NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF CIVIL ENGINEERING
STRUCTURAL ENGINEERING DEPARTMENT
INSTITUTE OF STRUCTURAL ANALYSIS & ANTISEISMIC RESEARCH

A finite element mesh smoothing method based on geometric element transformations

D I S S E R T A T I O N   T H E S I S

Ph.D.

DIMITRIOS VARTZIOTIS
Civil Engineer (M.Sc.)
Aeronautical & Space Engineer (M.Sc.)
University of Stuttgart

SUPERVISOR
MANOLIS PAPADRAKAKIS
Professor NTUA

ATHENS MARCH 2013

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF CIVIL ENGINEERING
STRUCTURAL ENGINEERING DEPARTMENT
INSTITUTE OF STRUCTURAL ANALYSIS & ANTISEISMIC RESEARCH

A finite element mesh smoothing method based on geometric element transformations

D I S S E R T A T I O N   T H E S I S

Ph.D.

DIMITRIOS VARTZIOTIS
Civil Engineer (M.Sc.)
Aeronautical & Space Engineer (M.Sc.)
University of Stuttgart

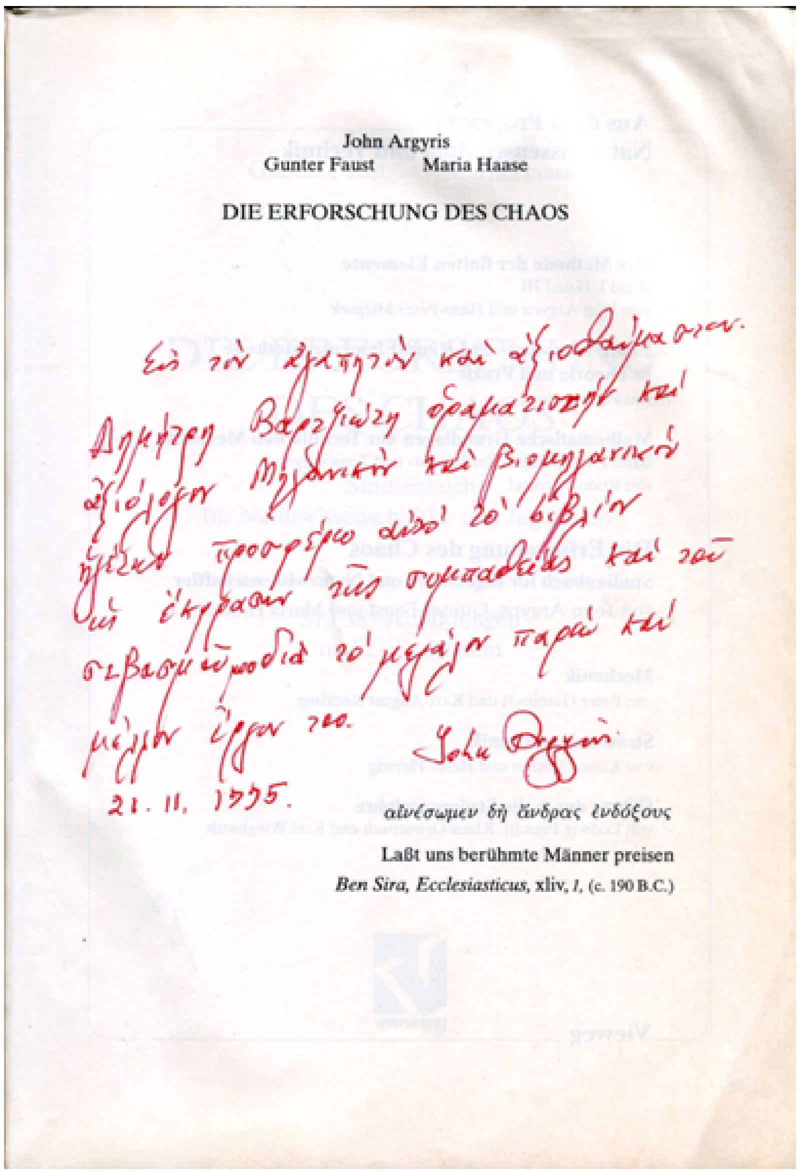
SUPERVISOR
MANOLIS PAPADRAKAKIS
Professor NTUA

ATHENS MARCH 2013

Dedication

Στη μνήμη του καθηγητή και φίλου μου John Argyris (1913-2004).

In grateful memory of John Argyris (1913-2004), my teacher and friend.



John Argyris
Gunter Faust Maria Haase

DIE ERFORSCHUNG DES CHAOS

Εἰς τὸν ἀγαπητὸν καὶ ἀξιοθαύμαστον
Δημήτρη Βαρτζιώτη ὁραματιστὴν καὶ
ἀξιόλογον Μηχανικὸν καὶ Βιομηχανικὸν
ἡγέτην προσφέρω αὐτὸ τὸ βιβλίον
ὡς ἔκφρασιν τῆς ἐκτιμήσεως καὶ τοῦ
σεβασμοῦ μου διὰ τὸ μεγάλο παρὸν καὶ
μέλλον ἔργον του.

John Argyris

21. 11. 1995.

αἰνέσωμεν δὴ ἄνδρας ἐνδόξους
Laßt uns berühmte Männer preisen
Ben Sira, Ecclesiasticus, xliv, *1*, (c. 190 B.C.)

To the esteemed Dimitris Vartziotis, visionary and leading figure of Engineering and Industry, I offer this book as an expression of my appreciation and respect of his grand work, present and future.

21.11.1995 John Argyris

Acknowledgements

I would like to thank Professor Manolis Papadrakakis for the excellent cooperation. From the first moment he has supported my ideas for the development of the finite element mesh smoothing method. I thank him for his systematic guidance of the work published in journals of high scientific standing.

I thank also Iraklis Goudas, Theodoros Athanasiadis, Bernd Schwald and Simon Huggenberger from the companies NIKI MEPE Digital Engineering, Greece and TWT GmbH Science & Innovation, Germany. I would especially like to thank Joachim Wipper from TWT GmbH for his support at various stages of my research. I thank them all for their participation in individual topics within published articles.

I also thank Vivi Zapsa, Irini Mantziou and Christina Kontosoglou for their services with the editing and organization of original documents and files from my studies and the carrying out of various communications.

I thank NIKI MEPE and TWT GmbH for granting permission to the use of finite element industrial models.

Abstract

The geometric element transformation method (GETMe) described in this work is a novel approach to finite element mesh smoothing. It is based on regularizing element transformations represented by geometric constructions, which iteratively improve the regularity and thus the quality of single elements. Such transformations are defined and analyzed for arbitrary polygonal elements as well as for the most common volumetric element types. In a first stage, GETMe smoothing improves global mesh quality by averaging the new node positions obtained by element-wise applied transformations. In a second stage, minimal element quality is improved by successively transforming the worst elements of the mesh. These stages are generalized by an adaptive variant of GETMe smoothing and aspects of implementation and parallelization are discussed. Various numerical examples presented in this work confirm that GETMe smoothing leads to superior mesh quality if compared to other geometry-based methods like variants of Laplacian smoothing. In terms of resulting mesh quality, it can even compete with state of the art global optimization-based techniques, despite being conceptually significantly simpler and considerably faster. On the example of Poisson's equation it will also be shown numerically that this smoothing method is particularly suitable, from a finite element application point of view, since it leads to a significant reduction of discretization errors within the first few smoothing steps requiring only little computational effort.

Contents

List of Figures

List of Tables

Introduction

The generation of quality meshes plays a fundamental role in computations based on the finite element method, since mesh quality affects the accuracy of the simulation as well as the accuracy of direct solvers and the convergence of iterative solvers. A large number of mesh generation methods for planar polygonal meshes, surface meshes, and volume meshes have been proposed [1–4]. Amongst the latter Delaunay-based approaches to tetrahedral mesh generation are quite popular, since they are comparatively well suited for automation, of reasonable complexity, and since they result in good quality meshes. However, the use of hexahedral meshes is preferred, for example, in elastic/elasto-plastic solid continuum analysis or CFD applications, since more accurate results can be obtained with a lower number of elements [5, 6]. Compared to the case of tetrahedral mesh generation, meshing complex volume models by hexahedra is considerably harder and the achieved degree of generality and automation is still far from the state obtained in the case of tetrahedral meshing. Therefore, the usage of hybrid meshes and corresponding mesh generators has been proposed in order to combine the advantages of both element types [7–9]. In addition, pyramids or prisms serve as transition elements to connect triangular and quadrilateral element faces. Depending on the application, hybrid meshes combining other types of elements might be preferred such as tetrahedral meshes with prismatic boundary layers in computational biofluid dynamic computations [10].

Usually mesh improvement techniques are incorporated into the corresponding mesh generation processes or applied afterwards [8, 11]. They can be based on mesh topology altering operations like face swapping, node insertion, and removal [12–14] or smoothing techniques. The latter improve mesh quality by mesh topology preserving node movements only. The most noted approach of this kind is Laplacian smoothing, where node positions are iteratively updated by the arithmetic mean of neighboring node positions [15–17]. Hence, Laplacian smoothing is computationally inexpensive and algorithmically well suited for meshes combining arbitrary element types. However, due to its simple mesh node averaging scheme Laplacian smoothing does not incorporate element quality aspects and thus can lead to mesh quality deterioration and the generation of inverted elements. This can be avoided by accomplishing node movements only if quality is improved or by using relaxation techniques. Such methods are denoted as smart or constrained Laplacian smoothing [18, 19].

Better results with respect to a given quality criterion can be obtained by local optimization, that is by placing nodes in order to maximize the quality of adjacent elements. Since this results in a higher computational effort, a combined approach of Laplacian smoothing and local optimization was proposed in [18], where optimization is only accomplished in problematic regions of the mesh.

Focusing on overall mesh quality, a natural choice is to use a global optimization approach incorporating quality numbers of all mesh elements into one objective function [20, 21]. Here, all free vertices are moved simultaneously within a single iteration. Again, this comes at the expense of a higher implementational and computational effort. Furthermore, mathematical aspects, like optimization techniques and the proper choice of quality metrics, play a fundamental role [22, 23]. For example, in the case of mixed mesh smoothing, quality metrics have to be applicable and balanced in their behavior for all element types under consideration. In addition, optimization-specific additional requirements like differentiability, or practice-oriented requirements like evaluation efficiency, have to be taken into account.

A more generalized approach to the construction of appropriate and flexible objective functions with respect to optimization aims is given by the target-matrix paradigm [24]. Here, for each sample point of the mesh two matrices are required. One is the Jacobian matrix of the current mesh, the other is a target-matrix representing the desired Jacobian matrix in the optimal mesh. Usually sample points are given by element nodes and the target matrices are derived from type dependent reference elements. Using objective functions based on these matrices allow mesh quality to be defined on a higher conceptual level.

Compared to the simple geometry-based approach of smart Laplacian smoothing, global optimization-based methods result in meshes of higher quality at the expense of significantly higher computational and implementational effort. This is one of the reasons why smart Laplacian smoothing was able to retain its popularity even though the resulting mesh quality is inferior. As a way out of this dilemma between mesh quality and computational effort, the geometric element transformation method (GETMe), which is the main subject of this work, has been proposed as a new geometry-based approach to mesh smoothing [25–32]. Various numerical tests indicate that it results in high quality meshes, comparable to those obtained by global optimization-based approaches, within significantly shorter runtimes. Being based on regularizing single element transformations it is easy to implement and is well suited for parallelization. Furthermore, due to its universal approach GETMe smoothing has proven its effectiveness and practicability for various kinds of mesh types including triangular and mixed element planar or surface meshes, as well as all-tetrahedral, all-hexahedral, and mixed volume meshes.

The driving force behind GETMe smoothing are regularizing element transformations which, if applied iteratively, lead to more regular hence better quality elements. Such transformations for polygons with an arbitrary number of nodes have been proposed and analyzed in this work and published in [33–36]. In the case of polyhedral elements a special transformation for tetrahedra based on opposite face normals is proposed [29] and a more generalized transformation based on dual elements, which is applicable to all common volumetric elements, is developed [31].

GETMe smoothing consists of two consecutively applied steps focusing on different aspects of mesh quality. In the first GETMe smoothing step, all elements are transformed simultaneously and new node positions are obtained as weighted means of the resulting transformed element nodes, thus improving overall mesh quality. This is repeated until overall mesh quality converges. In the second GETMe smoothing step, only the worst elements are iteratively transformed in order to improve their quality numbers, thus specifically improving minimal element quality. In both cases, mesh quality is incorporated into the smoothing process by simple element quality dependent parameters and control mechanisms. Due to its general approach, GETMe smoothing can be applied to all types of meshes with elements for which regularizing transformations exist. Aiming at the generation of regular elements, GETMe smoothing significantly reduces the number of extremal angles, which makes this approach particularly suitable for finite element applications [37]. A combined approach of GETMe smoothing and global optimization for hexahedral mesh untangling and smoothing is proposed [38]. Here a variant of GETMe smoothing is applied first. Smoothing is terminated if mesh quality is sufficient. Otherwise optimization is applied in order to untangle or further improve the mesh.

An advanced version of GETMe smoothing, named GETMe adaptive, is proposed [32], which improves the former version with respect to the following aspects: Enhanced applicability and flexibility with an adaptive smoothing control, unified approach by incorporating both smoothing stages within one main smoothing loop, submesh smoothing instead of worst element smoothing further facilitating parallelization, and adaptive node relaxation instead of invalid element node resetting. Furthermore, from an algorithmic point of view smoothing control is simplified and the number of parameters is significantly reduced compared to GETMe smoothing. From an application point of view, the resulting mesh quality is improved whereas memory requirements and smoothing time are significantly reduced.

Outside the context of mesh smoothing, regularizing transformations also play a role in classical geometry. For example, the construction of similar figures on the sides of a polygon and the resulting symmetric polygons has fascinated mathematicians for over a century [39]. The most popular theorem in this area is Napoleon's Theorem [40] where a regular triangle is constructed within one transformation step by connecting the centers of equilateral triangles erected on each side of the initial triangle. Since then, this approach has been generalized with respect to the number of vertices of the initial polygon, the geometric construction schemes, the resulting symmetries, or the number of transformation steps [41–44]. One example is the Petr-Douglas-Neumann theorem [45], which transforms an n-gon within $n-2$ steps into an equilateral polygon. Another class of transformation schemes uses an infinite number of iteration steps [44, 46, 47].

In this work, aspects of the finite element method with respect to mesh generation and improvement will be discussed first in Chapter 1. After giving a brief overview of the history of the finite element in Section 1.1, aspects of mesh generation and improvement are discussed in Section 1.2 and Section 1.3, respectively. In preparation of the definition of the geometric element transformation method for finite element mesh smoothing, fundamentals of regularizing element transformations for polygonal elements will be discussed in Chapter 2. Here, the following polygon transformation will be analyzed. On each side of the polygon, similar isosceles triangles are erected. After that, the apices of these similar triangles are connected, which results in a new polygon. In contrast to existing approaches, the sequence resulting from iteratively applying the same transformation will be analyzed with respect to the base angle $\theta \in (0, \pi/2)$ of the isosceles triangles. This is done first for triangular elements by means of analysis in Section 2.1, which also discusses some variants of this transformation.

For polygons with an arbitrary number $n \geq 3$ of nodes it will be shown in Section 2.2 by means of linear algebra that there is a change in the geometry of limit figures of the sequences of scaled polygons at each characteristic angle $\theta_k = \pi(2k+1)/(2n)$, $k \in \{0, \ldots, \lfloor n/2 \rfloor - 1\}$, leading to a full classification of possible limit polygons. Whereas the polygons for θ within an interval bounded by characteristic angles are regular polygons or equilateral stars with possibly multiple vertices, the polygons for $\theta = \theta_k$, $k > 0$, are linear combinations of the neighboring limit polygons. Furthermore, the unscaled polygons degenerate to their common centroid, become bounded regular n-gons or grow infinitely in the case of θ being smaller, equal or larger than $\theta_0 = \pi/(2n)$.

Following the parameter based approach, more general polygon transformations based on similar triangles are analyzed in Section 2.3. This is done by using not only a given base angle $\theta \in (0, \pi/2)$, as in the case of isosceles triangles, but also a subdivision ratio $\lambda \in (0, 1)$ defining the apex perpendicular. Here, the matrix representation of the linear transformation is derived with respect to the construction parameters θ and λ. Furthermore, a combined transformation represented by a circulant Hermitian matrix is defined, which eliminates the rotational effect of the basic transformation, which is adverse in the mesh smoothing context. It is shown that the theorems of Napoleon and Petr-Douglas-Neumann can be naturally deduced by finding the roots of an explicit representation of the eigenvalues and that there are no other parameter combinations leading to equivalent construction schemes. Additionally, eigenpolygons, their basic properties, and the decomposition of polygons into eigenpolygons are described. Sequences of transformed polygons are also analyzed. Explicit representations of the parameter subdomains are derived, for which the sequence converges to specific eigenpolygons. Furthermore, parameter sets of eigenvalue intersections are extracted, for which linear combinations of up to three eigenpolygons occur as limit polygons. This leads to a full classification of similar triangle based transformations with respect to the construction parameters and the resulting limit polygons.

Regularizing transformations for tetrahedral, hexahedral, pyramidal and prismatic finite elements are presented in Chapter 3. Here, the mean ratio quality criterion is described first in Section 3.1, which provides a general approach to measure the regularity of polygonal and polyhedral elements. Section 3.2 describes a regularizing transformation for tetrahedral elements based on shifting the nodes by the scaled normals of the opposing element faces. A unified regularizing transformation scheme for all volumetric element types under consideration based on dual elements is given in Section 3.3. Properties of all transformations are analyzed and the regularizing effect is substantiated by numerical tests, which will also serve as basis for an adaptive choice of transformation parameters in the context of mesh smoothing.

The geometric element transformation method is discussed in Chapter 4. After giving a short overview of its fundamental concepts in Section 4.1, the simultaneous approach for mesh smoothing by simultaneously transforming all elements and obtaining new node positions by weighted node averaging is given in Section 4.2. A description of the sequential approach consisting of iteratively transforming only the worst quality element is given in Section 4.3. Both approaches are incorporated in the GETMe smoothing approach as is described in Section 4.4. Whereas these approaches use static concepts for smoothing control and a fixed number of elements to be transformed within one smoothing step, the adaptive approach given in Section 4.5 is based on smoothing a dynamic subset of elements in the second stage. Furthermore, instead of using a static concept for repairing inverted elements as in the case of the GETMe approach, GETMe adaptive incorporates a node-based adaptive relaxation scheme. Implementational aspects and the use of alternative quality criteria is addressed in Section 4.6 and Section 4.7, respectively.

An extensive collection of numerical smoothing results is given in Chapter 5 for planar polygonal meshes, polygonal surface meshes, all-tetrahedral, all-hexahedral and mixed volumetric meshes. Synthetical as well as real world models and meshes are considered covering a broad range of aspects and mesh element numbers. Comparisons with respect to resulting mesh quality, smoothing runtime, convergence behavior, and memory requirements are not only given for the described variants of GETMe smoothing, but also for variants of Laplacian smoothing and a state of the art global optimization-based approach, which are used for comparison.

Finally, Chapter 6 provides a detailed description of the impact of mesh smoothing on finite element solution efficiency and simulation accuracy in the context of the Poisson's equation. This is demonstrated for a number of comparatively large practice-oriented triangular, quadrilateral, tetrahedral and hexahedral meshes, using piecewise linear and quadratic basis functions. Detailed comparisons are performed with the results of area- or volume-weighted Laplacian smoothing, smart Laplacian smoothing, and a global optimization-based approach in terms of solution error norms, condition numbers and performance of system of linear equations solvers. By focusing on the differences in smoothing runtime behavior, in a novel approach finite element solution errors are derived after each smoothing iteration. As a result, this approach allows to analyze the impact of mesh smoothing on mesh quality and finite element solution accuracy with respect to smoothing time.

Chapter 1

The finite element method and its mesh creation

In the following, a brief history of the finite element method will be given. The role of finite element meshes and techniques for mesh generation and mesh improvement will be addressed. This is accompanied by a more detailed description of the mesh smoothing methods used for comparing results in the subsequent chapters.

1.1 A brief history of the finite element method

The finite element method (FEM) is a versatile numerical technique for computing approximate solutions to boundary value problems incorporating partial differential equations. Categories of application are equilibrium problems, eigenvalue problems and time-dependent problems originating in the context of structural analysis, heat transfer, fluid mechanics, electromagnetism, biomechanics, geomechanics, and acoustics. Key features of the finite element method are [48]: (a) the continuum is divided into a finite number of parts (elements), the behavior of which is specified by a finite number of parameters, and (b) the solution of the complete system as an assembly of its elements. Developing the finite element method as it is known today was a long evolutionary process in which various engineers, physicists and mathematicians have provided major contributions over several decades [49–52].

In World War II Argyris invented the finite element technique in the course of the stress analysis of the swept back wing of the twin engined Meteor Jet Fighter [53, 54]. In 1954 and 1955 he published a series of articles, also republished in the monograph [55], in which the matrix theory of structures is developed for discrete elements. It is shown that this is only a particular case of the general continuum in which stresses and strains have been specified, which leads to the concept of flexibility and stiffness. Argyris recognized the concept of duality between equilibrium and compatibility and provides equations relating stresses and strains to loads and displacements [49]. In the historical overview [50], Clough stated in 2004: "In my opinion, this monograph (...) certainly is the most important work ever written on the theory of structural analysis, ...".

Independent of the work of Argyris and from a mathematical point of view Courant developed in [56] the idea of minimizing a functional using linear approximation over subregions, with the values being specified at discrete points, which in essence become the node points of a mesh of elements. Courant solved the Saint-Venant's torsion of a square hollow box and in the appendix of his paper he introduced the idea of linear approximation over triangular areas [49]. Courant's contribution is based on earlier results by Rayleigh [57] and Ritz [58].

