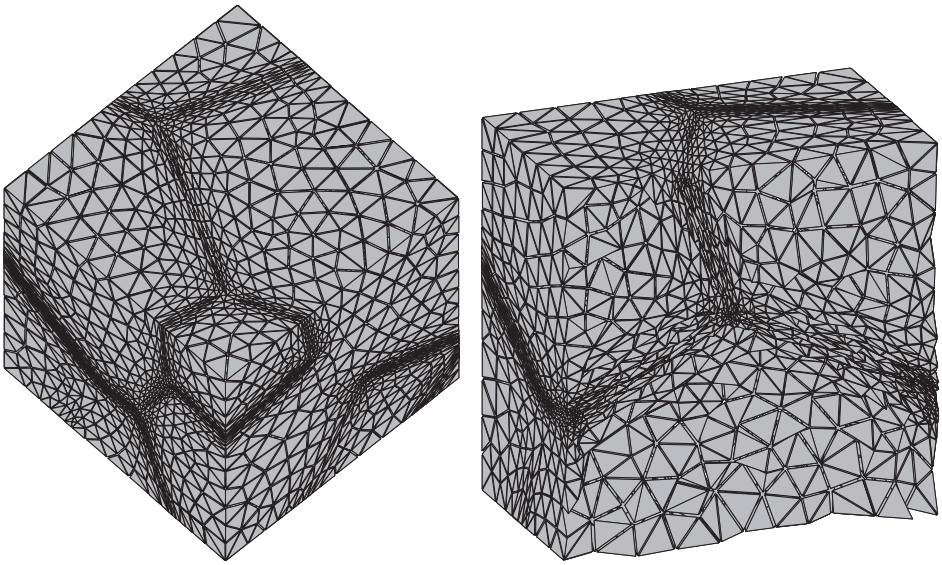
For meshes CUBE2 (Fig. A.10) and CUBE3 (Fig. A.11) the cubicodial domain was also used. However, the mesh sizing field was additionally enriched with a set of inner planar

metric sources enforcing smaller elements. The first mesh (CUBE2) is isotropic (which was obtained by setting the maximum anisotropy ratio $\gamma_a = 1$). For the latter one (CUBE3) this value was set to $\gamma_a = 25$, producing an anisotropic mesh.



(a) mesh (b) cross-section

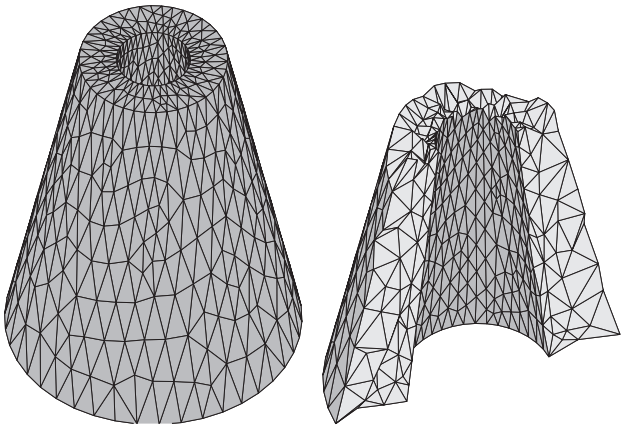
Figure A.10: Tetrahedral mesh CUBE2



(a) mesh (b) cross-section

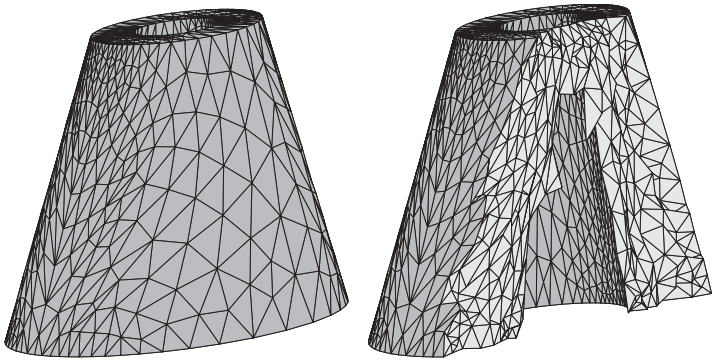
Figure A.11: Tetrahedral mesh CUBE3

The discretization of a geometric domain in the shape of a truncated cone with a hollow is illustrated on Fig. A.12 (CONE1). The mesh CONE2 (Fig. A.13) was generated for a similar model with additional stretching in one of the dimensions.