Under the guidance of Turner, Clough started in 1952 to work on in-plane stiffness matrices for 2D plates for analyzing the vibration properties of delta wings for Boeing. This resulted in solving plane stress problems by means of triangular elements whose properties were determined from the equations of elasticity theory as reported in [59]. This publication also addresses the question of convergence in the case of mesh refinement [49] and introduced the direct stiffness method for determining finite element properties [51]. Results of further treatments of Clough on the plane elasticity problem are reported in the publication [60], which also gives the verification that for known geometries and loading the stresses converged to the corresponding analytic solution [49]. This publication of Clough also introduced the name finite element method. As an explanation for his choice, Clough stated in a speech on the early history of the FEM given in [50]: "On the basis that a deflection analysis done with these new 'pieces' (or elements) of the structure is equivalent to the formal integration procedure of integral calculus, I decided to call the procedure the FEM because it deals with finite components rather than differential slices".

The first book on finite elements was published in 1967 by Zienkiewicz. The revised version [48] of this book is still one of the standard references for FEM. Being mainly introduced in the context of structural analysis, Zienkiewicz clarified the connection to function minimization techniques and opened the way to the analysis of field problems by the FEM [49]. Revealing the broad applicability of the finite element method, this was also the starting point for a vital research of the scientific community not only with respect to fields of application but also with respect to efficiency, effectivity and implementational aspects of the FEM.

This is indicated by Table 1.1, which lists the number of documents published over the last decades containing the key words "finite element" in their title. Results are given for the databases of Google Scholar (`http://scholar.google.de`), Scirus (`http://www.scirus.com`), and Zentralblatt MATH (`http:`

Table 1.1: Number of finite element publications

Interval	Scholar	Scirus	ZMATH	ZMATH*
1960–1969	965	60	54	59
1970–1979	9,090	1,572	1,513	1,542
1980–1989	15,900	3,763	3,679	7,934
1990–1999	19,500	8,351	5,254	14,257
2000–2009	36,300	19,937	5,959	20,399
2010–2012	16,200	21,169	1,423	4,879

`//www.zentralblatt-math.org`) as of 2012/10/19. This specific search does not cover the additional multitude of FEM publications omitting these key words in their title. This is addressed by the last column of Table 1.1 named ZMATH* also covering the publications containing the words "finite element" alternatively in their abstracts or reviews. These numbers impressively show the vivid development of the finite element method and its applications for more than 50 years.

1.2 Finite element mesh generation techniques

One step of the finite element method consists of tessellating the simulation domain into simple and finite elements, like triangles and quadrilaterals in 2D or tetrahedra and hexahedra in 3D, in order to define the trial functions approximating the solution of the boundary value problem to be solved. It is well known that the finite element solution accuracy and computational efficiency depend on the size and shape of the elements and hence on the quality of the underlying mesh [61]. For example, large element angles lead to large errors in the gradient, whereas small angles can significantly increase the condition of the stiffness matrix [62, 63] impairing the solution accuracy and the convergence properties of iterative solvers. Beyond that, requirements on finite element meshes depend heavily on the type of application [64]. Therefore, in the finite element simulation process mesh generation techniques play an important role and hence are a main topic of research. In the following, a brief overview of mesh generation techniques will be given based on [2–4].

Structured meshes are characterized by the property that all interior nodes of the mesh have an equal number of adjacent elements. A basic example for such meshes is a Cartesian grid consist of quadrilateral or hexahedral elements only. The main idea of all structured mesh generation techniques is to generate a grid for canonical domains and to map the resulting mesh to a physical domain defined by its boundary discretization [2]. The generation of structured meshes becomes difficult if the complexity of the physical domain increases. Also domain decomposition techniques can be applied in order to reduce the problem to local mappings, however, automation of such processes is not straight forward.

In contrast, unstructured meshes allow any number of elements to meet a single node and therefore provide a significantly greater flexibility in meshing arbitrarily complex domains. Therefore, these meshes are of greater practical importance and the following will address common unstructured mesh generation techniques only.

For simplicial mesh generation, that is meshes consisting entirely of simplices, the quadtree or octree method [65–67] uses hierarchical tree structures obtained by recursive subdivision of a square or cube containing the domain or surface until the desired resolution is reached. On each level, the obtained cells are classified into outer cells, which can be omitted, irregular cells, where boundary intersections have to be computed, and inner cells. On the last recursion level, irregular and inner cells are then meshed into triangles or tetrahedra. To ensure that element sizes do not change too dramatically, the maximum difference in subdivision levels between adjacent cells can be limited, hence leading to balanced trees. Also, subsequent mesh improvement steps are accomplished to increase mesh quality [68, 69].

In contrast, advancing front techniques progressively build elements inward starting from the boundary. This iterative technique proceeds by placing new nodes and the generation of new elements, which are connected to the existing elements, until the entire domain is meshed [15, 70, 71]. Common steps in advancing front techniques are: front analysis, internal point creation, element generation, validation, convergence control, mesh improvement [2]. Due to the high-quality point distribution, the advancing front method is used in many commercial meshing software packages [4].

Mesh generation by Delaunay triangulation belong to the most popular simplicial mesh generation techniques due to their simplicity, reliability, efficiency and favorable properties [2, 4, 72, 73]. Common steps in Delaunay-based mesh generation are: initial triangulation of the bounding box of the domain, boundary points insertion into the triangulation, original boundary recovery, internal point creation and insertion, mesh improvement. The triangulation process is based on the empty sphere criterion [74] stating that any node must not be contained within the circumsphere of any simplex within the mesh.

One of the remarkable properties of the Delaunay triangulation is its uniqueness for a non-degenerated set of nodes, i.e. a set of nodes where no three nodes are on the same line and no four nodes are on the same circle [75]. An essential property from a finite element application point of view is the fact that the minimum angle in a Delaunay triangulation is greater than the minimum angle in any other triangulation of the same nodes [75]. Computing the Delaunay triangulation of a set of n nodes can be accomplished in $\mathcal{O}(n \log n)$ as has been shown in [76]. A modified divide and conquer approach using pre-partitioning was shown in [77] to run in $\mathcal{O}(n \log \log n)$ expected time while retaining the optimal $\mathcal{O}(n \log n)$ worst case complexity. Alternative methods exist, which tetrahedralize volumetric models by a recursive subdivision approach [78, 79].

Hexahedral meshes are preferred over tetrahedral meshes in specific finite element and finite volume applications, like for example elastic or elasto-plastic solid continuum analysis or CFD applications, since more suitable trial functions can be defined on such tessellations resulting in a higher solution accuracy combined with a lower number of mesh elements [5, 6]. However, depending on the geometrical complexity of the simulation domain, the generation of good quality all-hexahedral meshes is in general considerably harder than the generation of tetrahedral meshes. Hence a variety of hexahedral mesh generators, as described e.g. in [80–82], are build on subdividing the initial domain into primitives. The latter can subsequently be meshed using for example sweeping, i.e. by moving a quadrilateral mesh from a source surface along a given path to a target surface while generating hexahedral elements [83–85]. Alternative subdomain meshing techniques can be based on octrees [67, 86, 87], advancing front like approaches as plastering [88] or Whisker weaving [89, 90]. The latter is based on the concept of the spatial twist continuum, which is the dual of the hexahedral mesh represented by an arrangement of intersecting surfaces which bisect hexahedral elements in each direction [4]. Furthermore, specialized algorithms exist for example in order to mesh composite domains consisting of heterogeneous materials with non-manifold region surfaces [11].

Since meshing complex volume models by hexahedra is considerably harder than by tetrahedra, the achieved degree of generality and automation is still far from the state obtained in the case of tetrahedral meshing. Therefore, the usage of hybrid meshes and corresponding mesh generators has been proposed in order to combine the advantages of tetrahedral and hexahedral elements [7–9]. In addition, pyramids and prisms serve as transition elements to connect triangular and quadrilateral element faces. Depending on the application, hybrid meshes combining other types of elements might be preferred such as tetrahedral meshes with prismatic boundary layers in computational biofluid dynamic computations [10].

1.3 Mesh smoothing

1.3.1 General overview

As has been noted in the previous section, mesh improvement methods are a key component of the most common mesh generation techniques for the finite element method since the shape and distribution of the mesh elements have impact on the trial functions and thus the numerical stability as well as solution accuracy [1, 2, 61, 63, 91–93]. Furthermore, mesh improvement methods are also applied in the process of surface reconstruction using scanner data or automated joining of different meshes [2, 94–98].

The various existing mesh improvement methods can roughly be classified into two categories: methods that use topological modifications and mesh smoothing methods preserving mesh topology by applying node relocations only. Typical topology altering improvement methods refine or coarsen the mesh by element splitting or merging, node insertion or deletion, local subdivision, and edges or face swapping [2, 12–14, 99]. In the case of simplicial meshes, these methods usually try to achieve the Delaunay property for the mesh or to ensure specific element properties with respect to minimum angle or edge ratios.

Mesh smoothing methods are of particular interest, if mesh interfaces, boundaries or even the complete mesh topology has to be preserved like in iterative solution methods and design optimization algorithms. Furthermore, due to the specific adjacency structure, topological modifications of hexahedral meshes are far more complex than in the case of tetrahedral meshes since such modifications usually propagate along complete sheets. Therefore, improving hexahedral mesh quality by smoothing is much simpler than improving it by topology altering operations.

One of the most popular smoothing methods is Laplacian smoothing, where each node is repeatedly replaced by the arithmetic mean of its edge connected neighbor nodes [15, 16]. This method is popular due to its simple implementation, low numerical complexity and fast convergence behavior. However, since Laplacian smoothing is not specifically geared towards improving element quality, possible issues of this method are deformation and shrinkage in the case of surface meshes, occurrence of inverted or invalid elements, and mesh quality deterioration. In particular, the method, in its basic form, is less suitable for meshes generated by adaptive octree-based methods [87] and mixed meshes, that is, meshes consisting of elements of different type like triangles and quadrilaterals [100].

Therefore, improved methods have been proposed to circumvent these problems [17, 101]. Amongst these are length weighted Laplacian smoothing, which uses averaging functions for neighboring nodes and angle based methods [18, 100, 102] comparing and adjusting angles incident to a node, thus reducing the risk of generating inverted elements. Alternatively, smart or constrained Laplacian smoothing incorporate quality aspects by applying node movements only if they lead to an improvement of local mesh quality with respect to a given metric [18, 19]. In the case of mixed meshes, significantly better results can be obtained by using weighted means [19, 102, 103]. A more detailed discussion of Laplacian smoothing and some of its variants is given in Section 1.3.2.

Whereas geometry-based methods, like Laplacian or angle-based smoothing, determine new node positions by geometric rules, local optimization techniques move nodes in order to optimize objective functions based on quality numbers of the incident elements [104–107]. Such methods effectively prevent the generation of degenerate or inverted elements [104, 108]. For example, [106] proposes a smoothing method by relocating interior vertices to the mass centers of their surrounding hexahedra. As in the case of smart Laplacian smoothing, the solution of this minimization problem is only applied if quality is improved. A node quality metric based local optimization method using a combined gradient driven and simulated annealing technique is presented in [109].

Due to the incorporation of element quality metrics and standard optimization techniques, like the conjugate gradient method or linear programming, the computational costs of local optimization techniques are increased if compared to geometry-based methods. Therefore, combined methods have been proposed using variants of Laplacian smoothing as well as local optimization techniques. Here, local optimization is only applied in the case of elements with a quality below a given threshold in order to find a balance between quality and computational effort [18, 19, 110].

Whereas local optimization techniques are well suited in order to efficiently resolve mesh problems locally, global optimization-based smoothing methods incorporating all mesh elements are geared towards effectively improving the overall mesh quality [20, 21, 111–115]. Due to their holistic approach, global optimization techniques result in a superior overall mesh quality. However, depending on the quality metrics and optimization techniques involved, these methods can be very demanding in terms of computation time, memory and implementational complexity. This can be circumvented to some extend by the use of streaming techniques [116]. A more in-depth description of a global optimization-based mesh smoothing approach and its main components is given in Section 1.3.3.

Specialized optimization-based methods exist in the case of surface meshes in order to preserve the given shape and its characteristics like discrete normals and curvature [117–119]. They are mainly based on using local parameterizations and modified objective functions. A combined global optimization approach improving element size as well as shape is presented in [113]. Here, smoothing is driven by minimizing Riemannian metric non-conformity. Since the objective function is not easily differentiable, a brute force approach is used for minimization.

All optimization-based methods have in common that the the proper choice of quality metrics and objective functions plays a crucial role [22, 23, 120]. For example, in the case of mixed mesh smoothing, quality metrics have to be applicable and balanced in their behavior for all element types under consideration. In addition, optimization-specific additional requirements, like differentiability or practice-oriented requirements like evaluation efficiency, have to be taken into account. Hence, a lot of effort has been put on finding suitable quality metrics.

Common quality metrics are based on geometric entities like angles, lengths, surface areas, as well as their aspect ratios [2]. However, in the context of mesh optimization, algebraic quality metrics are more suitable [22, 23, 121–123]. Furthermore, by a proper choice of the objective function and quality metrics, global optimization can also be used in order to untangle meshes [124]. In this context [24] provides a more generalized approach to the construction of appropriate and flexible objective functions with respect to optimization aims by introducing the target-matrix paradigm. Here, for each sample point of the mesh two matrices are required. One is the Jacobian matrix of the current mesh, the other a target-matrix representing the desired Jacobian matrix in the optimal mesh. Usually sample points are given by element nodes and the target matrices are derived from type dependent reference elements. Using objective functions based on these matrices allow mesh quality to be defined on a higher conceptual level. An example for this approach is the mean ratio criterion, for which the definition is given in Section 3.1. By analyzing two energy functions based on conformal and isoparametric mappings, this metric has been recently shown in [120] to be equivalent to the angle-preserving energy. Hence, mesh optimization results in minimizing the energy.

Due to the rapidly growing complexity of nowadays simulations, there is a great demand for fast mesh improvement methods providing high quality results. In this context, the computational effort of global optimization-based methods may become computationally too expensive. As an alternative, the geometric element transformation method (GETMe) has been proposed [27–32], which is also the main subject of this work. Instead of improving element shape based on solving optimization problems, GETMe achieves mesh improvement by applying specific geometric element transformations, which successively transform an arbitrary mesh element into its regular counterpart [31, 34–36]. This is combined with a relaxation and weighted node averaging scheme involving mesh quality. In doing so GETMe smoothing combines a global-optimization like smoothing effectivity with a Laplacian smoothing like runtime efficiency.

Further mesh smoothing approaches exist. For example, methods based on physical models like systems of springs [125], packings of bubbles [126], and solving a global system derived from local stiffness matrices [114] or statistical approaches [127]. In [128] the mesh is considered as a deformable solid. In doing so, smoothing of the mesh is treated as a matrix problem in finite elements. Alternatively, [129] proposes a method for improving hexahedral meshes by quasi-statistical modeling of mesh quality parameters and the approach given in [130] is based on space mapping techniques. In [131] mesh smoothing is conducted by solving a large nonlinear algebraic system obtained by Laplace-Beltrami equations for an unstructured mixed element volume mesh. However, solving the system by using Newton-Krylov methods is a critical task as issues in solver convergence observed in numerical examples have shown.

There also exist various smoothing methods, which are more intimately connected with the finite element approximation process, such as the a priori-based approach given in [132]. Here, optimization is based on a measure of the interpolation error associated with the finite element model. In contrast, a posteriori smoothing methods incorporate the information obtained by error estimates [133–135]. Such adaptive techniques usually apply mesh improvement and refinement within a finite element iteration scheme. They are particularly well suited for anisotropic problems or problems with singularities.

Each of the existing methods for mesh improvement faces some distinct problems. The topology based methods often rely heavily on the insertion of new elements or the splitting of old ones in order to improve mesh quality. In this case, the resulting mesh might be comprised of an excessive number of elements or elements of very small dimensions. Thus, numerical solvers typically used for FEM computations can be severely affected due to the need for unreasonable computing power or due to the ill-conditioned numerical models arising when miniscule elements exist. Furthermore, altering the number of elements makes the mapping of results in similar models cumbersome. On the other hand smoothing methods are restricted by the given mesh topology, since adverse topological configurations prevent smoothing methods from reaching high quality results. Thus it is natural to develop hybrid methods combining topological modifications, smoothing and optimization methods as proposed in [18, 19]. In these hybrids, each method can be used as a preconditioner for the other or iteratively until the desired mesh quality is achieved.

1.3.2 Laplacian smoothing and its variants

Due to its simplicity and efficiency, one of the most popular smoothing methods until today is the Laplacian smoothing approach. Let p_i denote an arbitrary node of the mesh and $J(i)$ the index set of all other mesh nodes, which are connected to p_i by an edge. The new position p'_i of p_i is derived as

$$p'_i := \frac{1}{|J(i)|} \sum_{j \in J(i)} p_j \,, \tag{1.1}$$

where $|J(i)|$ denotes the number of elements in the index set $J(i)$. As can be seen, p'_i is the arithmetic mean of its edge connected nodes [15, 16]. All nodes are updated according to (1.1) and this is iterated until the maximal node movement distance $\max_i |p'_i - p_i|$ drops below a given threshold. In the case of a rectangular grid, this method can be derived by the finite difference approximation of the Laplace operator, which also justifies its name. Iteratively updating the nodes in Laplacian smoothing implies that results depend on the order of the nodes. Furthermore, since no quality or validity criteria are incorporated, mesh quality can deteriorate, invalid elements can be generated.

A variant is the area- or volume-weighted Laplacian smoothing approach given in [136]. For an arbitrary mesh element E_j, $j \in I_E$, let c_j denote its centroid obtained as the arithmetic mean of all nodes of E_j, and v_j the area or volume of E_j in the 2D or 3D case, respectively. Within one smoothing loop, each interior node p_i is corrected by the movement vector

$$\Delta p_i := \frac{\sum_{j \in J(i)} v_j (c_j - p_i)}{\sum_{j \in J(i)} v_j} \,, \tag{1.2}$$

where $J(i)$ again denotes the index set of all elements attached to the node p_i. This is repeated until the maximal absolute movement $\max_i |\Delta p_i|$ drops a given threshold. In the current implementation used for comparing results in the subsequent sections, this threshold is given by 1% of the average length of all edges in the initial mesh. As in the case of Laplacian smoothing by successively updating all nodes, results of this approach depend on the node order. Furthermore, due to (1.2) and the geometric termination criterion, area- or volume-weighted Laplace does not involve the quality measures and thus cannot guarantee quality improvements and the validity of the resulting mesh.

These issues are addressed by the smart Laplacian smoothing approach given in [18]. It is based on the classical Laplacian smoothing scheme, where the new position of an interior node is derived as the arithmetic mean of all edge-connected nodes. However, this new node position is only set, if this leads to an increase of the arithmetic mean of quality values of all attached elements. Evaluating this local mean quality values before and after node movement results in a significantly increased computational effort.

To ensure that the results of smart Laplacian smoothing are independent of the node order, which is relevant if run in parallel, the following additional modification is also applied. Quality improving node updates are not conducted immediately, but stored separately until all nodes have been tested. After that, all relevant node updates are applied. This may lead to the generation of invalid elements. Therefore nodes with incident invalid elements are reset to their previous position. This is iterated until no invalid element remains in the mesh. Smoothing is terminated if the improvement of arithmetic mean of all element quality numbers drops below a given tolerance.

1.3.3 Global optimization

Whereas Laplacian smoothing variants are easy to implement and comparably fast, the resulting mesh quality generally cannot compete with that obtained by global optimization-based methods [19, 20]. These methods improve overall mesh quality by minimizing an objective function incorporating quality numbers of all mesh elements. Optimization is conducted by using standard methods, like the conjugate gradient approach, Newton's algorithm or active set algorithm.

As a representative of this class, the shape improvement wrapper of the mesh quality improvement toolkit Mesquite [137] is described in detail in the following. This state of the art global optimization-based approach will also serve as quality benchmark in subsequent examples. Smoothing is based on minimizing an objective function representing the arithmetic mean of the quality all mesh elements and minimization is accomplished by using a feasible Newton approach. In the following, the key components of the instruction queue of Mesquite used by the shape improvement wrapper are described in more detail mainly following the description given in [21]:

Quality metric: The quality metric used by the shape improvement wrapper is based on the inverse of the mean ratio quality criterion described in Section 3.1. For each element E of the mesh the mean ratio quality number $q(E)$ represents the deviation of an arbitrary valid element from its regular counterpart. In addition to the quality metric value, the analytic gradient and Hessian information is also provided, which is required by the optimization algorithm.

Objective function: While the quality metric provides a way to evaluate the properties of individual mesh entities, the objective function provides a way of combining those values into a single number for the entire mesh. In addition, the gradient and Hessian with respect to the vertex positions are determined. Following a global optimization based approach, in the shape improvement wrapper of Mesquite the arithmetic mean of the inverse of the mean ratio number $q(E_i)$ of all mesh elements E_i is used as an objective function, i.e.

$$\sum_{i=1}^{n_E} \frac{1}{q(E_i)} \;\rightarrow\; \min \tag{1.3}$$

is determined, where n_E denotes the number of all mesh elements.

Quality improver: This method takes as input the objective function and makes extensive use of the gradient and Hessian information provided therein. In addition a termination criterion has to be provided to stop iterating over the mesh. As vertex mover, the feasible Newton Algorithm is applied. Newton's Method minimizes a quadratic approximation of a non-linear objective function and is known to converge super-linearly near a non-singular local minimum. Therefore, mesh-optimization problems that are performed within the neighborhood of the minimum are favorable for this approach. The algorithm requires the objective function value, gradient, and Hessian. The latter is sparse for the objective function described before, which allows all vertex positions to be improved simultaneously. As a draw back, mesh configurations not in the vicinity of a local minimum may require a significant computational effort in order to be smoothed, as will be shown in by distorted mesh examples given in Chapter 5.