(a) mesh (b) cross-section

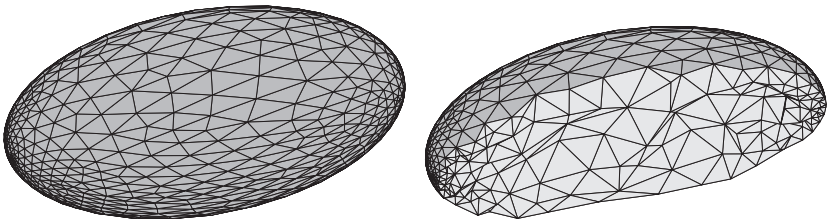
Figure A.12: Tetrahedral mesh CONE1



(a) mesh (b) cross-section

Figure A.13: Tetrahedral mesh CONE2

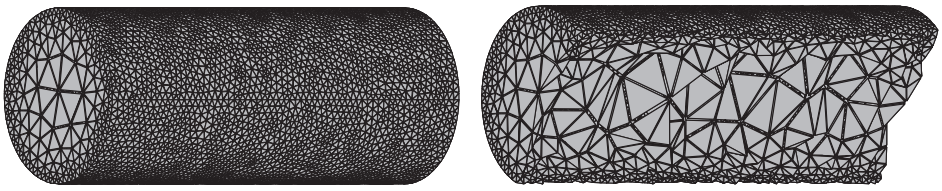
Figure A.14 presents the mesh ELLIPSOID1 created for ellipsoidal domain.



(a) mesh (b) cross-section

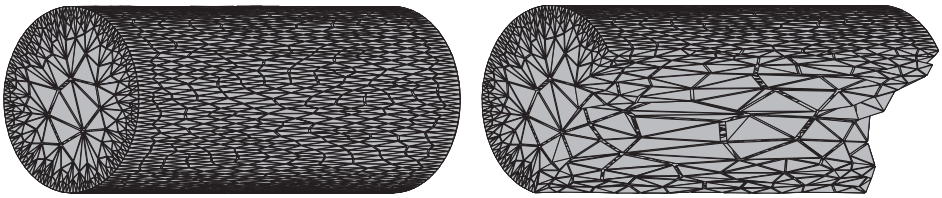
Figure A.14: Tetrahedral mesh ELLIPSOID1

The meshes CYLINDER1 (Fig. A.15) and CYLINDER2 (Fig. A.16) were created for cylindrical domain. The only difference between these two results is the selected maximum anisotropy ratio ($\gamma_a = 1$ for CYLINDER1, and $\gamma_a = 10$ for CYLINDER2).