Termination criterion: This criterion controls the termination of the smoothing process. In case of the the shape improvement wrapper termination is based on the L^2-Norm of the gradient.

Additional components involved are the quality assessor, which evaluates the quality metric for all mesh elements and accumulates statistical information, and a mesh untangling preprocessing step in order to ensure mesh validity, which is a condition in the definition of the mean ratio quality criterion.

Chapter 2

Regularizing transformations for polygonal finite elements

Regularizing element transformations are the driving force behind the geometric element transformation method for finite element mesh smoothing, which is presented later in this work. These transformations represent simple geometric constructions, which, if applied iteratively to single mesh elements, lead to more regular, hence better quality elements. In this chapter, regularizing transformations are introduced and analyzed for planar polygonal elements. First, elementary proofs based on the methods of analysis are given for triangles in Section 2.1. Also providing a deep insight into the process of regularization, such an approach is not straightforward for polygons with an arbitrary number of nodes. Therefore, an additional study of the general case with the means of linear algebra is presented in the Sections 2.2 and 2.3. This will result in a full classification of the given regularizing transformations for planar polygons with respect to their construction parameters and the resulting limit polygons.

2.1 Iterative geometric triangle transformations

An initial triangle is transformed into a new triangle by erecting isosceles triangles on each of the sides of the initial one. The apices of the erected triangles are the vertices of the resulting triangle. Repeating this transformation leads to a sequence of triangles. The shapes of these triangles converge to a characteristic shape that does not depend on the choice of the initial triangle but on the shape of the erected isosceles triangles. In this section it is explicitly shown that this characteristic shape is approximated in every transformation step. Moreover, there is an upper bound for the speed of convergence.

2.1.1 Introduction and notation

A well-known theorem in geometry of triangles is the following: If equilateral triangles are erected externally on the sides of any triangle, their centers form an equilateral triangle. This theorem can be found in [138] and is often attributed to Napoleon Bonaparte, although it is questionable whether he knew enough geometry for this achievement, see [139].

There are several generalizations of this theorem. One is the following, which can be found in [139, 140]: If similar triangles PCB, CQA and BAR are erected externally on the sides of any triangle ABC, their circumcenters form a triangle similar to the three triangles. I. M. Yaglom proves in [141] the following generalization of the theorem above: On the sides of an arbitrary triangle ABC, exterior to it, isosceles triangles BCA_1, ACB_1, ABC_1 are erected with angles at the vertices A_1, B_1 and C_1, respectively equal to α, β and γ. If $\alpha+\beta+\gamma=2\pi$, then the angles of the triangle $A_1B_1C_1$ are equal to $\alpha/2$, $\beta/2$ and $\gamma/2$.

Note that the case $\alpha=\beta=\gamma=2\pi/3$ is just the same as taking the centers of equilateral triangles. It is easy to check that these two generalizations are equivalent by decomposing the triangle ABC in three isosceles triangles that have a common vertex in the circumcenter M of ABC (note that $\angle BMC = 2\cdot\angle BAC$).

Considering the formulation of Yaglom, the condition $\alpha+\beta+\gamma=2\pi$ is dropped and the transformation is repeated to obtain an infinite sequence of triangles. In doing so, the angles α, β and γ stay fixed. Two cases are analyzed. In the first case, all three angles are the same. In the second case, two angles coincide and the third equals π (hence, the corresponding erected triangle is degenerate). Equivalently to the second case, one may erect only two similar isosceles triangles and take the center of remaining side as the third vertex of the new triangle. It is shown that in both cases, the shape of the triangles converge to the shape of the triangle one would get if the condition $\alpha+\beta+\gamma=2\pi$ were satisfied. That is an equilateral triangle in the first case and a rectangular isosceles triangle in the second case.

In this section Δ_0 always denotes the initial triangle with vertices A_0, B_0 and C_0 (ordered counterclockwise). For $n\in\mathbb{N}$ the points A_{n+1}, B_{n+1} and C_{n+1} are defined recursively such that $A_nC_nB_{n+1}$, $B_nA_nC_{n+1}$, $C_nB_nA_{n+1}$ are isosceles triangles. The triangle with vertices A_n, B_n and C_n is denoted by Δ_n. The side-lengths of Δ_n are denoted by $x_n := \overline{B_nC_n}$, $y_n := \overline{C_nA_n}$ and $z_n := \overline{A_nB_n}$ and the angles are denoted by $\alpha_n := \angle B_nA_nC_n$, $\beta_n := \angle C_nB_nA_n$ and $\gamma_n := \angle A_nC_nB_n$.

The triangles do not have to be non-degenerate. The degenerate case where all three points are pairwise distinct but on a common line will be used to show that some given bounds are sharp. In the other degenerate cases there have to be vertices of the triangle that coincide and hence there has to be a side with length 0. Although such a side has no direction, this does not cause any problems for the iteration since in this case, the erected triangle always degenerates to a single point. Thus, the new vertex coincides with the two old ones. As a direct consequence, the degenerate case where all three points coincide is without any interest since such a triangle is invariant under all transformations introduced above. Therefore, the case $A_0 = B_0 = C_0$ is excluded from the following study.

2.1.2 Equilateral case

In this section, the triangles erected externally on the sides of Δ_n are similar to each other. More precisely, there is an angle $0 < \theta < \pi/2$ such that $\angle A_{n+1}B_nC_n = \angle B_nC_nA_{n+1} = \angle B_{n+1}C_nA_n = \angle C_nA_nB_{n+1} = \angle C_{n+1}A_nB_n = \angle A_nB_nC_{n+1} = \theta$. The so erected triangles are also known as Kiepert triangles [142].

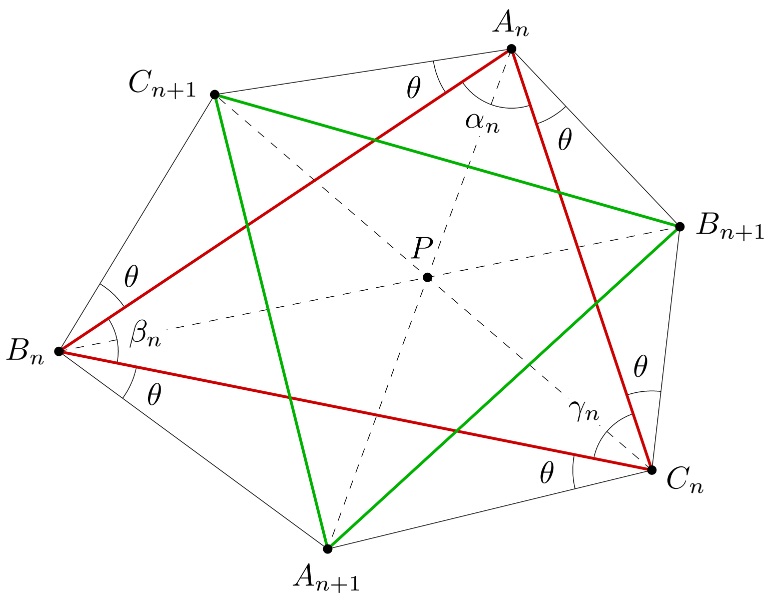


Figure 2.1: Transformation with similar isosceles externally erected triangles

Remark 2.1. The lines A_nA_{n+1}, B_nB_{n+1} and C_nC_{n+1} meet in a common point P, see [142] for a proof. Note that this point is inside the triangle Δ_n if and only if all angles of Δ_n are $< \pi - \theta$.

It is first shown how the side-lengths of the triangle Δ_{n+1} can be expressed in terms of the preceding triangle. Regarding the transformation there is no distinction between x_n, y_n and z_n. Therefore, the claims are usually stated for only one instance, but in the following the analogue statements will be used as well.

Lemma 2.1. *Let E_n denote the area of Δ_n. Then the following identities hold:*

$$x_{n+1}^2 = \frac{1}{2}\tan^2\theta \cdot (y_n^2 + z_n^2) + \frac{1}{4}(1 - \tan^2\theta)\cdot x_n^2 + 2\tan\theta \cdot E_n \tag{2.1}$$

$$x_{n+1}^2 - y_{n+1}^2 = \frac{1}{4}\left(1 - 3\tan^2\theta\right)(x_n^2 - y_n^2) \tag{2.2}$$

Proof. Note that $\overline{A_nC_{n+1}} = 1/(2\cos\theta)\cdot z_n$ and $\overline{A_nB_{n+1}} = 1/(2\cos\theta)\cdot y_n$. Hence, applying the law of cosines to the triangle $A_nC_{n+1}B_{n+1}$ yields

$$\begin{aligned} x_{n+1}^2 &= \frac{1}{4\cos^2\theta}\cdot y_n^2 + \frac{1}{4\cos^2\theta}\cdot z_n^2 - 2\cdot\frac{1}{4\cos^2\theta}\cdot y_nz_n\cos(2\theta+\alpha_n) \\ &= \frac{1}{4\cos^2\theta}\left(y_n^2+z_n^2\right) - \frac{\cos(2\theta+\alpha_n)}{2\cos^2\theta}\cdot y_nz_n\,. \end{aligned}$$

Using the addition theorems for the cosine and the sine one obtains $\cos(2\theta+\alpha_n) = (\cos^2\theta - \sin^2\theta)\cos\alpha_n - 2\sin\theta\cos\theta\sin\alpha_n$. Applying this and $E_n = (\sin\alpha_n/2)y_nz_n$ to the identity above leads to

$$\begin{aligned} x_{n+1}^2 &= \frac{1}{4\cos^2\theta}\left(y_n^2+z_n^2\right) - \frac{(1-\tan^2\theta)}{2}\cos\alpha_n\cdot y_nz_n + \tan\theta\sin\alpha_n\cdot y_nz_n \\ &= \frac{1}{4\cos^2\theta}\left(y_n^2+z_n^2\right) - \frac{1-\tan^2\theta}{2}\cdot\frac{y_n^2+z_n^2-x_n^2}{2} + 2\tan\theta\cdot E_n \\ &= \left(\frac{1-\cos^2\theta}{4\cos^2\theta} + \frac{1}{4}\tan^2\theta\right)\cdot(y_n^2+z_n^2) + \frac{1}{4}(1-\tan^2\theta)\cdot x_n^2 + 2\tan\theta\cdot E_n \\ &= \frac{1}{2}\tan^2\theta\cdot(y_n^2+z_n^2) + \frac{1}{4}(1-\tan^2\theta)\cdot x_n^2 + 2\tan\theta\cdot E_n \end{aligned}$$

The second equation is a direct consequence of the first one together with its analogue for y_{n+1}. □

Corollary 2.1. *Let $x_n \geq y_n$. Then*

$$\begin{aligned} x_{n+1} &\geq y_{n+1} \quad \textit{if } \theta \leq \frac{\pi}{6} \textit{ and} \\ x_{n+1} &\leq y_{n+1} \quad \textit{if } \theta \geq \frac{\pi}{6}\,, \end{aligned}$$

where equality on the left-hand sides hold if and only if $\theta = \pi/6$ or $x_n = y_n$.

Proof. This is a direct consequence of Equation (2.2) since $\tan^2(\pi/6) = 1/3$. □

Remark 2.2. It may be assumed that in the initial triangle x_0 is the greatest side and z_0 is the smallest. If $\theta < \pi/6$, the corollary above implies that x_n is the greatest side of Δ_n and z_n is the smallest one for every n. If $\theta > \pi/6$, things are different. For even n, it holds that $x_n \geq y_n \geq z_n$, whereas $x_n \leq y_n \leq z_n$ holds for odd n. In the special case $\theta = \pi/6$, the triangle Δ_n is equilateral for every $n > 0$. This observation is a motivation to consider the two subsequences $(\Delta_{2n})_{n\in\mathbb{N}}$ and $(\Delta_{2n+1})_{n\in\mathbb{N}}$ sometimes separately.

To study the behavior of corresponding side-lengths during the iteration, some estimates are stated. First, note the following two simple inequations, which will be used several times.

Lemma 2.2. *Let r and s be two positive real numbers. Then $r^2 + s^2 \geq \frac{1}{2}(r+s)^2$ and $r^2 + s^2 \geq 2rs$.*

Proof. Since $(r-s)^2$ is positive, one obtains $4r^2 + 4s^2 \geq (4r^2 - (r-s)^2) + (4s^2 - (s-r)^2) = (3r-s)(r+s) + (3s-r)(s+r) = (2r+2s)(r+s)$. Subtracting $2(r^2+s^2)$ on both sides implies the second claim. □

Lemma 2.3. *For the corresponding side-lengths of subsequent triangles, the following lower bounds hold:*

$$x_{n+1} \geq \frac{1}{2} x_n \tag{2.3}$$

$$x_{n+2}^2 \geq \frac{1}{16}\left(9\tan^4\theta + 1\right) \cdot x_n^2 \tag{2.4}$$

Moreover, if Δ_n is non-degenerate, both bounds are strict.

Proof. By Lemma 2.2 one obtains $y_n^2 + z_n^2 \geq \frac{1}{2}(y_n + z_n)^2 \geq \frac{1}{2}x_n^2$. Applying this to Equation (2.1) yields

$$x_{n+1}^2 \geq \frac{1}{4}\tan^2\theta \cdot x_n^2 + \frac{1}{4}(1 - \tan^2\theta) \cdot x_n^2 + 2\tan\theta \cdot E_n \geq \frac{1}{4}x_n^2 .$$

Now the first inequality follows directly. In the last step $2\tan\theta \cdot E_n$ is subtracted. Since $\tan\theta > 0$, equality cannot occur if Δ_n is non-degenerate.
Using the analogue of Equation (2.1) provides the following identity:

$$\begin{aligned} y_{n+1}^2 + z_{n+1}^2 &= \frac{\tan^2\theta}{2}\left(2x_n^2 + y_n^2 + z_n^2\right) + \frac{1-\tan^2\theta}{4}\left(y_n^2 + z_n^2\right) + 4\tan\theta \cdot E_n \\ &= \tan^2\theta \cdot x_n^2 + \frac{1+\tan^2\theta}{4}\left(y_n^2 + z_n^2\right) + 4\tan\theta \cdot E_n \end{aligned}$$

Applying this to the analogue of Equation (2.1) for x_{n+2}^2 results in

$$\begin{aligned}
x_{n+2}^2 &\geq \frac{1}{2}\tan^2\theta\left(\tan^2\theta\cdot x_n^2+\frac{1+\tan^2\theta}{4}\left(y_n^2+z_n^2\right)+4\tan\theta\cdot E_n\right)\\
&\quad+\frac{1-\tan^2\theta}{4}\left(\frac{\tan^2\theta}{2}\left(y_n^2+z_n^2\right)+\frac{1-\tan^2\theta}{4}\cdot x_n^2+2\tan\theta\cdot E_n\right)\\
&=\frac{\tan^2\theta}{4}\left(y_n^2+z_n^2\right)+\frac{8\tan^4\theta+(1-\tan^2\theta)^2}{16}\cdot x_n^2+\frac{3\tan^3\theta+\tan\theta}{2}\cdot E_n\\
&\geq\frac{\tan^2\theta}{8}\cdot x_n^2+\frac{9\tan^4\theta-2\tan^2\theta+1}{16}\cdot x_n^2\\
&=\frac{9}{16}\tan^4\theta\cdot x_n^2+\frac{1}{16}x_n^2
\end{aligned}$$

In the last but one step $(3\tan^3\theta+\tan\theta)E_n/2$ was subtracted. Since $3\tan^3\theta+\tan\theta>0$, equality cannot occur if Δ_n is non-degenerate. □

The first lower bound given in the previous lemma is sharp as can be checked by considering the degenerate case $\alpha_n=\pi$ and $y_n=z_n$. The second lower bound is sharp, too. This can be seen by considering again the degenerate case $\alpha_n=\pi$ and $y_n=z_n$ while θ tends to 0. As a consequence of these two estimates, the following theorems are stated.

Theorem 2.1. *For $n\geq 1$, the side-length of Δ_n are all >0.*

Proof. It may be assumed that Δ_{n-1} is degenerate since otherwise the claim follows directly from Equation (2.3).

Since Δ_0 has at least one side of length >0, Equation 2.3 implies that every triangle has at least one side of length >0. Suppose $x_n=0$. Then Equation (2.3) yields $x_{n-1}=0$. Thus, $E_{n-1}=0$ and consequently, $x_n^2=\tan^2\theta\cdot(y_{n-1}^2+z_{n-1}^2)/2$ by Equation (2.1). It holds that $y_{n-1}^2+z_{n-1}^2>0$ since otherwise all vertices of Δ_{n-1} would coincide. With $\tan^2\theta>0$ follows $x_n>0$, a contradiction. □

Theorem 2.2.

$$\begin{aligned}
&\textit{If } 0<\theta<\frac{\pi}{6}, \textit{ then} && \lim_{n\to\infty}x_n=0.\\
&\textit{If } \frac{\pi}{6}<\theta<\frac{\pi}{2}, \textit{ then} && \lim_{n\to\infty}x_n=\infty.\\
&\textit{If } \theta=\frac{\pi}{6}, \textit{ then} && x_n=x_1 \qquad \forall n>0.
\end{aligned}$$

Proof. First let $0<\theta<\pi/6$. Assume $x_0\geq y_0\geq z_0$. Then $x_n\geq y_n\geq z_n$ for every n by Remark 2.2. Hence, $\lim_{n\to\infty}x_n=0$ implies $\lim_{n\to\infty}y_n=\lim_{n\to\infty}z_n=0$. Thus, it suffices to prove the claim for the case $x_0\geq y_0\geq z_0$. Since $x_n\geq y_n\geq z_n$, it holds that $\alpha_n \geq \beta_n \geq \gamma_n$ due to the law of sines. This implies $\gamma_n \leq \pi/6$ and hence, $\sin\gamma_n \leq \sqrt{3}/2$. With $E_n = (\sin\gamma_n/2)x_n y_n$, Equation (2.1) implies

$$\begin{aligned} x_{n+1}^2 &\leq \frac{1}{2}\tan^2\theta \cdot \left(y_n^2 + z_n^2\right) + \frac{1}{4}(1-\tan^2\theta)\cdot x_n^2 + \frac{\sqrt{3}}{2}\tan\theta \cdot x_n y_n \\ &\leq (\frac{1}{4} + \frac{3}{4}\tan^2\theta + \frac{\sqrt{3}}{2}\tan\theta)\cdot x_n^2\,. \end{aligned}$$

Now $\theta < \pi/6$ implies $\tan\theta < \sqrt{3}/3$ and consequently $\frac{1}{4} + \frac{3}{4}\tan^2\theta + \frac{\sqrt{3}}{2}\tan\theta < 1$. The claim follows. Now let $\pi/6 < \theta < \pi/2$. Assume $x_0 \leq y_0 \leq z_0$. Then $x_{2n} \leq y_{2n} \leq z_{2n}$ for every n by Remark 2.2. Hence, $\lim_{n\to\infty} x_{2n} = \infty$ implies $\lim_{n\to\infty} y_{2n} = \lim_{n\to\infty} z_{2n} = \infty$. Thus, to prove $\lim_{n\to\infty} x_{2n} = \infty$ it suffices to consider the case $x_0 \leq y_0 \leq z_0$. Let n be even. Since $x_n \leq y_n \leq z_n$, one obtains $\alpha_n \leq \beta_n \leq \gamma_n$. This implies $\gamma_n \geq \frac{\pi}{6}$ and hence, $\sin\gamma_n \geq \sqrt{3}/2$. With $E_n = (\sin\gamma_n/2)x_n y_n$ one obtains by Equation (2.1)

$$\begin{aligned} z_{n+1}^2 &\geq \frac{1}{2}\tan^2\theta \cdot \left(x_n^2 + y_n^2\right) + \frac{1}{4}(1-\tan^2\theta)\cdot z_n^2 + \frac{\sqrt{3}}{2}\tan\theta \cdot x_n y_n \\ &\geq (\frac{1}{4} + \frac{3}{4}\tan^2\theta + \frac{\sqrt{3}}{2}\tan\theta)\cdot x_n^2\,. \end{aligned}$$

Now $\theta < \pi/6$ implies $\tan\theta < \sqrt{3}/3$ and consequently $\kappa := \frac{1}{4} + \frac{3}{4}\tan^2\theta + \frac{\sqrt{3}}{2}\tan\theta > 1$. Since $z_{n+1} \leq y_{n+1} \leq x_{n+1}$ by Remark 2.2, it can be concluded analogously that $x_{n+2}^2 \geq \kappa z_{n+1}^2$. Thus, $x_{n+2}^2 \geq \kappa^2 x_n^2$ whenever n is even and consequently $\lim_{n\to\infty} x_{2n} = \infty$. By analogous reasons one obtains $\lim_{n\to\infty} x_{2n+1} = \infty$ and the claim follows.

In the last case, one concludes by Corollary 2.1 that for every $n > 0$ the triangle Δ_n is equilateral. Furthermore, $\tan\theta = \sqrt{3}/3$. Thus, for $n > 0$, Equation (2.1) yields $x_{n+1}^2 = x_n^2$. □

Due to the unbounded growth of the triangles for $\theta > \pi/6$ and the fact that for $\theta < \pi/6$, the triangle sequence collapses to a single point, the only reasonable case to study seems to be the case where θ equals $\pi/6$. However, since only the shape of the triangle is of interest, the size of the triangle does not matter.

For further estimates another two simple inequations are stated:

Lemma 2.4. *Let r, s and t be positive real numbers all smaller than* 1. *Then* $1 - s^2 < r(1-t^2)$ *implies* $(1-s) < r(1-t)$.

Proof. It follows $1 - s^2 < 1 - t^2$ and therefore $s > t$. Thus, $(1-s) = (1-s^2)/(1+s) < r(1-t^2)/(1+s) < r(1-t^2)/(1+t) < r(1-t)$. □

The following general proposition applies separately to the two triangle sequences, i. e. the one with the even and the one with the odd indices. As in Remark 2.2 the inequation $x_0 \geq y_0$ is assumed for the following proposition.

Proposition 2.1. *Let $x_0 > y_0$. Then for every angle θ, there is a constant $0 \leq \kappa < 1$ such that*

$$0 \leq 1 - \frac{y_{n+2}}{x_{n+2}} \leq \kappa \left(1 - \frac{y_n}{x_n}\right),$$

where equality holds if and only if $\theta = \pi/6$ or $y_n = x_n$.

Proof. First note that for $n \geq 1$, Theorem 2.1 states $x_n > 0$. Furthermore, since $x_0 \geq y_0$ and the case $A_0 = B_0 = C_0$ is excluded, one obtains $x_0 > 0$.

Equation (2.2) provides

$$x_{n+2}^2 - y_{n+2}^2 = \left(\frac{1}{4} - \frac{3}{4}\tan^2\theta\right)^2 \cdot (x_n^2 - y_n^2)$$

and hence,

$$1 - \frac{y_{n+2}^2}{x_{n+2}^2} = \frac{(1 - 3\tan^2\theta)^2}{16} \cdot \frac{x_n^2}{x_{n+2}^2}\left(1 - \frac{y_n^2}{x_n^2}\right).$$

Thus, one may assume $x_n \neq y_n$ since otherwise one is done. Applying Inequation 2.4 yields

$$1 - \frac{y_{n+2}^2}{x_{n+2}^2} \leq \frac{(1 - 3\tan^2\theta)^2}{1 + 9\tan^4\theta}\left(1 - \frac{y_n^2}{x_n^2}\right).$$

Since $0 \leq (1-3\tan^2\theta)^2 < 1+9\tan^4\theta$ the claim follows for $\kappa := (1-3\tan^2\theta)^2/(1+9\tan^4\theta)$ by Lemma 2.4. Note that $\kappa = 0$ if and only if $\theta = \pi/6$. □

For $\theta \leq \pi/4$, the ratio of side-lengths tends to 1 in every step of iteration. In other the function $n \mapsto 1 - \min\{x_n, y_n\}/\max\{x_n, y_n\}$ is strictly decreasing as long as the values differ from 0. Note that for $\theta > \pi/6$, the role of the smaller side-length alternates.

Proposition 2.2. *Let $x_0 > y_0$ and $\theta \neq \pi/6$. Then there is a positive constant $\kappa < 1$ that only depends on θ such that the following holds:*

$$0 < 1 - \frac{y_{n+1}}{x_{n+1}} \leq \kappa\left(1 - \frac{y_n}{x_n}\right) \qquad \text{if } \theta < \frac{\pi}{6}$$

$$0 < 1 - \frac{x_{n+1}}{y_{n+1}} \leq \kappa\left(1 - \frac{y_n}{x_n}\right) \qquad \text{if } \frac{\pi}{6} < \theta \leq \frac{\pi}{4} \text{ and } n \text{ even}$$

$$0 < 1 - \frac{y_{n+1}}{x_{n+1}} \leq \kappa\left(1 - \frac{x_n}{y_n}\right) \qquad \text{if } \frac{\pi}{6} < \theta \leq \frac{\pi}{4} \text{ and } n \text{ odd}$$

Proof. First note that $x_n > 0$ and $y_n > 0$ for $n \geq 1$ by Theorem 2.1. Moreover, $x_0 > 0$ since $x_0 \geq y_0$ and the case $A_0 = B_0 = C_0$ is excluded.

Let $\omega := 1 - 3\tan^2\theta$. First assume $\theta < \pi/6$. Then $x_n > y_n$ by Corollary 2.1. Furthermore, $0 < \tan^2\theta < 1/3$ and therefore $0 < \omega < 1$. Inequation (2.3) implies $x_n \leq 2x_{n+1}$. Hence, dividing both sides of Equation (2.2) by x_{n+1}^2 provides

$$1 - \frac{y_{n+1}^2}{x_{n+1}^2} = \frac{\omega}{4} \cdot \frac{x_n^2}{x_{n+1}^2}\left(1 - \frac{y_n^2}{x_n^2}\right) \leq \omega\left(1 - \frac{y_n^2}{x_n^2}\right).$$

Now the claim follows from Lemma 2.4 by setting $\kappa := \omega$.

For $\theta > \frac{\pi}{6}$ Corollary 2.1 implies that $x_n > y_n$ if n is even and $y_n > x_n$ otherwise. The following is restricted to the case where n is even. The other case can be obtained by exchanging x_n with y_n, x_{n+1} with y_{n+1} and z_0 with z_1. Note that $\omega < 0$ for $\theta > \frac{\pi}{6}$.

Assuming $\frac{\pi}{6} < \theta < \frac{\pi}{4}$ implies $\tan^2\theta \leq 1$. Hence, Equation (2.1) yields $y_{n+1}^2 \geq \frac{1}{2}(x_n^2 + z_n^2)\tan^2\theta$. Furthermore, $\tan^2\theta \leq 1$ implies $-\omega \leq 2\tan^2\theta$. Dividing both sides of Equation (2.2) by $-y_{n+1}^2$ results in

$$\begin{aligned}
1 - \frac{x_{n+1}^2}{y_{n+1}^2} &= \frac{-\omega}{4} \cdot \frac{x_n^2}{y_{n+1}^2}\left(1 - \frac{y_n^2}{x_n^2}\right) \\
&\leq \frac{\tan^2\theta}{2} \cdot \frac{2x_n^2}{(x_n^2 + z_n^2)\tan^2\theta}\left(1 - \frac{y_n^2}{x_n^2}\right) \\
&= \frac{x_n^2}{x_n^2 + z_n^2}\left(1 - \frac{y_n^2}{x_n^2}\right).
\end{aligned}$$

Now set $\varepsilon := \min\{1, (\frac{z_0}{x_0})^2\}$. Then Proposition 2.1 together with induction implies $z_n^2 \geq \varepsilon x_n^2$. The claim follows for $\kappa := \frac{1}{1+\varepsilon}$ by using Lemma 2.4. □

Remark 2.3. For $\theta > \pi/4$ and $x_n > y_n$, it is possible that $1 - x_{n+1}/y_{n+1}$ exceeds $1 - y_n/x_n$, especially if θ is close to $\pi/2$. However, in this situation there is another observation one can make: While θ tends to $\pi/2$, the angle α_{2n} tends to α_0 for every $n \in \mathbb{N}$. Analogously, $\lim_{\theta\to\pi/2}\beta_{2n} = \beta_0$ and $\lim_{\theta\to\pi/2}\gamma_{2n} = \gamma_0$. Hence, the shape of Δ_2 tends to the shape of Δ_0. On the other hand $\lim_{\theta\to\pi/2} x_1 = \lim_{\theta\to\pi/2} y_1 = \lim_{\theta\to\pi/2} z_1 = \infty$ as long as Δ_0 is non-degenerate. Thus, one cannot speak of a limit triangle.

To avoid the enormous growth of the triangles, one can dilate each transformed triangle after the iteration with the reciprocal of the largest side-length. Equivalently, one can apply this dilation before the step of iteration. By doing so, Δ_1 converges pointwise while θ tends to $\pi/2$ as long as one takes a fixed center for the dilations. Thus, one obtains a limit triangle called Δ_1'. Repeating this process leads to two sequences of pairwise similar triangle Δ_{2n}' and Δ_{2n+1}'. The triangles Δ_0 and Δ_1' do not have to be similar.

Clearly, while θ tends to $\pi/2$, the factor of the dilation applied to Δ_0 tends to 0. Thus, the vertices A'_1, B'_1 and C'_1 of Δ'_1 lie on the lines through the dilation center that are perpendicular to one of the sides of Δ_0. Moreover, the proportions of the distances from the dilation center to A'_1, B'_1, C'_1 match the proportions of z_1, y_1, x_1. Hence, a triangle similar to Δ'_1 can be obtained by taking three concurrent rays r_0, r_1, r_2 such that r_0 and r_1 span the angle $2\alpha_0$, r_0 and r_2 span the angle $2\beta_0$ and r_1 and r_2 span the angle $2\gamma_0$. Taking the points on r_0, r_1, r_2 at distance z_0, y_0, x_0, respectively, to the intersection of the three rays provides a triangle similar to Δ'_1.

Since Δ'_2 is similar to Δ_0 again, the shape of Δ'_1 can be seen as some kind of dual shape to the shape of Δ_0.

The following theorem represents the main result of this section, namely, regarding only the shape of the triangles, Δ_n tends to an equilateral triangle for $n \to \infty$.

Theorem 2.3. *For every initial triangle Δ_0 and every angle $0 < \theta < \frac{\pi}{2}$, the following two limits hold:*

$$\lim_{n\to\infty} \frac{x_n}{y_n} = 1$$

$$\lim_{n\to\infty} \alpha_n = \frac{\pi}{3}$$

Proof. The first limit is a direct consequence of Proposition 2.1. The second limit follows by the first together with the law of sines. □

This section is concluded by two theorems concerning the position and the orientation of the triangles.

Theorem 2.4. *For every $n > 0$, the centroid of Δ_n coincides with the centroid of Δ_0.*

Proof. Consider the Euclidean plane as vector space. For $n \in \mathbb{N}$, let a_n, b_n and c_n be the vectors representing the points A_n, B_n and C_n, respectively. Let δ be the linear transformation that rotates the Euclidean plane by $\pi/2$. Then $a_{n+1} = \frac{1}{2}(b_n + c_n) + (\frac{1}{2}\tan\theta(b_n - c_n))^\delta$. Since δ is linear, this implies $a_{n+1} + b_{n+1} + c_{n+1} = a_n + b_n + c_n$. Thus, the centroid of Δ_n, defined as $\frac{1}{3}(a_n + b_n + c_n)$ coincides with the one of Δ_{n+1}. The claim follows by induction. □

Theorem 2.5. *For every $n > 0$, the triangle Δ_n is non-degenerate and counter-clockwise oriented.*

Proof. Assume that Δ_n is non-degenerate and counterclockwise oriented. By symmetric reasons one may assume $x_n \geq y_n \geq z_n$. Hence, $\alpha_n \geq \beta_n \geq \gamma_n$ by the law of sines and therefore $\beta_n \leq \pi/2$ and $\gamma_n \leq \pi/2$.

Let A'_n, B'_n and C'_n denote the centers of B_nC_n, C_nA_n and A_nB_n, respectively. Since the triangle $A'_nC_nB'_n$ is similar to Δ_n, one obtains $\angle C_nA'_nB'_n = \beta_n$. On the other hand, let l be the perpendicular bisector of the side C_nA_n, i. e. the line through B'_n and B_{n+1}. Since $x_n \geq z_n$, the line l intersects x_n in a point S. It follows $\angle C_nSB_{n+1} = \pi/2 - \gamma_n$. Thus, $\min\{\beta_n, \pi/2 - \gamma_n\} \leq \angle C_nA'_nB_{n+1} \leq \max\{\beta_n, \pi/2 - \gamma_n\}$ and therefore $0 < \angle C_nA'_nB_{n+1} < \pi/2$. Analogously, $0 < \angle C_{n+1}A'_nB_n < \pi/2$. This implies that the angles $\angle C_{n+1}A'_nA_{n+1}$ and $\angle A_{n+1}A'_nB_{n+1}$ are greater than $\pi/2$ and smaller than π and consequently, $\angle B_{n+1}A'_nC_{n+1} < \pi$. It follows that A'_n is inside the triangle Δ_{n+1} and the Δ_{n+1} is counterclockwise oriented since $\angle A_{n+1}A'_nB_{n+1} < \pi$. Now the claim follows by induction. $\square$

2.1.3 Rectangular isosceles case

As in the previous section, the points B_{n+1} and C_{n+1} are the apices of similar isosceles triangles erected to the outside of Δ_n over the edges y_n and z_n, respectively. More precisely, there is an angle $0 < \theta < \pi/2$ such that $\angle B_{n+1}C_nA_n = \angle C_nA_nB_{n+1} = \angle C_{n+1}A_nB_n = \angle A_nB_nC_{n+1} = \theta$. In contrast to the previous section, the point A_{n+1} is the center of B_nC_n (or, equivalently, the apex of a degenerate isosceles triangle with angle π).

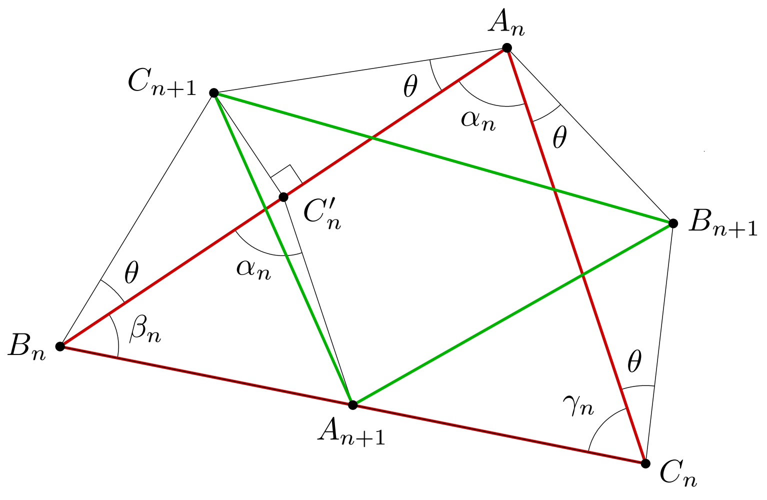


Figure 2.2: Transformation with two similar isosceles externally erected triangles and one midpoint of a side

Again, equations for the side-lengths of the triangle Δ_{n+1} in terms of Δ_n are given first. Regarding the transformation there is no distinction between y_n and z_n except for the orientation. Therefore the claims are usually stated for only one instance, but the analogue statements will be used as well in the following.

Lemma 2.5. *Let E_n be the area of Δ_n. Then the following identities hold:*

$$x_{n+1}^2 = \frac{1}{2}\tan^2\theta \cdot (y_n^2 + z_n^2) + \frac{1}{4}(1-\tan^2\theta)\cdot x_n^2 + 2\tan\theta \cdot E_n \tag{2.1}$$

$$y_{n+1}^2 = \frac{1}{4}\cdot y_n^2 + \frac{1}{4}\tan^2\theta \cdot z_n^2 + \tan\theta\cdot E_n \tag{2.5}$$

Proof. The first equation is obtain in precisely the same way as in the proof of Lemma 2.1.

Let C_n' be the center of A_nB_n. Then $\overline{C_{n+1}C_n'} = \frac{1}{2}\tan\theta\cdot z_n$ and $\overline{A_{n+1}C_n'} = \frac{1}{2}\cdot y_n$. Applying the law of cosines to the triangle $A_{n+1}C_n'C_{n+1}$ (possibly degenerate for $\alpha_n = \frac{\pi}{2}$ and oriented clockwise for $\alpha_n > \frac{\pi}{2}$) yields

$$y_{n+1}^2 = \frac{1}{4}\cdot y_n^2 + \frac{1}{4}\tan^2\theta\cdot z_n^2 - 2\cdot\frac{1}{4}\tan\theta\cdot y_nz_n\cos(\frac{\pi}{2}+\beta_n+\gamma_n).$$

With $\cos(\frac{\pi}{2}+\beta_n+\gamma_n) = \sin(-\beta_n-\gamma_n) = \sin(\alpha_n - \frac{\pi}{2}) = -\sin(\alpha_n)$ and $E_n = \frac{1}{2}\sin\alpha_n\cdot y_nz_n$, the second identity follows. □

The following identities are immediate consequences of the previous lemma.

$$y_{n+1}^2 - z_{n+1}^2 = \frac{1}{4}\left(1-\tan^2\theta\right)\cdot\left(y_n^2 - z_n^2\right) \tag{2.6}$$

$$y_{n+1}^2 + z_{n+1}^2 = \frac{1}{4}\left(1+\tan^2\theta\right)\cdot\left(y_n^2 + z_n^2\right) + 2\tan\theta\cdot E_n \tag{2.7}$$

$$x_{n+1}^2 - y_{n+1}^2 - z_{n+1}^2 = \frac{1}{4}\left(1-\tan^2\theta\right)\cdot\left(x_n^2 - y_n^2 + z_n^2\right) \tag{2.8}$$

Corollary 2.2. *Let $y_n \geq z_n$. Then*

$$\begin{array}{lcll} y_{n+1} & \geq & z_{n+1} & \textit{if } \theta \leq \frac{\pi}{4} \textit{ and} \\ y_{n+1} & \leq & z_{n+1} & \textit{if } \theta \geq \frac{\pi}{4}, \end{array}$$

where equality on the left-hand side holds if and only if $\theta = \frac{\pi}{4}$ or $y_n = z_n$.

Proof. This is a direct consequence of Equation (2.6) since $\tan(\pi/4) = 1$. □

Since A_{n+1} is obtained in a different way than B_{n+1} and C_{n+1}, there is no corresponding condition that involves x_n and x_{n+1}. The next step is to give lower bounds for the side-lengths after two steps of iteration.

Lemma 2.6. *For the corresponding side-lengths of subsequent triangles, the following lower bounds hold:*

$$x_{n+2}^2 \geq \frac{1}{16}\left(1+\tan^4\theta\right)x_n^2 \tag{2.9}$$

$$y_{n+2}^2 \geq \frac{1}{16}\left(1+\tan^4\theta\right)y_n^2 \tag{2.10}$$

Proof. Applying Equation (2.7) and Equation (2.1) to the analogue of Equation (2.1) for x_{n+2}^2 yields

$$\begin{aligned} x_{n+2}^2 &\geq \frac{\tan^2\theta}{2}\left(\frac{1+\tan^2\theta}{4}\cdot\left(y_n^2+z_n^2\right)+2\tan\theta\cdot E_n\right) \\ &\quad+\frac{1-\tan^2\theta}{4}\left(\frac{\tan^2\theta}{2}\cdot\left(y_n^2+z_n^2\right)+\frac{1-\tan^2\theta}{4}\cdot x_n^2+2\tan\theta\cdot E_n\right) \\ &= \frac{\tan^2\theta}{4}\cdot\left(y_n^2+z_n^2\right)+\frac{(1-\tan^2\theta)^2}{16}\cdot x_n^2+\frac{\tan\theta+\tan^3\theta}{2}\cdot E_n \end{aligned}$$

Now Lemma 2.2 implies $y_n^2+z_n^2\geq(y_n+z_n)^2/2\geq x_n^2/2$ and thus,

$$\begin{aligned} x_{n+2}^2 &\geq \frac{\tan^2\theta}{8}\cdot x_n^2+\frac{1-2\tan^2\theta+\tan^4\theta}{16}\cdot x_n^2 \\ &= \frac{1}{16}x_n^2+\frac{1}{16}\tan^4\theta\cdot x_n^2\,. \end{aligned}$$

Using Equation (2.5) repeatedly yields

$$\begin{aligned} y_{n+2}^2 &\geq \frac{1}{4}y_{n+1}^2+\frac{1}{4}\tan^2\theta\cdot z_{n+1}^2 \\ &\geq \frac{1}{16}y_n^2+\frac{1}{8}\tan^2\theta\cdot z_{n+1}^2+\frac{1}{16}\tan^4\theta\cdot y_n^2 \\ &\geq \frac{1}{16}y_n^2+\frac{1}{16}\tan^4\theta\cdot y_n^2\,. \end{aligned}$$

□

As in the previous section, the lemma above motivates to consider the sequence of triangle as two separated sequences.

Theorem 2.6. *For $n\geq 1$, the side-length of Δ_n are all >0.*

Proof. The side-length x_n does not depend on A_n and hence, for a given triangle Δ_{n-1} and a given angle θ, the side-length x_n is just the same as in the previous section. Thus, $x_n>0$ by Theorem 2.1.

For y_n and z_n, one may assume that Δ_{n-1} is degenerate since otherwise the claim follows directly from Equation (2.5).

Since Δ_0 has at least two sides of length >0, Equation (2.5) implies $y_1>0$ and analogously, $z_1>0$. Now the claim follows by induction using Equation (2.5). □

In the following the ratio of corresponding side-lengths are studied.

Proposition 2.3. *Let $y_0 \geq z_0$. Then for every angle $0 < \theta < \pi/2$, there is a constant $0 \leq \kappa < 1$ such that*

$$0 \leq 1 - \frac{z_{n+2}}{y_{n+2}} \leq \kappa \left(1 - \frac{z_n}{y_n}\right),$$

where equality holds if and only $\theta = \pi/4$ or $y_n = z_n$.

Proof. Note that $y_n > 0$ for $n \geq 1$ by Theorem 2.6. Moreover, $y_0 > 0$ since otherwise $z_0 = y_0 = 0$ and hence, $A_0 = B_0 = C_0$.

Equation (2.6) provides

$$y_{n+2}^2 - z_{n+2}^2 = \left(\frac{1}{4} - \frac{1}{4}\tan^2\theta\right)^2 \cdot (y_n^2 - z_n^2)$$

and hence,
$$1 - \frac{z_{n+2}^2}{y_{n+2}^2} = \frac{(1-\tan^2\theta)^2}{16} \cdot \frac{y_n^2}{y_{n+2}^2}\left(1 - \frac{z_n^2}{y_n^2}\right)$$

One may assume $y_n \neq z_n$ since otherwise one is done. Applying Inequation 2.10 yields

$$1 - \frac{z_{n+2}^2}{y_{n+2}^2} \leq \frac{(1-\tan^2\theta)^2}{1+\tan^4\theta}\left(1 - \frac{z_n^2}{y_n^2}\right).$$

Since $0 \leq (1-\tan^2\theta)^2 < 1 + \tan^4\theta$ the claim follows for $\kappa := (1-\tan^2\theta)^2/(1+\tan^4\theta)$ by Lemma 2.4. Note that $\kappa = 0$ if and only if $\theta = \pi/4$. □

Proposition 2.4. *For every angle θ, there is a constant $0 \leq \kappa < 1$ such that*

$$0 \leq \left|1 - \frac{y_{n+2}^2 + z_{n+2}^2}{x_{n+2}^2}\right| \leq \kappa \cdot \left|1 - \frac{y_n^2 + z_n^2}{x_n^2}\right|,$$

where equality holds if and only $\theta = \pi/4$ or $x_n^2 = y_n^2 + z_n^2$.