(a) mesh (b) cross-section

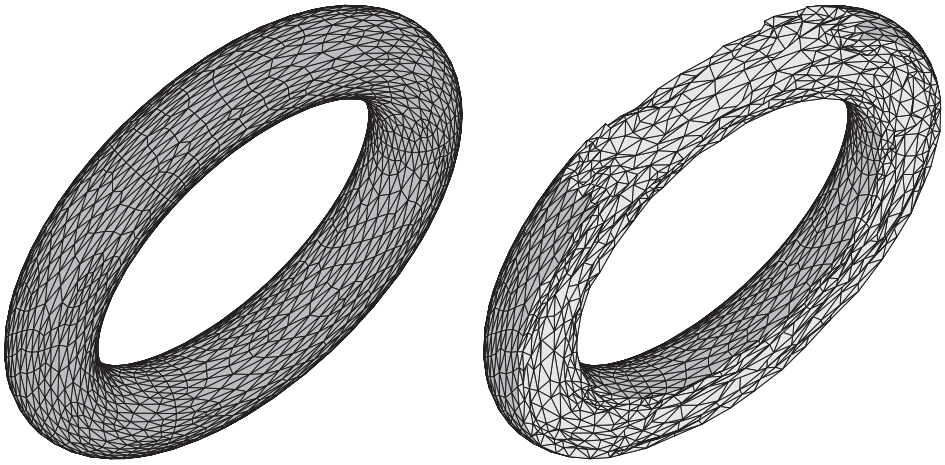
Figure A.15: Tetrahedral mesh CYLINDER1



(a) mesh (b) cross-section

Figure A.16: Tetrahedral mesh CYLINDER2

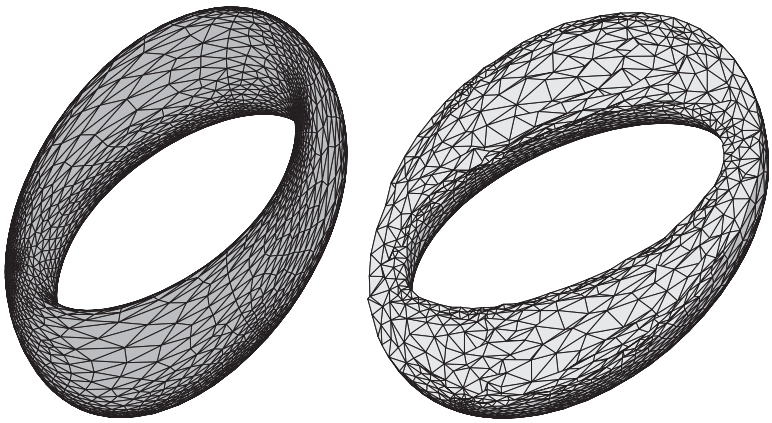
The mesh TORUS1 (Fig. A.17) presents the discretization of a toroidal domain.



(a) mesh (b) cross-section

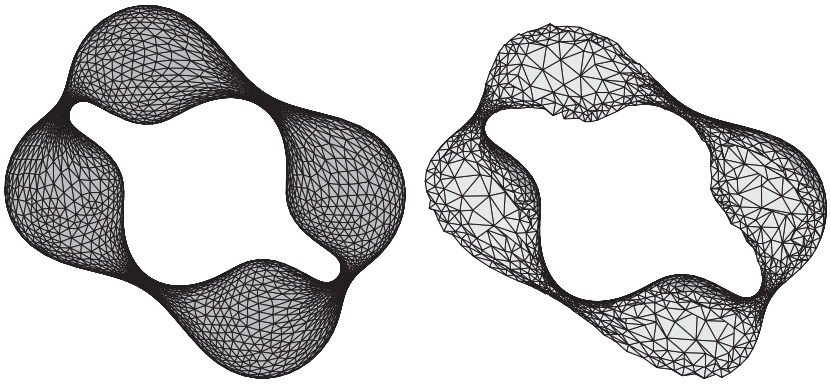
Figure A.17: Tetrahedral mesh TORUS1

Figures A.18 and A.19 show meshes TORUS2 and TORUS3 generated for toroidal domain with variable radii. The mesh SPIRAL1 (Fig. A.20) is a discretization of a spiral domain constructed from a transformed torus and parts of ellipsoid.