Proof. By Theorem 2.6, the only possibility where one of the fractions is not defined is the case $n = 0$ and $x_0 = 0$. In this case, the term at the right hand side can be understood as a term of infinite value, which makes the claim obviously true for this case.

Equation (2.8) provides

$$x_{n+2}^2 - y_{n+2}^2 - z_{n+2}^2 = \left(\frac{1}{4} - \frac{1}{4}\tan^2\theta\right)^2 \cdot (x_n^2 - y_n^2 - z_n^2)$$

and hence,
$$1 - \frac{y_{n+2}^2 + z_{n+2}^2}{x_{n+2}^2} = \frac{(1-\tan^2\theta)^2}{16} \cdot \frac{x_n^2}{x_{n+2}^2}\left(1 - \frac{y_n^2 + z_n^2}{x_n^2}\right).$$

One may assume $x_n^2 \neq y_n^2 + z_n^2$ since otherwise one is done. Applying Inequation 2.9 yields

$$\left|1 - \frac{y_{n+2}^2 + z_{n+2}^2}{x_{n+2}^2}\right| \leq \frac{\left(1 - \tan^2\theta\right)^2}{1 + \tan^4\theta} \cdot \left|1 - \frac{y_n^2 + z_n^2}{x_n^2}\right| .$$

For $\kappa := (1 - \tan^2\theta)^2/(1 + \tan^4\theta)$, the claim follows since $0 \leq (1 - \tan^2\theta)^2 < 1 + \tan^4\theta$. Note that $\kappa = 0$ if and only if $\theta = \pi/4$. □

Now, the main result can be stated. Regarding only the shape of the triangles, Δ_n tends to a rectangular isosceles triangle for $n \to \infty$.

Theorem 2.7. *For every initial triangle Δ_0 and every angle $0 < \theta < \frac{\pi}{2}$, the following limits hold:*

$$\lim_{n\to\infty} \frac{y_n^2 + z_n^2}{x_n^2} = 1 \qquad \lim_{n\to\infty} \alpha_n = \frac{\pi}{2}$$
$$\lim_{n\to\infty} \frac{y_n}{z_n} = 1 \qquad \lim_{n\to\infty} \beta_n = \frac{\pi}{4}$$

Proof. The limits on the left-hand side are immediate consequences of Propositions 2.3 and 2.4. By the law of cosines it follows that $\cos\alpha_n = \frac{1}{2}(y_n^2 + z_n^2 - x_n^2)/(y_n z_n)$. Now the limits on the left-hand side imply $\lim_{n\to\infty} \cos\alpha_n = 0$ and hence, $\lim_{n\to\infty} \alpha_n = \pi/2$. The last limit follows from $\lim_{n\to\infty} y_n/z_n = 1$ together with the law of sines. □

As in the previous section, the size of the triangles becomes stable for only one specific choice of θ. For every greater angle, the triangles grow unbounded and for every smaller angle, the triangles collapse to a single point.

Theorem 2.8.

If $0 < \theta < \frac{\pi}{4}$, *then* $\quad \lim_{n\to\infty} x_n = \lim_{n\to\infty} y_n = 0.$

If $\frac{\pi}{4} < \theta < \frac{\pi}{2}$, *then* $\quad \lim_{n\to\infty} x_n = \lim_{n\to\infty} y_n = \infty.$

If $\theta = \frac{\pi}{4}$, *then* $\quad x_n = x_1 = \sqrt{2} \cdot y_1 = \sqrt{2} \cdot y_n \qquad \forall n > 0.$

Proof. First let $0 < \theta < \pi/4$. Then $\tan\theta < 1$ and hence, Equation 2.7 implies

$$\begin{aligned} y_{n+1}^2 + z_{n+1}^2 &< \frac{1}{2}\left(y_n^2 + z_n^2\right) + \tan\theta \sin\alpha_n \cdot y_n z_n \\ &\leq \frac{1}{2}\left(y_n^2 + z_n^2\right) + \tan\theta \cdot y_n z_n . \end{aligned}$$

With Lemma 2.2 one concludes $y_{n+1}^2 + z_{n+1}^2 < \frac{1}{2}(1 + \tan\theta)(y_n^2 + z_n^2)$. Since $\frac{1}{2}(1 + \tan\theta) < 1$, this implies $\lim_{n\to\infty}(y_n^2 + z_n^2) = 0$ and hence, $\lim_{n\to\infty} y_n = \lim_{n\to\infty} z_n = 0$. The claim follows.

For $\frac{\pi}{4} < \theta$ it follows that $\tan\theta > 1$. Let $\varepsilon > 0$ such that $\tan\theta > (1+\varepsilon)^2$. By Theorem 2.7 there are natural numbers n_z and n_α such that $(1+\varepsilon/2)z_n > y_n$ for every $n > n_z$ and $(1+\varepsilon/2)\sin\alpha_n > 1$ for every $n > n_\alpha$. Set $n_0 := \max\{n_z, n_\alpha\}$. Then for every $n > n_0$, Equation (2.5) implies

$$\begin{aligned} y_{n+1}^2 &> \frac{1}{4}\cdot y_n^2 + \frac{1}{4}(1+\varepsilon)^4 \cdot z_n^2 + (1+\varepsilon)^2 \cdot \frac{1}{2}\sin\alpha_n y_n z_n \\ &> \frac{1}{4}\cdot y_n^2 + \frac{1}{4}(1+\varepsilon)^2 \cdot y_n^2 + \frac{1}{2}\cdot y_n^2 \\ &> (1+\frac{\varepsilon}{2})y_n^2 \end{aligned}$$

Thus, $\lim_{n\to\infty} y_n = \infty$. Now $\lim_{n\to\infty} x_n = \infty$ follows from Theorem 2.7.

For the last case, Corollary 2.2 implies $y_n = z_n$ for every $n > 0$. The rest follows from Equation (2.8). □

This section is concluded with a statement concerning the orientation of the triangles Δ_n.

Theorem 2.9. *For every $n > 0$, the triangle Δ_n is non-degenerate and counterclockwise oriented.*

Proof. Assume that Δ_n is non-degenerate and counterclockwise oriented. By symmetric reasons it is also assumed that $y_n \geq z_n$. Let B'_n and C'_n denote the centers of C_nA_n and A_nB_n, respectively.

First the case $x_n \geq y_n$ is considered. Then $\alpha_n \geq \beta_n$ and $\alpha_n \geq \gamma_n$ by the law of sines and therefore $\beta_n \leq \pi/2$ and $\gamma_n \leq \pi/2$. Since the triangle $A_{n+1}C_nB'_n$ is similar to Δ_n, one obtains $\angle C_nA_{n+1}B'_n = \beta_n$. On the other hand, let l be the perpendicular bisector of the side y_n, i.e. the line through B'_n and B_{n+1}. Since $x_n \geq z_n$, the line l intersects B_nC_n in a point S. One obtains $\angle C_nSB_{n+1} = \pi/2 - \gamma_n$. Thus, $\min\{\beta_n, \pi/2 - \gamma_n\} \leq \angle C_nA_{n+1}B_{n+1} \leq \max\{\beta_n, \pi/2 - \gamma_n\}$ and therefore $0 < \angle C_nA_{n+1}B_{n+1} < \pi/2$. Analogously, $0 < \angle C_{n+1}A_{n+1}B_n < \pi/2$. This implies $\angle C_{n+1}A_{n+1}B_n < \pi$ and the claim holds for Δ_{n+1}.

Now assume $x_n < y_n$. Then one obtains analogously to the above $0 < \angle A_nB'_nC_{n+1} < \pi/2$ and $\angle A_{n+1}B'_nC_n = \alpha_n < \pi/2$. Thus, both angles $\angle A_{n+1}B'_nB_{n+1}$ and $\angle B_{n+1}B'_nC_{n+1}$ are greater than $\pi/2$ and smaller than π and consequently, it holds that $\angle C_{n+1}B'_nA_{n+1} < \pi$. It follows that B'_n is inside the triangle Δ_{n+1} and that the claim holds for Δ_{n+1}.

The proof is completed by applying induction. □

Remark 2.4. The missing case where only on one side, say B_nC_n, an isosceles triangle is erected and of the other two sides the center is taken is not very interesting. Following I. M .Yaglom [141], the distinguished angle for the isosceles triangle would be $\angle C_nA_{n+1}B_n = 0$ and therefore $\theta = \angle A_{n+1}B_nC_n = \angle B_nC_nA_{n+1} = \pi$, which is not possible. One gains the idea that the shape to which the triangles converge should be degenerate. Moreover, since every possible choice θ is smaller than π, the triangles should collapse to a single point. These claims are easy to prove: One can see immediately $x_{n+1} = x_n/2$. Furthermore, $y_{n+1} < y_n/2 + \tan\theta \cdot x_n/2$ and $z_{n+1} < z_n/2 + \tan\theta \cdot x_n/2$. Thus, the ratios x_n/y_n and y_n/z_n tend to 0 while y_n/z_n converges to 1. After reaching the point $\tan\theta \cdot x_n < y_n$, one obtains additionally $y_{n+1} < y_n$. The analogue holds for z_n. Hence, $\lim_{n\to\infty} x_n = \lim_{n\to\infty} y_n = \lim_{n\to\infty} z_n = 0$. Consequentially, for triangles, the next task would be to consider three different angles θ_x, θ_y and θ_z for the isosceles triangles that are erected on the sides of Δ_n.

2.2 Iterative geometric polygon transformations

The transformation of a triangular element based on erecting isosceles triangles on its side has been analyzed in the previous section. However, this type of elementary analysis is not straightforward for polygons with an arbitrary number of nodes. Therefore, in this section the analysis of the transformation for arbitrary polygons is based on techniques of linear algebra.

2.2.1 Motivation

In accordance to the triangular case, polygons with an arbitrary number of nodes are transformed by erecting similar isosceles triangles on its sides. The transformed polygon consists of the apices of these similar triangles. The sequence resulting from iteratively applying the same transformation will be analyzed with respect to the base angle $\theta \in (0, \pi/2)$ of the isosceles triangles. This is illustrated in Fig. 2.3 for random initial polygons with $n = 10$ vertices (upper) and $n = 11$ vertices (lower).

It can be seen that there is a change in the geometry of limit figures of the sequences of scaled polygons at each characteristic angle $\theta_k = \pi(2k+1)/(2n)$, $k \in \{0, \ldots, \lfloor n/2 \rfloor - 1\}$. This will be shown in the following, leading to a full classification of limit polygons of isosceles triangles-based polygon transformations. Whereas the polygons for θ within an interval bounded by characteristic angles are regular polygons or equilateral stars with possibly multiple vertices (positioned above the center of the according interval in Fig. 2.3), the polygons for $\theta = \theta_k$, $k > 0$, are linear combinations of the neighboring limit polygons. Fur-

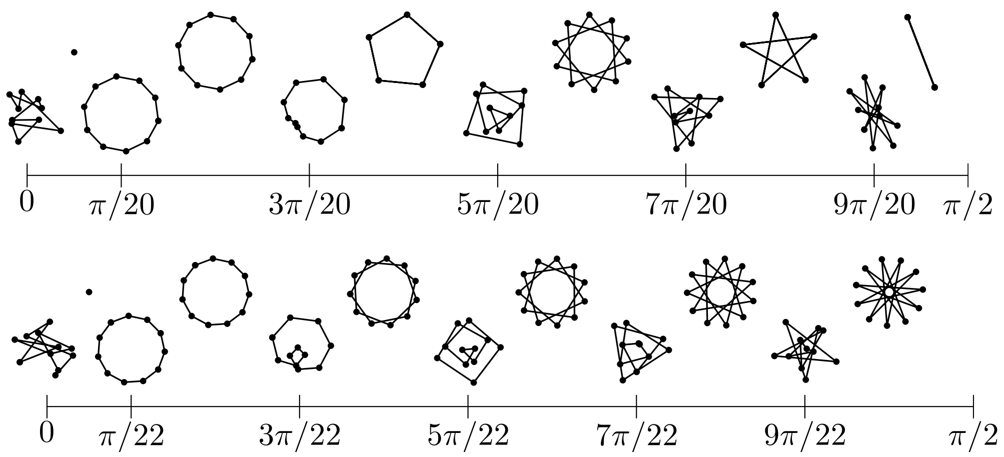


Figure 2.3: Initial random polygons (plotted above $\theta = 0$) and resulting limit polygons for $n = 10$ (upper) and $n = 11$ (lower) depending on the base angle $\theta \in (0, \pi/2)$

thermore, the unscaled polygons degenerate to their common centroid, become bounded regular n-gons or grow infinitely in the case of θ being smaller, equal or larger than $\theta_0 = \pi/(2n)$.

In the following, a representation of the transformation using complex numbers and circulant matrices is given. By analyzing the transformation using the discrete Fourier transform, the characteristic angles are derived and it will be shown that the obtained shapes are linear combinations of specific eigenpolygons also known as fundamental polygons [143, 144].

2.2.2 Transformation of a polygon

First, a definition of the transformation of a polygon using complex numbers will be given. Let $z^{(0)} \in \mathbb{C}^n$ denote a plane counterclockwise oriented polygon with $n \geq 3$ vertices $z_k^{(0)} := (z^{(0)})_k$, $k \in \{0, \dots, n-1\}$, and sides of length > 0, which may possibly intersect each other. Over each side $z_k^{(0)} z_{(k+1) \bmod n}^{(0)}$ of the polygon an outward directed isosceles triangle $z_k^{(0)} z_k^{(1)} z_{(k+1) \bmod n}^{(0)}$ with base angle $\theta \in (0, \pi/2)$ is constructed, as depicted in Fig. 2.4.

The apices are given by

$$z_k^{(1)} := w z_k^{(0)} + \overline{w} z_{(k+1) \bmod n}^{(0)} \quad \text{with} \quad w := \frac{1}{2}(1 + \mathrm{i} \tan \theta)$$

and $\overline{w}$ denoting the complex conjugate of w. The vertices $z_k^{(1)}$ in term define a new polygon $z^{(1)}$, which can be obtained by using the following matrix formulation.

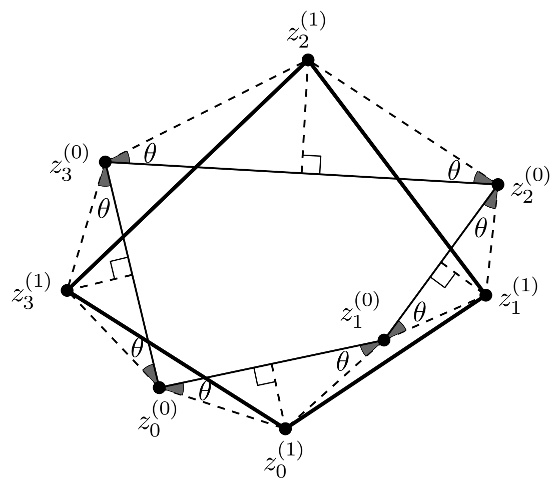


Figure 2.4: Transformation of a polygon in the case of $n = 4$ using $\theta = \pi/6$

Definition 2.1. *The linear transformation of a polygon $z^{(\ell)} \in \mathbb{C}^n$ into a new polygon $z^{(\ell+1)} \in \mathbb{C}^n$ is given by*

$$z^{(\ell+1)} = \begin{pmatrix} z_0^{(\ell+1)} \\ z_1^{(\ell+1)} \\ \vdots \\ z_{n-2}^{(\ell+1)} \\ z_{n-1}^{(\ell+1)} \end{pmatrix} := \underbrace{\begin{pmatrix} w & \overline{w} & & & \\ & w & \overline{w} & & \\ & & \ddots & \ddots & \\ & & & w & \overline{w} \\ \overline{w} & & & & w \end{pmatrix}}_{=:M} \begin{pmatrix} z_0^{(\ell)} \\ z_1^{(\ell)} \\ \vdots \\ z_{n-2}^{(\ell)} \\ z_{n-1}^{(\ell)} \end{pmatrix} = Mz^{(\ell)}\,, \tag{2.11}$$

where $w = (1 + \mathrm{i}\tan\theta)/2$.

Due to $w+\overline{w} = 1$, the sum of each row and column of M is one, thus leading to the familiar result that the transformation preserves the centroid of the polygon. That is, $\frac{1}{n}\sum_{k=0}^{n-1} z_k^{(\ell+1)} = \frac{1}{n}\sum_{k=0}^{n-1} z_k^{(\ell)}$, which can easily be shown by rearranging the sum and collecting the coefficients of $z_k^{(\ell)}$.

Iteratively applying the transformation given by definition 2.1 to an initial polygon $z^{(0)}$ leads to a sequence of concentric polygons $z^{(\ell)}$, $\ell \in \mathbb{N}$, with $z^{(\ell)} = M^\ell z^{(0)}$. This is depicted in Fig. 2.5 for different transformation angles θ and iteration numbers ℓ. In this, the same initial 10-gon $z^{(0)}$ as depicted on the upper left of Fig. 2.3 has been used. Additionally, the polygon size has been normalized after each iteration step, to avoid the rapid increase of size in the case of larger θ values. In order to determine the convergence behavior of such sequences, the matrix M will be analyzed in the following.

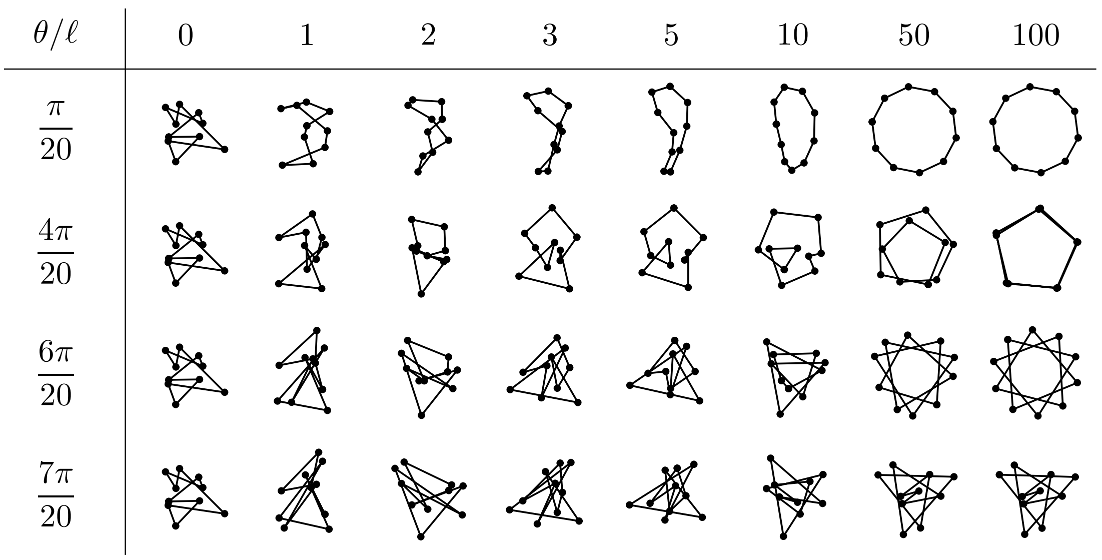


Figure 2.5: Sequences of polygons $z^{(\ell)} = M^\ell z^{(0)}$ obtained by iteratively applying the transformation using different transformation angles θ

2.2.3 Circulant matrices and eigenpolygon decompositions

Since each row of M is a cyclic shift of its preceeding row, the theory of circulant matrices can be applied. For the convenience of the reader, the relevant results will be given briefly [145, 146].

The square matrix $A \in \mathbb{C}^{n\times n}$ is called circulant, if it is of the form

$$A := \begin{pmatrix} a_0 & a_1 & \dots & a_{n-1} \\ a_{n-1} & a_0 & \dots & a_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \dots & a_0 \end{pmatrix} .$$

It is fully defined by its first row vector $a := (a_0, \dots, a_{n-1})$ with zero-based element indices. With $r := \exp(2\pi\mathrm{i}/n)$ denoting the n-th root of unity it holds that A is diagonalized by the unitary discrete Fourier matrix

$$F := \frac{1}{\sqrt{n}} \begin{pmatrix} r^{0\cdot 0} & \dots & r^{0\cdot(n-1)} \\ \vdots & \ddots & \vdots \\ r^{(n-1)\cdot 0} & \dots & r^{(n-1)\cdot(n-1)} \end{pmatrix}$$

with entries $f_{\mu,\nu} = r^{\mu\cdot\nu}/\sqrt{n}$ and likewise zero-based indices $\mu, \nu \in \{0, \dots, n-1\}$. That is, the diagonal matrix D of the eigenvalues η_k, $k \in \{0, \dots, n-1\}$, is given by

$$D = \mathrm{diag}(\eta_0, \dots, \eta_{n-1}) := F^* A F \,, \tag{2.12}$$

with F^* denoting the conjugate transpose of F. Furthermore, the vector $\eta := (\eta_0, \dots, \eta_{n-1})^{\mathrm{t}}$ of all eigenvalues can be easily computed by multiplying the non-normalized Fourier matrix with the transposed first row of A, i.e. $\eta = \sqrt{n} F a^{\mathrm{t}}$.

In the case of the geometric transformation M given in the previous section it holds that $a = (w, \overline{w}, 0, \dots, 0)$. And like w the eigenvalues η_k depend on the base angle θ. But in order to simplify the notation, η_k will be used instead of $\eta_k(\theta)$.

Lemma 2.7. *The eigenvalues of the iteration matrix M according to* (2.11) *are given by*

$$\eta_k = w + r^k \overline{w} = \sec\theta \cos\left(\theta - \frac{\pi k}{n}\right) \mathrm{e}^{\mathrm{i}\pi k/n}\,, \tag{2.13}$$

where $k \in \{0, \dots, n-1\}$. In particular it holds that $\eta_0 = 1$.

Proof. Since the representation $\eta_k = w + r^k \overline{w}$ follows readily from the representation $\eta = \sqrt{n} F (w, \overline{w}, 0, \dots, 0)^{\mathrm{t}}$ and the definition of F, only the second representation has to be shown.

Due to $\theta \in (0, \pi/2)$, it holds that $|w| = \frac{1}{2}\sqrt{1 + \tan^2\theta} = \frac{1}{2}\sec\theta$. Furthermore, the argument of w is given by θ and that of r^k by $2\pi k/n$. Collecting the real and imaginary parts of the two summands in $\eta_k = w + r^k \overline{w}$ and applying trigonometric identities to the following expressions in squared brackets results in

$$\begin{aligned} \eta_k &= \frac{1}{2}\sec\theta\left(\left[\cos\theta + \cos\left(\frac{2\pi k}{n} - \theta\right)\right] + \mathrm{i}\left[\sin\theta + \sin\left(\frac{2\pi k}{n} - \theta\right)\right]\right) \\ &= \sec\theta\cos\left(\theta - \frac{\pi k}{n}\right)\left(\cos\frac{\pi k}{n} + \mathrm{i}\sin\frac{\pi k}{n}\right)\,, \end{aligned}$$

which implies (2.13). The special case $\eta_0 = 1$ follows likewise from (2.13) or $\eta_0 = w + \overline{w} = 2\,\mathrm{Re}\, w = 1$. □

The fact that all circulant matrices are diagonalizable by the same Fourier matrix F is remarkable and has far reaching consequences with respect to resulting symmetries. Hence, the orthonormal vectors given by the columns

$$f_k := (f_{0,k}, \dots, f_{n-1,k})^{\mathrm{t}} = \frac{1}{\sqrt{n}}\left(r^{0\cdot k}, r^{1\cdot k}, \dots, r^{(n-1)\cdot k}\right)^{\mathrm{t}}\,,$$

$k \in \{0, \dots, n-1\}$, of F build a natural basis for the analysis of circulant polygon transformations. The vectors f_k can also be interpreted as polygons, which will be called Fourier polygons.

The coefficients c_k in the representation of $z^{(0)} = \sum_{k=0}^{n-1} c_k f_k$ in terms of the Fourier polygons are given by the entries of the vector $c := F^* z^{(0)}$. Furthermore, due to (2.12) the transformation matrix M can be written as $M = F D F^*$ with D

denoting the diagonal matrix of the eigenvalues η_k. Hence, the polygon obtained by successively applying ℓ transformation steps can be written as

$$z^{(\ell)} = M^\ell z^{(0)} = (FDF^*)^\ell z^{(0)} = FD^\ell F^* z^{(0)} = FD^\ell c = \sum_{k=0}^{n-1} \eta_k^\ell c_k f_k\,. \tag{2.14}$$

That is, $z^{(\ell)}$ is a linear combination of the n polygons $c_k f_k$ scaled by the ℓ-th power of the associated eigenvalues. Therefore, this special polygons will be defined and analyzed below.

Definition 2.2. *For $k \in \{0, \dots, n-1\}$ the n-dimensional complex vector*

$$v_k := c_k f_k = \frac{\left(F^* z^{(0)}\right)_k}{\sqrt{n}} \left(r^{0\cdot k}, \dots, r^{(n-1)\cdot k}\right)^{\mathrm{t}} \tag{2.15}$$

will be called the k-th eigenpolygon of the initial polygon $z^{(0)}$.

Due to $f_0 = \frac{1}{\sqrt{n}}(1, \dots, 1)^{\mathrm{t}}$ and $\left(F^* z^{(0)}\right)_0 = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} z_k^{(0)}$, each vertex of the associated degenerated eigenpolygon v_0 is $\frac{1}{n} \sum_{k=0}^{n-1} z_k^{(0)}$ representing the centroid of $z^{(0)}$. Furthermore, due to $\eta_0 = 1$ and the decomposition (2.14) it holds that

$$z^{(\ell)} = \sum_{k=0}^{n-1} \eta_k^\ell v_k = v_0 + \sum_{k=1}^{n-1} \eta_k^\ell v_k\,. \tag{2.16}$$

It should be noted that the eigenpolygons v_k do not necessarily form a basis of $\mathbb{C}^n$, since some of the coefficients c_k might be zero causing the associated v_k to become zero vectors. However, since $\eta_0 = 1$ in contrast to all other eigenvalues does not depend on the base angle θ, v_o representing the centroid is always preserved as can be seen by equation (2.16).

Since all circulant $(n \times n)$-matrices can be diagonalized by the same Fourier matrix F, all geometric transformations represented by such matrices lead to the same eigenpolygons. Therefore, the eigenpolygons do not depend on the base angle θ, but only on the initial polygon $z^{(0)}$. Hence, different transformation schemes result in different eigenvalues thus placing emphasis on different symmetric configurations.

Since the eigenpolygon v_k is the Fourier polygon f_k times the coefficient $c_k \in \mathbb{C}$, v_k preserves the symmetry of the f_k. This will be stated more precisely in the following lemma, since this symmetry is also reflected by the limit polygons of the iterated transformation.

Lemma 2.8. *For $k, \mu \in \{0, \dots, n-1\}$ it holds that*

$$(v_k)_\mu = r^{\mu k} (v_k)_0\,, \tag{2.17}$$

that is, the μ-th vertex of the eigenpolygon v_k can be derived by rotating the first vertex by angle $2\pi\mu k/n$. In particular it follows that $(v_k)_{\mu \bmod n} = r^k (v_k)_{\mu-1}$ where $\mu \in \{1, \ldots, n\}$.

Proof. This follows readily from equation (2.15) since

$$(v_k)_\mu = c_k (f_k)_\mu = c_k \left(\frac{1}{\sqrt{n}} r^{\mu k} \right) = r^{\mu k} \left(\frac{c_k}{\sqrt{n}} \cdot 1 \right) = r^{\mu k} (v_k)_0 \,.$$

The rotation angle can be derived from $r^{\mu k} = \exp(2\pi \mathrm{i} \mu k/n)$ and the representation $(v_k)_{\mu \bmod n} = r^k (v_k)_{\mu-1}$ is implied by equation (2.17). □

If ℓ tends to infinity, $z^{(\ell)}$ tends to the scaled eigenpolygon belonging to the eigenvalue with the largest absolute value. Using (2.13) in order to determine the dominating eigenvalue implies

$$|\eta_k| = \sec\theta \left| \cos\left(\theta - \frac{\pi k}{n}\right) \right| \,.$$

Fig. 2.6 depicts the values of $|\eta_k|$ depending on $\theta \in (0, \pi/2)$ for $k \in \{0, \ldots, n-1\}$ in the case of $n \in \{5, 6\}$. One can observe that intersections of the functions $|\eta_k|$ occur only at angles θ which are multiples of $\pi/(2n)$. This is stated by the following lemma.

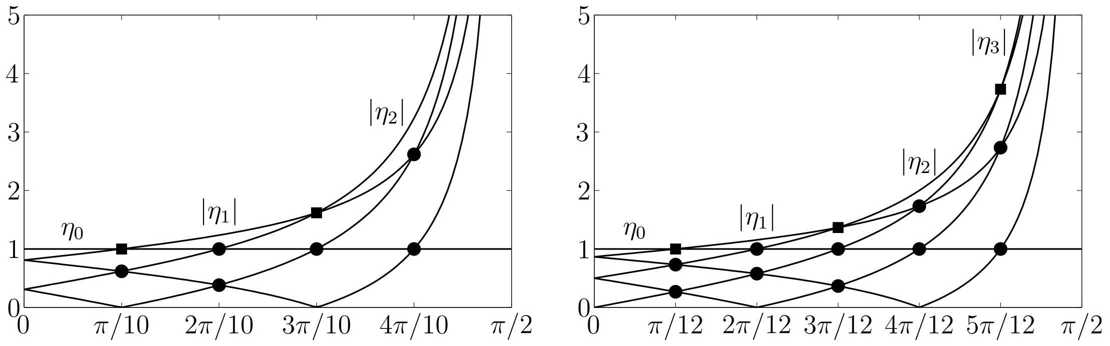


Figure 2.6: Absolute values of eigenvalues for $\theta \in (0, \pi/2)$ in the case of $n = 5$ (left) and $n = 6$ (right). Dominating eigenvalues are named

Lemma 2.9. *For $k, m \in \{0, \ldots, n-1\}$, $k \neq m$, the functions $|\eta_k|$ and $|\eta_m|$ of $\theta \in (0, \pi/2)$ may only intersect for $\theta = \mu\pi/(2n)$ where $\mu \in \{1, \ldots, n-1\}$.*

Proof. Since $|\eta_k|$ and $|\eta_m|$ are positive, this will be shown by analyzing the roots of the difference function of the squared values

$$\begin{aligned} d_{k,m}(\theta) &:= |\eta_k|^2 - |\eta_m|^2 = \sec^2\theta \left(\cos^2\left(\theta - \frac{\pi k}{n}\right) - \cos^2\left(\theta - \frac{\pi m}{n}\right) \right) \\ &= \sec^2\theta \sin\left(2\theta - \pi\,\frac{k+m}{n}\right) \sin\left(\pi\,\frac{k-m}{n}\right) \,. \end{aligned} \tag{2.18}$$

Since $|k-m|/n \in \{1/n, \ldots, (n-1)/n\}$, roots can only occur, if the argument of the first sine factor in (2.18) is a multiple of π, that is

$$2\theta - \pi\frac{k+m}{n} = \nu\pi \quad \Leftrightarrow \quad \theta = \frac{\pi}{2}\left(\nu + \frac{k+m}{n}\right), \quad \nu \in \mathbb{Z}\,.$$

Due to $\theta \in (0, \pi/2)$, the factor $\nu + (k+m)/n$ has to be in $(0,1)$ thus implying $k+m \neq n$ and $\nu = -\lfloor (k+m)/n \rfloor$, where $\lfloor \cdot \rfloor$ denotes rounding towards zero. Since $k \neq m$, this results in $(k+m) \in \{1, \ldots, 2n-3\} \setminus \{n\}$.

In the case of $(k+m) \in \{1, \ldots, n-1\}$ and $(k+m) \in \{n+1, \ldots, 2n-3\}$ it follows that $\nu = 0$ and $\nu = -1$ respectively, thus providing the roots

$$\theta = \frac{\pi}{2n}(k+m) \quad \text{and} \quad \theta = \frac{\pi}{2n}(k+m-n)$$

respectively, as stated in the lemma. □

The behavior of $z^{(\ell)}$ depends on the dominating eigenvalue, which itself depends on the base angle θ. As can be seen in Fig. 2.6, the dominating eigenvalue changes at each odd multiple of $\pi/(2n)$ marked by square markers. This is stated by the following lemma, which also gives the index of the dominating eigenvalue for each interval.

Lemma 2.10. *For $k \in \{0, \ldots, \lfloor n/2 \rfloor - 1\}$, where $\lfloor \cdot \rfloor$ denotes rounding towards zero, let*

$$\theta_k := \frac{\pi}{2n}(2k+1)\,, \quad \textit{furthermore} \quad \theta_{-1} := 0, \quad \theta_{\lfloor n/2 \rfloor} := \frac{\pi}{2}\,. \tag{2.19}$$

On each interval (θ_{k-1}, θ_k), $k \in \{0, \ldots, \lfloor n/2 \rfloor\}$, the index of the dominating eigenvalue is given by k, that is

$$\theta \in (\theta_{k-1}, \theta_k) \quad \Rightarrow \quad |\eta_k| > |\eta_m| \quad \forall m \in \{0, \ldots, n-1\} \setminus \{k\}\,. \tag{2.20}$$

In addition, for $k \in \{0, \ldots, \lfloor n/2 \rfloor - 1\}$

$$\theta = \theta_k \quad \Rightarrow \quad |\eta_k| = |\eta_{k+1}| > |\eta_m| \quad \forall m \in \{0, \ldots, n-1\} \setminus \{k, k+1\}\,, \tag{2.21}$$

that is, for $\theta = \theta_k$ the function $|\eta_k|$ is only intersected by $|\eta_{k+1}|$.

Proof. Induction over the intervals (θ_{k-1}, θ_k) for $k \in \{0, \ldots, \lfloor n/2 \rfloor\}$ will be used. In the case of $k = 0$ implying $\theta \in (\theta_{-1}, \theta_0) = \big(0, \pi/(2n)\big)$ equation (2.20) holds, if the associated difference function $d_{0,m}(\theta)$ according to (2.18) is positive for all $m \in \{1, \ldots, n-1\}$. This is the case, since

$$d_{0,m}(\theta) = \underbrace{\sec^2\theta}_{>0}\,\underbrace{\sin\left(2\theta - \pi\,\frac{m}{n}\right)}_{<0}\,\underbrace{\sin\left(\pi\,\frac{-m}{n}\right)}_{<0} > 0\,,$$

due to $\sec^2\theta > 0$ in the case of the first factor, $-\pi < 2\cdot 0 - \pi\frac{n-1}{n} < 2\theta - \pi\frac{m}{n} < 2\frac{\pi}{2n} - \pi\frac{1}{n} = 0$ in the case of the second factor, and $-\pi\frac{m}{n} \in (-\pi, 0)$ in the case of the third factor.

In order to show (2.21) the equation

$$d_{0,m}\left(\frac{\pi}{2n}\right) = \underbrace{\sec^2\frac{\pi}{2n}}_{\neq 0} \sin\left(\frac{\pi}{n} - \pi\frac{m}{n}\right) \underbrace{\sin\left(\pi\frac{-m}{n}\right)}_{\neq 0} \overset{!}{=} 0$$

implies $\frac{\pi}{n} - \pi\frac{m}{n} = \pi\frac{1-m}{n} \overset{!}{=} \nu\pi$ with $\nu \in \mathbb{Z}$, hence $m = 1$, since $m \in \{1, \dots, n-1\}$. That is, for $\theta = \theta_0$ the function $|\eta_0|$ is only intersected by $|\eta_1|$. Furthermore, since all $|\eta_k|$ are continuous as well as differentiable on $(0, \pi/2)$ except the points where $|\eta_k| = 0$, and dominated by $|\eta_0|$ on $\big(0, \pi/(2n)\big)$, it holds that $1 = \eta_0 = |\eta_1| > |\eta_m|$ for $m \in \{2, \dots, n-1\}$ in θ_0.

For a given $k \in \{1, \dots, \lfloor n/2 \rfloor - 1\}$ it will now be assumed that for θ_{k-1} the dominating eigenvalue changes from η_{k-1} to η_k. In order to prove (2.20) for the interval (θ_{k-1}, θ_k) it suffices to show that $|\eta_k|$ is not intersected by any other $|\eta_m|$ with $m \in \{0, \dots, n-1\} \setminus \{k\}$.

According to lemma 2.9 intersections may only occur for $\theta = \pi k/n$, which is the midpoint of the interval (θ_{k-1}, θ_k). Since

$$d_{k,m}\left(\frac{\pi k}{n}\right) \overset{!}{=} 0 \quad \Rightarrow \quad \sin\left(2\frac{\pi k}{n} - \pi\frac{k+m}{n}\right) \overset{!}{=} 0$$

implies $2\frac{\pi k}{n} - \pi\frac{k+m}{n} = \frac{\pi}{n}(k-m) \overset{!}{=} \nu\pi$, $\nu \in \mathbb{Z}$ and therefore $m = k$, the dominating function $|\eta_k|$ is not intersected by any other $|\eta_m|$ inside of (θ_{k-1}, θ_k).

Finally, (2.21) is shown by analyzing

$$d_{k,m}\left(\theta_k\right) = \sec^2\theta_k \, \sin\left(\pi\frac{k+1-m}{n}\right) \sin\left(\pi\frac{k-m}{n}\right) \overset{!}{=} 0\,.$$

The first sine becomes zero in the case of $m = k+1$. The second sine, which does not depend on θ, is nonzero due to $m \neq n-k$. Since the difference function changes its sign in θ_k, the dominating eigenvalue changes from η_k on the left of θ_k to η_{k+1} on the right as stated. □

2.2.4 Limit polygons

In the case of $\theta > \pi/(2n)$, the absolute value of the dominating eigenvalue is greater than one causing the vertices of $z^{(\ell)}$ to tend to infinity if ℓ increases. Therefore, the polygons are scaled according to the following definition.

Definition 2.3. *For $\theta \in (\theta_{k-1}, \theta_k]$, $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, with $\theta \neq \pi/2$ let*

$$z_{s,\theta}^{(\ell)} := v_0 + \frac{1}{|\eta_k|^\ell}\bigl(z^{(\ell)} - v_0\bigr) = v_0 + \sum_{\mu=1}^{n-1} \left(\frac{\eta_\mu}{|\eta_k|}\right)^\ell v_\mu \tag{2.22}$$

be the sequence of polygons scaled according to the centroid of $z^{(0)}$ and the dominating eigenvalue with index k.

The rightmost representation of $z_{s,\theta}^{(\ell)}$ follows readily from (2.16). It also holds that $z^{(\ell)} = z_{s,\theta}^{(\ell)}$ in the case of $\theta \in \bigl(0, \pi/(2n)\bigr]$. That is, scaling has no effect in this case since the dominating eigenvalue is given by $\eta_0 = 1$. Based on the results obtained so far, a full classification of the behavior of the sequence $z_{s,\theta}^{(\ell)}$ for $\ell \to \infty$ can now be given.

Theorem 2.10. *For the base angle $\theta \in \bigl(0, \pi/2\bigr)$ and $\ell \in \mathbb{N}$, the scaled polygons $z_{s,\theta}^{(\ell)}$ tend to the limit polygons*

$$p_{s,\theta}^{(\ell)} := \begin{cases} v_0 & \text{if } \theta \in (0, \theta_0), \\ v_0 + \mathrm{e}^{\mathrm{i}\pi\ell/n} v_1 & \text{if } \theta = \theta_0, \\ v_0 + \mathrm{e}^{\mathrm{i}\pi k\ell/n} v_k & \text{if } \theta \in (\theta_{k-1}, \theta_k), k \in \{1, \dots, \lfloor n/2 \rfloor\}, \\ v_0 + \mathrm{e}^{\mathrm{i}\pi k\ell/n} v_k + \mathrm{e}^{\mathrm{i}\pi(k+1)\ell/n} v_{k+1} & \text{if } \theta = \theta_k, k \in \{1, \dots, \lfloor n/2 \rfloor - 1\}, \end{cases} \tag{2.23}$$

with θ_k given by (2.19). That is $\lim_{\ell\to\infty} \left\| z_{s,\theta}^{(\ell)} - p_{s,\theta}^{(\ell)} \right\| = 0$, with $\|\cdot\|$ denoting the norm defined by $\|x\| = \sqrt{x^* x}$.

Proof. All four cases will be shown in the following manner. Based on the results of lemma 2.10 the dominating terms, which lead to the definition of $p_{s,\theta}^{(\ell)}$, will be separated in the scaled decomposition (2.22) from a remaining finite distortion sum. The latter will tend to the zero vector if ℓ tends to infinity.

In the case of $\theta \in (0, \theta_0)$ the dominating eigenvalue is given by η_0 with the associated eigenpolygon $v_0 = p_{s,\theta}^{(\ell)}$. It follows readily from (2.16) that

$$\lim_{\ell\to\infty} \left\| z_{s,\theta}^{(\ell)} - p_{s,\theta}^{(\ell)} \right\| = \lim_{\ell\to\infty} \left\| \sum_{\mu=1}^{n-1} \eta_\mu^\ell v_\mu \right\| \leq \sum_{\mu=1}^{n-1} \|v_\mu\| \underbrace{\lim_{\ell\to\infty} |\eta_\mu|^\ell}_{=0} = 0\,,$$

since $|\eta_\mu| < 1$ for $\mu \in \{1, \dots, n-1\}$ according to lemma 2.10.

In the case of $\theta = \theta_0 = \pi/(2n)$ the dominating eigenvalues are η_0 and η_1 with $\eta_0 = 1 = |\eta_1|$, where

$$\eta_1 = \sec\frac{\pi}{2n} \cos\left(\frac{\pi}{2n} - \frac{\pi}{n}\right) \mathrm{e}^{\mathrm{i}\pi/n} = \mathrm{e}^{\mathrm{i}\pi/n}\,.$$

Thus it can be stated that

$$z_{s,\theta_0}^{(\ell)} = v_0 + \eta_1^\ell v_1 + \sum_{\mu=2}^{n-1} \eta_\mu^\ell v_\mu = p_{s,\theta_0}^{(\ell)} + \sum_{\mu=2}^{n-1} \eta_\mu^\ell v_\mu \,.$$

Again, the distortion sum tends to the zero vector if ℓ tends to infinity since $|\eta_\mu| < 1$.

In the case of $\theta \in (\theta_{k-1}, \theta_k), k \in \{1, \dots, \lfloor n/2 \rfloor\}$, it holds that

$$z_{s,\theta}^{(\ell)} = v_0 + \left(\frac{\eta_k}{|\eta_k|}\right)^\ell v_k + \sum_{\substack{\mu=1\\ \mu\neq k}}^{n-1} \underbrace{\left(\frac{\eta_\mu}{|\eta_k|}\right)^\ell}_{\to 0} v_\mu \,. \tag{2.24}$$

At this, the eigenvalue representation (2.13) implies

$$\frac{\eta_k}{|\eta_k|} = \underbrace{\operatorname{sign}\left(\sec\theta \cos\left(\theta - \frac{\pi k}{n}\right)\right)}_{=1} \mathrm{e}^{\mathrm{i}\pi k/n} = \mathrm{e}^{\mathrm{i}\pi k/n} \,,$$

with $\theta \in (\theta_{k-1}, \theta_k)$, since the argument of the cosine factor varies within the interval $(-\pi/(2n), \pi/(2n))$. Replacing the obtained exponential representation of $\eta_k/|\eta_k|$ in (2.24) yields the limit polygon as stated in the third case of (2.23).