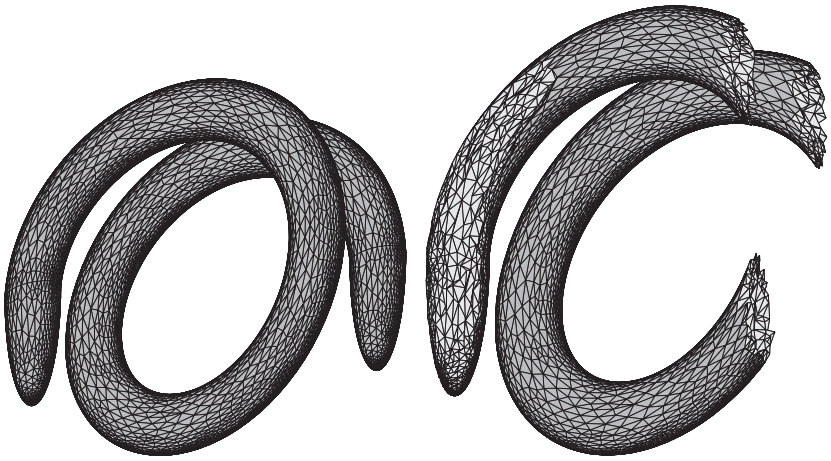
(a) mesh (b) cross-section

Figure A.18: Tetrahedral mesh TORUS2



(a) mesh (b) cross-section

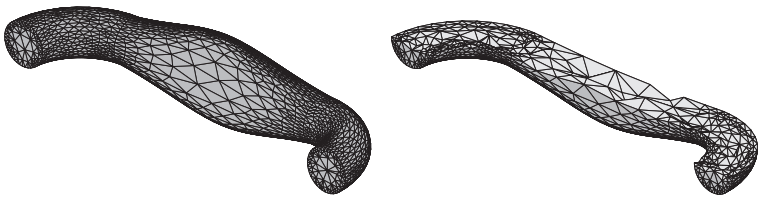
Figure A.19: Tetrahedral mesh TORUS3



(a) mesh (b) cross-section

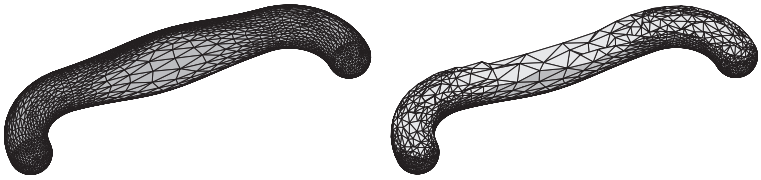
Figure A.20: Tetrahedral mesh SPIRAL1

The meshes COMP1 (Fig. A.21), COMP2 (Fig. A.22), COMP3 (Fig. A.23) and COMP4 (Fig. A.24) were created as the discretization of geometric domains with more complicated boundary description.



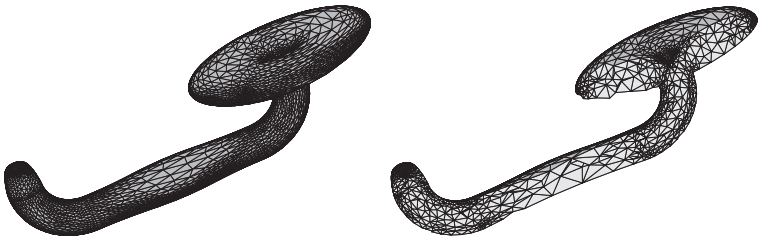
(a) mesh (b) cross-section

Figure A.21: Tetrahedral mesh COMP1



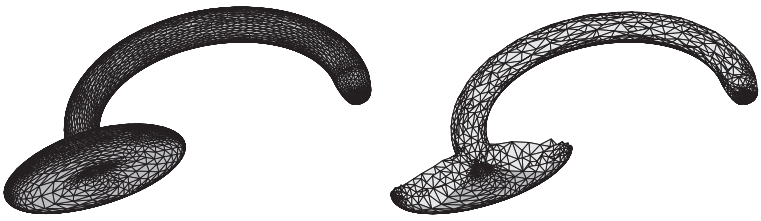
(a) mesh (b) cross-section

Figure A.22: Tetrahedral mesh COMP2



(a) mesh (b) cross-section

Figure A.23: Tetrahedral mesh COMP3



(a) mesh (b) cross-section

Figure A.24: Tetrahedral mesh COMP4

Table A.2: Volume mesh statistics (NP – number of points, NT – number of tetrahedra, μ_e, σ_e – average length and standard deviation of edges length in metric space

	NP	NT	μ_e	σ_e
CUBE1	17490	92902	1.035	0.161
POLYHEDRON1	6756	32637	1.006	0.197
CUBE2	107739	618691	0.972	0.312
CUBE3	11948	66528	0.919	0.219
CONE1	2590	12244	0.984	0.247
CONE2	4149	20290	0.974	0.266
ELLIPSOID	2044	9523	0.980	0.236
CYLINDER1	55471	284513	1.009	0.202
CYLINDER2	5076	23580	0.971	0.254
TORUS1	7503	37918	0.973	0.224
TORUS2	9233	45850	0.979	0.228
TORUS3	14546	72258	0.981	0.227
SPIRAL	19697	95664	0.968	0.224
COMP1	5645	27345	0.981	0.233
COMP2	7083	34316	0.987	0.226
COMP3	14210	71476	0.973	0.246
COMP4	12415	62361	0.970	0.261

The summary statistics of all presented volume meshes (including the number of mesh points, elements and quality evaluation based on an average metric length of edges) are presented in Table A.2.