Finally, in the case of $\theta = \theta_k$, $k \in \{1, \dots, \lfloor n/2 \rfloor - 1\}$ with the dominating eigenvalues η_k and η_{k+1} it holds that

$$z_{s,\theta_k}^{(\ell)} = v_0 + \underbrace{\left(\frac{\eta_k}{|\eta_k|}\right)^\ell v_k}_{=\mathrm{e}^{\mathrm{i}\pi k\ell/n} v_k} + \left(\frac{\eta_{k+1}}{|\eta_k|}\right)^\ell v_{k+1} + \sum_{\substack{\mu=1\\ \mu\neq k,k+1}}^{n-1} \underbrace{\left(\frac{\eta_\mu}{|\eta_k|}\right)^\ell}_{\to 0} v_\mu \,,$$

due to the results of the previous case for the second summand. In the case of the third summand it holds that

$$\frac{\eta_{k+1}}{|\eta_k|} = \underbrace{\operatorname{sign}\left(\sec\theta_k \cos\left(\theta_k - \frac{\pi(k+1)}{n}\right)\right)}_{=1} \mathrm{e}^{\mathrm{i}\pi(k+1)/n} = \mathrm{e}^{\mathrm{i}\pi(k+1)/n},$$

since the argument of the cosine function is $-\pi/(2n)$. □

It should be noted that the limit polygons $p_{s,\theta}^{(\ell)}$ depend on the iteration number ℓ due to the exponential coefficients of the eigenpolygons, which reflect the rotational effect of the transformation. Furthermore, only the counterclockwise oriented eigenpolygons v_k, with $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, are involved.

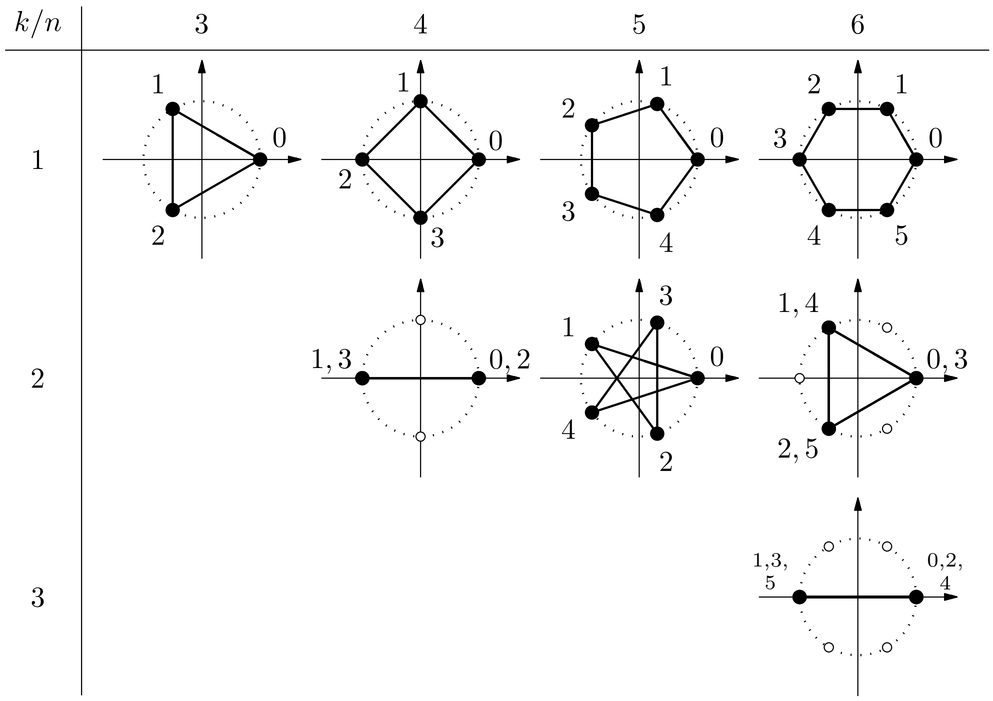


Figure 2.7: Fourier polygons f_k for $n \in \{3, \ldots, 6\}$ and $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$

The following will now focus on the shapes of the limit polygons given by theorem 2.10 and therefore on the shapes of the eigenpolygons v_k, which are scaled Fourier polygons. The latter are depicted in Fig. 2.7 for $n \in \{3, \ldots, 6\}$ and $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$. In this, circle markers indicate the scaled roots of unity lying on a circle with radius $1/\sqrt{n}$, whereas solid black markers indicate the vertices of the Fourier polygons. Also given is the vertex index or, in the case of multiple vertices, a comma separated list of indices. The case $k = 0$, where f_0 degenerates to one vertex with multiplicity n, is omitted. The shape of eigenpolygons is described by the following theorem.

Theorem 2.11. *Let* $\gcd(n, k)$ *denote the greatest common divisor of the two natural numbers* n *and* k*. Then the following holds for the eigenpolygons* v_k *and therefore for the limit polygons of* $z^{(\ell)}_{s,\theta}$ *in the case of* $\theta \in (\theta_{k-1}, \theta_k)$*,* $k \in \{1, \ldots, \lfloor n/2 \rfloor\}$*, and* $c_k \neq 0$*:*

1. *The eigenpolygon* v_k *is similar to the polygon obtained by successively connecting counterclockwise each* k*-th root of unity.*

2. *The eigenpolygon* v_k *is a* $\big(n/\gcd(n, k)\big)$*-gon with vertex multiplicity* $\gcd(n, k)$*.*

3. *The eigenpolygon* v_k *is a convex regular* (n/k)*-gon, if and only if* $k = \gcd(n, k)$*.*

Proof. The shape of the eigenpolygon v_k as described in item 1 follows readily from lemma 2.8.

To show item 2 let $m := \gcd(n,k)$, which results in the representations $n = m \cdot \hat{n}$ and $k = m \cdot \hat{k}$ with $\hat{n}, \hat{k} \in \mathbb{N}$. According to item 1, numbering n/m roots of unity by connecting each k root beginning with r^0 ends again in r^0, since

$$\left(\frac{n}{m} \cdot k\right) \bmod n = (\hat{n} \cdot m \cdot \hat{k}) \bmod n = (n \cdot \hat{k}) \bmod n = 0\,, \tag{2.25}$$

which implies $r^{nk/m} = r^0$. The following steps reproduce this polygon until each vertex has multiplicity m. Since m is the greatest common divisor, there is no larger divisor than m resulting in a polygon with less vertices.

To prove item 3, one can observe in the case of $k \neq \gcd(n,k)$ that according to (2.25) it takes $\hat{k}$ loops of roots of unity to return to r^0 leading to intersecting sides and therefore star shaped polygons, whereas in the case of $k = \gcd(n,k)$ one returns after n/k steps within one loop to r^0 thus leading to a convex polygon due to the geometry of the roots of unity. □

The angles given by the exponential factors in the definition (2.23) of the scaled limit polygons reveal the rotational effect of the transformation. They depend not only on the number n of vertices and ℓ of iterations, but also on the index k of the associated eigenpolygon thus leading to more symmetries as stated in the following theorem.

Theorem 2.12. *The limit polygons $p_{s,\theta}^{(\ell)}$ differ for odd and likewise for even iteration numbers ℓ only in a counterclockwise cyclic shift of vertex indices, but not in geometry. More precisely, it holds that*

$$\left(p_{s,\theta}^{(\ell)}\right)_{(\mu+1) \bmod n} = \left(p_{s,\theta}^{(\ell+2)}\right)_{\mu}\,, \tag{2.26}$$

where $\ell \in \mathbb{N}$, $\mu \in \{0,\dots,n-1\}$. Furthermore, for $\theta \in (\pi/(2n), \pi/2)$ the sequence of scaled polygons $z_{s,\theta}^{(\ell)}$ contains $2n$ converging subsequences with

$$\lim_{\ell\to\infty} z_{s,\theta}^{(2n\ell+\nu)} = p_{s,\theta}^{(\nu)}\,, \tag{2.27}$$

where $\nu \in \{0,\dots,2n-1\}$.

Proof. Since $p_{s,\theta}^{(\ell)}$ is a linear combination of rotated eigenpolygons, it suffices to show that (2.26) holds for each summand, that is

$$\left(\mathrm{e}^{\mathrm{i}\pi k\ell/n} v_k\right)_{(\mu+1) \bmod n} = \left(\mathrm{e}^{\mathrm{i}\pi k(\ell+2)/n} v_k\right)_{\mu}\,,$$

where $k \in \{1, \dots, n-1\}$. Using lemma 2.8 and the definition of the root of unity $r = \exp(2\pi\mathrm{i}/n)$ this follows from

$$\mathrm{e}^{\mathrm{i}\pi k\ell/n} \left(v_k\right)_{(\mu+1) \bmod n} = \mathrm{e}^{\mathrm{i}\pi k\ell/n} r^k \left(v_k\right)_\mu = \mathrm{e}^{\mathrm{i}\pi k(\ell+2)/n} \left(v_k\right)_\mu .$$

Since in the case of $k = 0$ all entries of v_0 are the same this implies (2.26).

Equation (2.27) follows from the $2n$-periodicity of the exponential factors given by the definition of $p_{s,\theta}^{(\ell)}$ according to (2.23), that is $\mathrm{e}^{\mathrm{i}\pi k(2n\ell+\nu)/n} = \mathrm{e}^{\mathrm{i}\pi k\nu/n}$ for $\nu \in \{0, \dots, 2n-1\}$. $\square$

In the case $\theta \in (\theta_{k-1}, \theta_k)$, $k \in \{1, \dots, \lfloor n/2 \rfloor\}$, the limit polygon is a rotated and scaled Fourier polygon with preserved centroid similar to those depicted in Fig. 2.7. In contrast, for $\theta = \theta_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$ there are two possible limit polygons each a linear combination of up to two eigenpolygons shifted by the centroid.

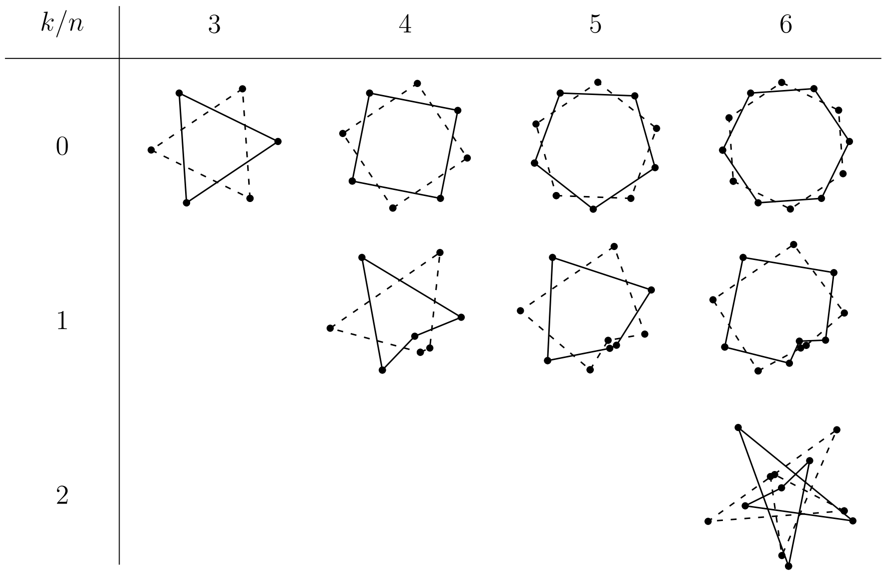


Figure 2.8: Limit polygons for $n \in \{3, \dots, 6\}$, $\theta = \theta_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$. Drawn through lines and dashed lines indicate limits for subsequences with odd and even ℓ respectively

For $(z^{(0)})_\mu := (\mu + 1) \exp(\mathrm{i}\pi\mu/(5n))$, $n \in \{3, \dots, 6\}$, $\mu \in \{0, \dots, n-1\}$, and $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$ the resulting two limit polygons are depicted in Fig. 2.8. Drawn through lines and dashed lines indicate limits for subsequences with odd and even ℓ respectively. In particular the first row contains two regular n-gons rotated by angle π/n, which is half the angle of the according root of unity.

The first case in Fig. 2.8, that is $n = 3$, $k = 0$ and therefore $\theta = \pi/6$, represents Napoleon's theorem. In particular, since $\eta_0 = |\eta_1| = 1$ and $\eta_2 = 0$ the distortion sum $z_{s,\theta}^{(\ell)} - p_{s,\theta}^{(\ell)}$ is the zero vector for all ℓ. Therefore one step suffices to generate a regular triangle. From that point on, according to theorem 2.12, further iterations lead to a sequence of two alternating dual triangles which are depicted in the table.

In the case of $n > 3$, $k = 0$ the distortion sum is nonzero, hence there is only one Napoleon. But since the error decreases by powers of the eigenvalue quotients this leads to a fast convergence and therefore asymptotically also to alternating dual regular n-gons.

In the following, an overview of properties of the unscaled iterative polygon transformation is given, which can now be readily derived from the results obtained so far.

Corollary 2.3. *The following holds:*

1. *For $\theta \in \big(0, \pi/(2n)\big)$ the concentric polygons $z^{(\ell)}$ degenerate to their common centroid if ℓ tends to infinity.*

2. *In the case of $\theta = \pi/(2n)$ the sequence $z^{(\ell)}$ consists of bounded polygons which become regular if ℓ tends to infinity.*

3. *For $\theta \in \big(\pi/(2n), \pi/2\big)$ the vertices of $z^{(\ell)}$ tend to infinity as ℓ grows. In this, the limit polygons given by theorem 2.10, scaled with respect to the centroid by the ℓ-th power of the eigenvalue with the largest absolute value, represent the dominating terms thus the asymptotical behavior.*

Proof. The items 1 and 2 follow due to $z^{(\ell)} = z_{s,\theta}^{(\ell)}$ since $\theta \in (0, \pi/(2n)]$ and the results given by theorem 2.10. Likewise item 3, where $z_{s,\theta}^{(\ell)}$ differs from $z^{(\ell)}$ just in the scaling by $|\eta_k|$, which is greater than one because of $\theta > \pi/(2n)$ and the results given by lemma 2.10. It should be noted that in this case the distortion sum may also grow to infinity, since the absolute values of other eigenvalues might be greater than one. Nevertheless, due to the scaling by powers of eigenvalues, the dominating eigenvalue(s) define the behavior of the sequence $z^{(\ell)}$ if ℓ tends to infinity. $\square$

The eigenpolygon decomposition presented in this section has been used to prove that specific geometric transformations result in regular polygons. Beyond that, it can also be used in order to find new geometric construction schemes leading to predefined symmetric configurations. As described in [147], this is done by an appropriate choice of the eigenvalues η_k and by interpreting the resulting transformation matrix $M = FDF^*$ geometrically.

2.3 A generalized polygon transformation

From an application point of view, the rotational effect of the transformation analyzed in the previous section is a major drawback, since it changes the orientation of the element and does not lead to a convergent series of scaled polygons. Therefore, a modified similar triangles-based transformation will be considered in the following, which also eliminates the rotational effect of the isosceles triangles-based transformation.

2.3.1 Definition of the generalized transformation

As in the previous section, let $z^{(0)} = (z_0^{(0)}, \dots, z_{n-1}^{(0)})^{\mathrm{t}} \in \mathbb{C}^n$ denote an arbitrary polygon in the complex plane with $n \geq 3$ vertices $z_\mu^{(0)}$ using zero-based indices $\mu \in \{0, \dots, n-1\}$ and sides $z_\mu^{(0)} z_{(\mu+1) \bmod n}^{(0)}$ oriented according to the order of vertices given by the vector $z^{(0)}$. The basic idea is to transform such a polygon by constructing equally oriented similar triangles on each side and taking the apices of these triangles which leads to a new polygon with n vertices. In order to analyze this kind of transformation, two parameters λ and θ are introduced, which uniquely define the similar triangles.

Here $\lambda \in (0, 1)$ represents the subdivision ratio used for all sides. At each subdivision vertex $\lambda z_\mu^{(0)} + (1-\lambda) z_{(\mu+1) \bmod n}^{(0)}$ a perpendicular is constructed to the right of the directed side on which the apex $z_{(\mu+1) \bmod n}^{(1/2)}$ is chosen in such a way that the triangle side $z_\mu^{(0)} z_{(\mu+1) \bmod n}^{(1/2)}$ and the polygon side $z_\mu^{(0)} z_{(\mu+1) \bmod n}^{(0)}$ enclose a predefined angle $\theta \in (0, \pi/2)$. Since the associated height is defined by the length of the side times $(1-\lambda)\tan\theta$, it holds that the apex is given by

$$z_{(\mu+1) \bmod n}^{(1/2)} = w z_\mu^{(0)} + (1-w) z_{(\mu+1) \bmod n}^{(0)}, \tag{2.28}$$

with the complex weight $w := \lambda + \mathrm{i}(1-\lambda)\tan\theta$ depending on the parameters θ and λ.

Hence, the linear transformation G_- mapping the polygon $z^{(0)}$ to the associated polygon $z^{(1/2)}$ of apices can be represented by the $(n \times n)$-matrix M_- with entries

$$(M_-)_{\mu,\nu} := \begin{cases} 1-w & \text{if } \mu = \nu \\ w & \text{if } \mu = (\nu+1) \bmod n \\ 0 & \text{otherwise} \end{cases}$$

using zero-based indices $\mu, \nu \in \{0, \dots, n-1\}$. That is $z^{(1/2)} = M_- z^{(0)}$.

This is depicted in Fig. 2.9 using $\lambda = 1/3$ and $\theta = \pi/4$. Starting from the initial polygon with vertices $z_\mu^{(0)}$, marked by a black drawn through line, the construction is performed according to (2.28). This leads to the polygon of apices $z_\mu^{(1/2)}$ marked by a blue drawn through line. In both cases only some sides of the polygon are depicted in order to simplify the figure. Gray arcs indicate angles θ and the perpendiculars subdivide the polygon sides in a ratio of $(1-\lambda):\lambda$.

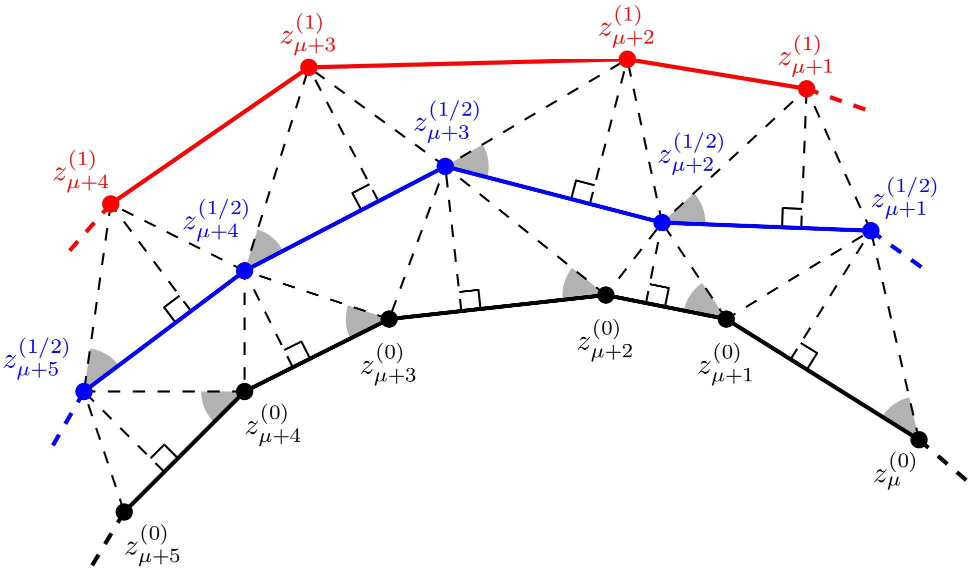


Figure 2.9: Initial polygon $z^{(0)}$ (black), G_- transformed polygon $z^{(1/2)} = M_- z^{(0)}$ (blue), and G_+ transformed polygon $z^{(1)} = M_+ z^{(1/2)} = M z^{(0)}$ (red)

Due to its construction, the transformation G_- has a rotational effect as will be shown later. Therefore, a second transformation named G_+ using the same construction parameters but with flipped similar triangles is applied in order to eliminate this effect. The resulting polygon with vertices $z_\mu^{(1)}$ is marked by a red drawn through line. The representation of the vertices $z_\mu^{(1)}$ can be derived analogously to (2.28) and is given by

$$z_\mu^{(1)} = (1-\overline{w}) z_\mu^{(1/2)} + \overline{w} z_{(\mu+1) \bmod n}^{(1/2)}, \tag{2.29}$$

where $\overline{w}$ denotes the conjugate complex of w. Hence, the matrix representation M_+ of the transformation G_+ is given by

$$(M_+)_{\mu,\nu} := \begin{cases} 1-\overline{w} & \text{if } \mu = \nu \\ \overline{w} & \text{if } \nu = (\mu+1) \bmod n \\ 0 & \text{otherwise} \end{cases},$$

with $z^{(1)} = M_+ z^{(1/2)}$. In Fig. 2.9 the transformation maps the blue polygon with vertices $z_\mu^{(1/2)}$ to the red polygon with vertices $z_\mu^{(1)}$. As can be seen, the subdivision ratio is interchanged and the base angle θ is positioned at the end of the side with respect to the orientation.

The resulting combined transformation $G := G_+ \circ G_-$ and its matrix representation M will be defined and analyzed in the following. In Fig. 2.9 it directly maps the polygon $z^{(0)}$ marked black to the polygon $z^{(1)}$ marked red.