List of Symbols and Abbreviations

Abbreviation	Description	Definition
P, Q	point in space	
$\langle \cdot, \cdot \rangle$	scalar product	
$\cdot \times \cdot$	cartesian product	
$\|\| \cdot \|\|$	Euclidean norm	
$d(\cdot, \cdot)$	Euclidean distance	
$\lfloor \cdot \rfloor$	floor function	
$\mathbf{n}$	normal vector	
Ω	discretization domain	page 7
$\mathcal{T}_h$	mesh	page 7
V	vertex of $\mathcal{T}_h$	page 7
E	edge of $\mathcal{T}_h$	page 7
F	face of $\mathcal{T}_h$	page 7
K	element of $\mathcal{T}_h$	page 7
$\mathcal{M}(P)$	metric tensor at point P	page 10
h	length of element in main metric direction	page 10
$\mathbf{M}(P)$	metric transformation tensor at point P	page 10
$\mathbf{p}$	parametric surface patch	page 11
$\mathcal{S}$	parametric domain of patch	page 11
$\mathbf{M_s}(P)$	sizing transformation matrix at point P	page 11
$\mathbf{M_p}(P)$	parameterization matrix at point P	page 12
$\mathcal{S}^{\mathcal{M}}, \Omega^{\mathcal{M}}$	image of mapping introduced by $\mathbf{M}$	page 13
$P^{\mathcal{M}}$	point transformed to metric space	page 13
$d_{\mathcal{M}}(\cdot, \cdot)$	distance between two points in metric space	page 13
$\mathcal{S}^{\mathcal{R}}$	image of mapping introduced by $\mathbf{M_p}$	page 13
$P^{\mathcal{R}}$	point transformed to $\mathcal{S}^{\mathcal{R}}$	page 13
$d_{\mathcal{R}}(\cdot, \cdot)$	distance between two points transformed with $\mathbf{M_p}$	page 13
$\min_{\mathcal{M}}(\cdot, \cdot)$	intersection of two transformation tensors	page 14
$\delta_{\mathcal{M}}(\cdot, \cdot)$	difference between two transformation tensors	page 17
$\gamma_{\mathcal{M}}$	metric gradation ratio	page 17
ϵ_δ	threshold for metric comparison	page 17
$\mathcal{M}_K$	metric tensor of element K	page 18

Abbreviation	Description	Definition
γ_c	curvature sizing ratio	page 19
γ_a	maximum anisotropy ratio	page 19
$\gamma_{h\max}$	maximum length ratio	page 19
$\gamma_{h\min}$	minimum length ratio	page 19
$\mathbf{c}, \mathbf{c_p}$	parametric contour	page 20
γ_{a_c}	anisotropy ratio for contour curvature	page 20
$\overline{\eta}_{\mathcal{M}}$	mean ratio quality coefficient of $\mathcal{T}_h$ in metric space	page 21
$\sigma_{\eta_{\mathcal{M}}}$	standard deviation of mean ratio	
$\overline{\mu}_{\mathcal{M}}$	edge length quality coefficient of $\mathcal{T}_h$ in metric space	page 21
$\sigma_{\mu_{\mathcal{M}}}$	standard deviation of edge length	
$\overline{\mathcal{E}}_K$	metric non-conformity quality coefficient of $\mathcal{T}_h$	page 22
CS	control space	page 23
ACS	adaptive control space	page 24
$\epsilon_{\mathcal{M}}$	threshold for metric distance between two points	page 28
$l_{\mathcal{M}\max}$	metric length threshold for adaptation	page 28
DT	Delaunay triangulation	page 43
EC	criterion of empty circumsphere/circumcircle	page 46
IA	criterion of inner angles	page 46
NP	number of vertices in $\mathcal{T}_h$	
NE	number of edges in $\mathcal{T}_h$	
NE(V)	number of edges adjacent to vertex V	
NT	number of triangles/tetrahedra in $\mathcal{T}_h$	
NQ	number of quadrilaterals in $\mathcal{T}_h$	
τ	mesh generation speed	
$q(K)$	quality of element K for mesh refinement	page 53
q_t	quality threshold for mesh refinement	page 54
CCS	placement of nodes in circumcenter of K	page 54
MLE	placement of nodes in middle of longest edge of K	page 54
q_{Bt}	refinement threshold for boundary triangulation	page 73
q_{It}	threshold for selection of insertion method	page 87

List of Figures

List of Tables

List of Algorithms