Definition 2.4. *For $\theta \in (0, \pi/2)$, $\lambda \in (0,1)$, and*

$$w := \lambda + \mathrm{i}(1-\lambda)\tan\theta$$

let G denote the polygon transformation $z^{(1)} = Mz^{(0)}$ defined by the matrix

$$(M)_{\mu,\nu} := \begin{cases} |1-w|^2 + |w|^2 & \textit{if } \mu = \nu \\ w(1-\overline{w}) & \textit{if } \mu = (\nu+1) \bmod n \\ \overline{w}(1-w) & \textit{if } \nu = (\mu+1) \bmod n \\ 0 & \textit{otherwise} \end{cases}, \tag{2.30}$$

where $\mu, \nu \in \{0, \dots, n-1\}$.

The entries of M can be easily derived by multiplying M_- and M_+ using that $w\overline{w} = |w|^2$. Due to the geometric construction M_- as well as M_+ are circulant and adjoint matrices, and, as a product of two circulant matrices, $M = M_-M_+ = M_+M_-$ is circulant and according to (2.30) Hermitian. Furthermore, each row and column of the matrices M, M_-, and M_+ sum up to one, which implies that all transformations preserve the centroid, that is,

$$\frac{1}{n}\sum_{\mu=0}^{n-1} z_\mu^{(0)} = \frac{1}{n}\sum_{\mu=0}^{n-1} z_\mu^{(1/2)} = \frac{1}{n}\sum_{\mu=0}^{n-1} z_\mu^{(1)},$$

which can be shown by inserting the representations (2.28), (2.29) and rearranging the sum in order to collect the coefficients of each vertex.

2.3.2 Eigenvalues and eigenpolygon decomposition

As in the case of the isosceles triangles-based transformation considered in the previous section, the modified transformation G will be analyzed using the eigenpolygon decomposition and the eigenvalues of M. Since M is circulant its eigenvectors are the columns of the unitary $n \times n$ Fourier matrix F with entries $(F)_{\mu,\nu} := r^{\mu\cdot\nu}/\sqrt{n}$, $\mu, \nu \in \{0, \dots, n-1\}$, and $r := \exp(2\pi\mathrm{i}/n)$ denoting a complex root of unity [145]. This implies the following results.

Lemma 2.11. *The eigenvalues of the transformation matrix M given by Definition 2.4 are*

$$\eta_k := \left|1 - \overline{w} + r^k\overline{w}\right|^2 = |1-w|^2 + |w|^2 + 2\operatorname{Re}\left(r^k\overline{w}(1-w)\right), \tag{2.31}$$

with $k \in \{0, \ldots, n-1\}$.

Proof. The eigenvalues of an arbitrary circulant matrix C can be computed by multiplying $\sqrt{n}F$ with the transposed first row of C. Hence (2.31) can be obtained by simplifying $\eta_k = \sum_{\nu=0}^{n-1} r^{k\cdot\nu}(M)_{0,\nu}$ or alternatively in preparation of following results by multiplying the eigenvalues

$$\begin{aligned} \eta_k^{(-)} &= \sum_{\nu=0}^{n-1} r^{k\cdot\nu}(M_-)_{0,\nu} = r^{k\cdot 0}(1-w) + r^{k\cdot(n-1)}w \\ \eta_k^{(+)} &= \sum_{\nu=0}^{n-1} r^{k\cdot\nu}(M_+)_{0,\nu} = r^{k\cdot 0}(1-\overline{w}) + r^{k\cdot 1}\overline{w} \end{aligned}$$

of the circulant matrices M_- and M_+ having the same eigenvectors. Due to $\overline{r^k} = r^{k(n-1)}$ it follows that $\eta_k^{(-)} = \overline{\eta_k^{(+)}}$, which implies that the eigenvalues of M are given by

$$\eta_k = \eta_k^{(+)}\eta_k^{(-)} = \eta_k^{(+)}\overline{\eta_k^{(+)}} = \left|1 - \overline{w} + r^k\overline{w}\right|^2.$$

The second representation of η_k given by the right hand side of (2.31) follows by expanding $\eta_k = \eta_k^{(+)}\eta_k^{(-)}$ and using $u\overline{u} = |u|^2$ and $u + \overline{u} = 2\operatorname{Re} u$. □

In contrast to the complex eigenvalues of M_- and M_+, the eigenvalues of M according to the representation (2.31) are real valued and positive. Furthermore, it holds that $\eta_0 = 1$ for all $\lambda \in (0,1)$ and $\theta \in (0, \pi/2)$. The eigenvalues of a circulant polygon transformation matrix play a central role as has been shown in the previous section by the eigenpolygon decomposition

$$z^{(\ell)} = M^\ell z^{(0)} = (FDF^*)^\ell z^{(0)} = FD^\ell F^* z^{(0)} = \sum_{k=0}^{n-1} \eta_k^\ell v_k\,,$$

according to (2.14) and (2.16).

The eigenpolygon v_0 is a degenerate representing n times the centroid of $z^{(0)}$ and v_1 represents a counterclockwise oriented regular n-gon. This is depicted in Fig. 2.10 showing the decomposition $z^{(0)} = \sum_{k=0}^{n-1} v_k$ of random n-gons in the case of $n \in \{5, 6\}$. Here, the first three vertices have been colored red, green, and blue respectively in order to denote the orientation.

The eigenpolygons are from left to right the centroid v_0, the counterclockwise oriented regular n-gon v_1, and finally the clockwise oriented regular n-gon v_{n-1}.

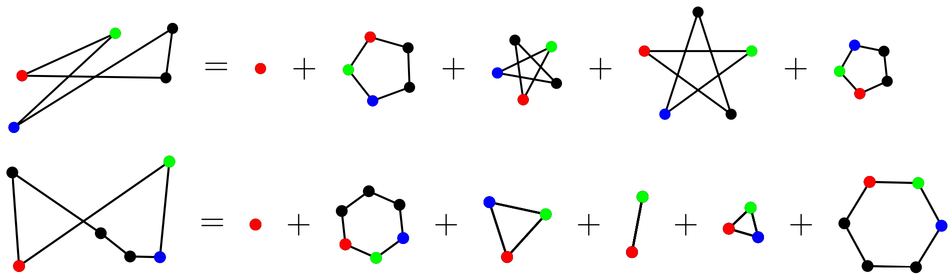


Figure 2.10: Decomposition of a random 5-gon (upper) and 6-gon (lower) into its eigenpolygons v_k

Between them only star shaped n-gons occur if n is a prime number. Otherwise, folded polygons with multiple vertices exist as can be seen in the case of $n = 6$ containing two double traversed triangles of opposite orientation and a segment with vertex multiplicity three.

Special cases occur if an eigenvalue becomes zero, since in this case one step of the transformation eliminates the associated eigenpolygon in the decomposition of $z^{(1)} = Mz^{(0)}$. In order to classify these cases, the roots of the eigenvalue functions with respect to the transformation parameters are determined.

Lemma 2.12. *For $\theta \in (0, \pi/2)$, $\lambda \in (0, 1)$, the eigenvalue η_k is strictly positive, i.e. $\eta_k > 0$, if and only if $k \in \{0, \dots, \lfloor n/2 \rfloor\}$. Otherwise η_k has exactly one isolated root at $\lambda = 1/2$, $\theta = \pi(2k - n)/(2n)$.*

Proof. Since $\eta_0 = 1$ has no root it will be assumed that $k \in \{1, \dots, n - 1\}$. According to representation (2.31) the eigenvalue η_k is positive and its roots are given by the solutions of

$$1 - \overline{w} + r^k \overline{w} = 0 \quad \Leftrightarrow \quad \overline{w} = \frac{1}{1 - r^k} = \frac{1}{2}\left(1 + \mathrm{i} \cot \frac{\pi k}{n}\right) .$$

Using $\overline{w} = \lambda - \mathrm{i}(1 - \lambda) \tan \theta$ yields $\lambda = 1/2$ and

$$\theta = - \arctan \left(\cot \frac{\pi k}{n} \right) = - \left(\frac{\pi}{2} - \frac{\pi k}{n} \right) = \frac{\pi(2k - n)}{2n} .$$

Due to the restriction $\theta \in (0, \pi/2)$ this implies $k \in \{\lfloor n/2 \rfloor + 1, \dots, n - 1\}$ as stated by the lemma. □

Since $\eta_k^{(-)} = \overline{\eta_k^{(+)}}$ and $\eta_k = \eta_k^{(+)} \eta_k^{(-)}$, the roots given by Lemma 2.12 also hold for M_- and M_+. The special case $n = 3$ and the choice $\lambda = 1/2$ and $\theta = \pi/6$ in order to obtain $\eta_2 = 0$ and $\eta_0 = \eta_1 = 1$ represents Napoleon's theorem [148]. In that case, one step of the transformation suffices to regularize an arbitrary triangle, since the eigenpolygon v_2 is eliminated by the transformation. This is the only case where for an arbitrary initial n-gon one step leads to a regular n-gon, since according to Lemma 2.12 for $n > 3$ there is no parameter constellation, where all eigenvalues except η_0 and η_1 or η_{n-1} vanish. Circulant matrices can also be used in order to derive relations concerning the area of Napoleon triangles as has been shown in [149].

Nevertheless, all except one of the distorting eigenpolygons v_k can be successively eliminated by applying $n-2$ different transformations with parameters $\lambda = 1/2$ and $\theta = \pi(2k-n)/(2n)$, where $k \in \{1,\ldots,n-1\}$, which is the result of the well known Petr-Douglas-Neumann theorem [150–152]. Since there is only one root for each eigenvalue this choice of the parameters is unique. However, the construction depends on the orientation of the initial triangle, which leads to an inner and outer construction, as has also been shown in [153] by using circulant matrices.

2.3.3 Sequences of transformed polygons

In the following, the behavior of the polygon $z^{(\ell)}$ will be analyzed with respect to the parameters λ and θ if ℓ tends to infinity. According to the decomposition (2.16) this depends on the dominant eigenvalue η_k. In order to classify parameter domains of dominant eigenvalues, the according intersection lines are determined first.

Lemma 2.13. *For $\theta \in (0,\pi/2)$, $\lambda \in (0,1)$, the eigenvalues η_k, η_m with $k,m \in \{0,\ldots,n-1\}$ and $k \neq m$ intersect only if $(k+m) \bmod n \neq 0$. In that case $\eta_k = \eta_m$ only holds along the parameter line*

$$\lambda_{k,m}(\theta) := \sin\theta\left(\sin\theta + \cot\left(\frac{\pi}{n}(k+m)\right)\cos\theta\right), \tag{2.32}$$

where $\theta \in (0,\pi/2)$ and $0 < \lambda_{k,m}(\theta) < 1$. Furthermore, the eigenvalues only intersect pairwise.

Proof. According to (2.31) the eigenvalues are positive real valued. Therefore, eigenvalue intersection parameters can be obtained by finding the roots of the difference function $\eta_k - \eta_m$, hence the solutions of

$$\operatorname{Re}\left((r^k - r^m)\overline{w}(1-w)\right) = \operatorname{Re}\left((r^k - r^m)(\overline{w} - |w|^2)\right) = 0\,. \tag{2.33}$$

Due to

$$r^k - r^m = -2\sin\left(\frac{\pi}{n}(k-m)\right)\underbrace{\left(\sin\left(\frac{\pi}{n}(k+m)\right) - \mathrm{i}\cos\left(\frac{\pi}{n}(k+m)\right)\right)}_{=:u_1}$$

and

$$\overline{w} - |w|^2 = (1-\lambda)\underbrace{\Big(\big(\lambda - (1-\lambda)\tan^2\theta\big) - \mathrm{i}\tan\theta\Big)}_{=:u_2}$$

equation (2.33) simplifies to

$$-2\sin\Big(\frac{\pi}{n}(k-m)\Big)\,(1-\lambda)\,\mathrm{Re}(u_1u_2) = 0\,.$$

Since $(k-m) \in \{-(n-1),\dots,(n-1)\}\setminus\{0\}$ and $\lambda \in (0,1)$, the factors preceeding $\mathrm{Re}(u_1u_2)$ are nonzero. Therefore, the solutions are characterized by $\mathrm{Re}(u_1u_2) = 0$, which is

$$\sin\Big(\frac{\pi}{n}(k+m)\Big)\Big(\lambda - (1-\lambda)\tan^2\theta\Big) - \cos\Big(\frac{\pi}{n}(k+m)\Big)\tan\theta = 0\,.$$

In the case of $(k+m) \bmod n = 0$ this equation reduces to $\pm\tan\theta = 0$, which has no solution for $\theta \in (0,\pi/2)$. Otherwise, rearranging the terms leads to the equation

$$\lambda(1+\tan^2\theta) - \tan^2\theta = \cot\Big(\frac{\pi}{n}(k+m)\Big)\tan\theta\,.$$

Since the left side is linear in λ, the following explicit representation of the intersection line of the eigenvalues η_k and η_m can be derived

$$\lambda = \frac{1}{1+\tan^2\theta}\tan\theta\Big(\tan\theta + \cot\Big(\frac{\pi}{n}(k+m)\Big)\Big)\,,$$

which simplifies to (2.32).

The eigenvalues intersect only pairwise since there is no index triple $k,m,j \in \{0,\dots,n-1\}$ with $k<m<j$ fulfilling $(k+m) \bmod n = \mu$ and $(k+j) \bmod n = \mu$ where $\mu \in \{1,\dots,n-1\}$. □

Due to the sum of indices in the representation (2.32) and the periodicity of the cotangent function all index pairs $k,m \in \{0,\dots,n-1\}$ with $(k+m) \bmod n = \mu$ share the same line of intersection parameters, which can be written as $\lambda_{0,\mu}(\theta)$. Furthermore, since $k \neq m$, $k+m \neq n$ there are only $n-1$ distinct parameter lines with $\mu \in \{1,\dots,n-1\}$.

The left side of Fig. 2.11 depicts the intersection lines $\lambda_{0,\mu}$ in the case of $n \in \{5,6\}$. The representation (2.32) and the monotonicity of the cotangent function imply that the intersection lines do not intersect each other. Furthermore, it holds that $\lim_{\theta\to 0}\lambda_{0,\mu}(\theta) = 0$ and $\lim_{\theta\to\pi/2}\lambda_{0,\mu}(\theta) = 1$. Hence, the parameter lines (2.32) lead to a natural partition of the parameter domain. In Fig. 2.11, the resulting subdomains are colored according to the dominant eigenvalues. As can be seen, there is a change at each second intersection line, which leads to the following partition.

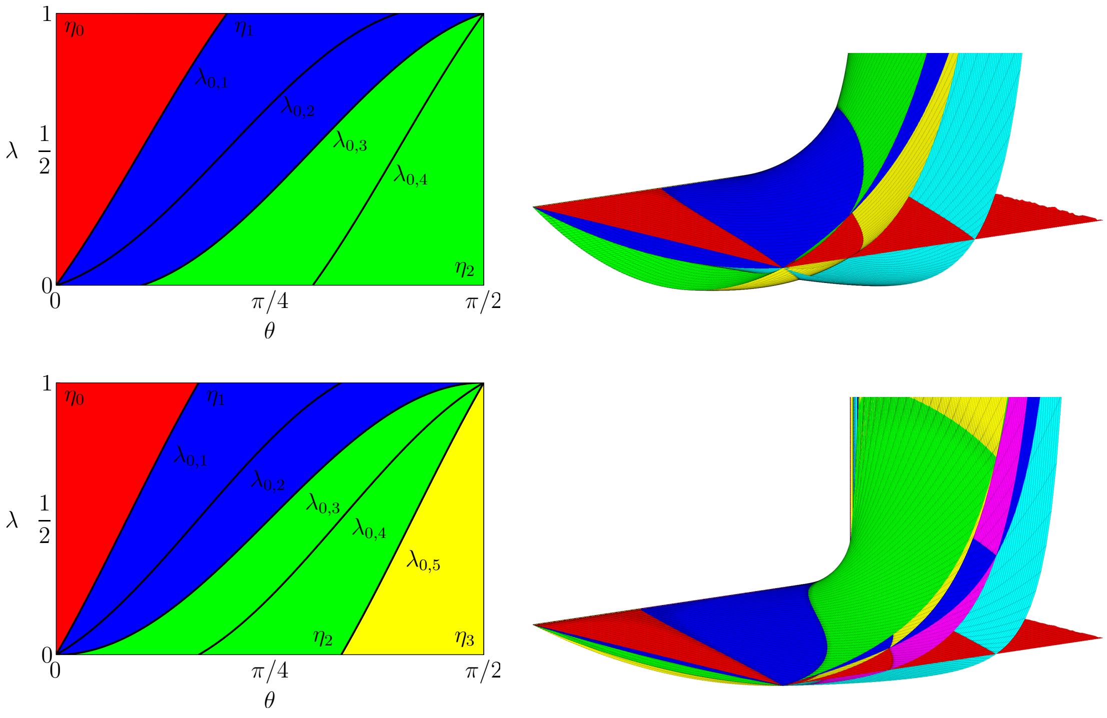


Figure 2.11: Domains of dominant eigenvalues η_k and intersection lines $\lambda_{0,\mu}$ (left) and eigenvalue meshes (right) in the case of $n = 5$ (upper) and $n = 6$ (lower)

Definition 2.5. *A partition of the parameter domain*

$$D := \{(\theta, \lambda) \,|\, \theta \in (0, \pi/2),\ \lambda \in (0, 1)\}$$

into subdomains D_k and eigenvalue intersection sets S_k is given by

$$D = \left(\bigcup_{k=0}^{\lfloor n/2 \rfloor} D_k\right) \cup \left(\bigcup_{k=0}^{\lfloor n/2 \rfloor - 1} S_k\right), \tag{2.34}$$

with

$$\begin{aligned} D_k &:= \{(\theta, \lambda) \,|\, \theta \in (0, \pi/2),\ \underline{\lambda}_k(\theta) < \lambda < \overline{\lambda}_k(\theta)\}, \\ S_k &:= \{(\theta, \lambda) \,|\, \theta \in (0, \pi/2),\ \lambda = \lambda_{k,k+1}(\theta)\} \cap D, \end{aligned}$$

where

$$\underline{\lambda}_k(\theta) := \begin{cases} 0 & \text{if } k = \lfloor n/2 \rfloor \\ \max\left(0, \lambda_{0,2k+1}(\theta)\right) & \text{otherwise} \end{cases}$$

and

$$\overline{\lambda}_k(\theta) := \begin{cases} 1 & \text{if } k = 0 \\ \min\left(1, \lambda_{0,2k-1}(\theta)\right) & \text{otherwise} \end{cases}.$$

Here, $\lfloor \cdot \rfloor$ denotes rounding towards zero.

The following lemma confirms the assumption of the pattern in which the dominant eigenvalue changes, as has been used in the partition of the parameter domain according to Definition 2.5.

Lemma 2.14. *For the parameter choice $(\theta, \lambda) \in D_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, it holds that $\eta_k > \eta_m$ for all $m \in \{0, \dots, n-1\} \setminus \{k\}$. In the case of $(\theta, \lambda) \in S_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$, it holds that $\eta_k = \eta_{k+1} > \eta_m$ for all $m \in \{0, \dots, n-1\} \setminus \{k, k+1\}$.*

Proof. In the previous section it has been shown for the eigenvalues $\eta_k^{(+)}$ of the transformation M_+ and parameters in $\widehat{D} := \{(\theta, \lambda) \mid \theta \in (0, \pi/2),\ \lambda = 1/2\}$ that for $(\theta, \lambda) \in \widehat{D} \cap D_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, it holds that $\eta_k^{(+)} > \eta_m^{(+)}$ for all $m \in \{0, \dots, n-1\} \setminus \{k\}$. Since the eigenvalue functions $\eta_k = |\eta_k^{(+)}|^2$ are continuous on D_k and according to Lemma 2.13 do not intersect each other within D_k, this dominance pattern also holds for η_k if $(\theta, \lambda) \in D_k$ as stated by the lemma.

According to Lemma 2.13 the eigenvalues η_k and η_{k+1} intersect for parameters $(\theta, \lambda) \in S_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$. Since these eigenvalues are continuous on D and dominant in the neighboring domains D_k and D_{k+1}, they are also dominant in S_k. Furthermore, since eigenvalues intersect only pairwise, there is no other η_m, $m \in \{0, \dots, n-1\} \setminus \{k, k+1\}$ with $\eta_m = \eta_{k+1} = \eta_m$, which proves the second part of Lemma 2.14. $\square$

Due to the decomposition (2.16), the dominant eigenvalue determines the behavior of the polygon $z^{(\ell)} = M^\ell z^{(0)}$ if ℓ tends to infinity. This is depicted in Fig. 2.12, which shows the resulting polygons for some iteration steps ℓ. Here, the 6-gon depicted on the lower left of Fig. 2.10 has been used as initial polygon. For each domain D_k or set S_k in the decomposition (2.34) of D one parameter pair (θ, λ) has been chosen. All polygons have been scaled with respect to their centroids by $(1/\eta_{\max})^\ell$, where $\eta_{\max} := \max_{k \in \{0, \dots, n-1\}} \eta_k$ denotes the maximal eigenvalue.

In the case of $(\theta, \lambda) \in D_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor\}$, the polygons are depicted in the same color as the associated parameter domains on the lower left of Fig. 2.11. As can be seen, for $(\theta, \lambda) \in D_0$ the polygon degenerates to its centroid (red), whereas in the case of $(\theta, \lambda) \in D_1$ it becomes a counterclockwise oriented regular 6-gon (blue). These limit figures, as well as the polygons resulting from (θ, λ) being element of D_2 and D_3 (green and yellow) can also be found in the decomposition of $z^{(0)}$ according to Fig. 2.10. For $(\theta, \lambda) \in S_k$, $k \in \{0, \dots, \lfloor n/2 \rfloor - 1\}$ (marked black), the resulting limit polygons are linear combinations of the neighboring eigenpolygons as will be stated by the following theorem.

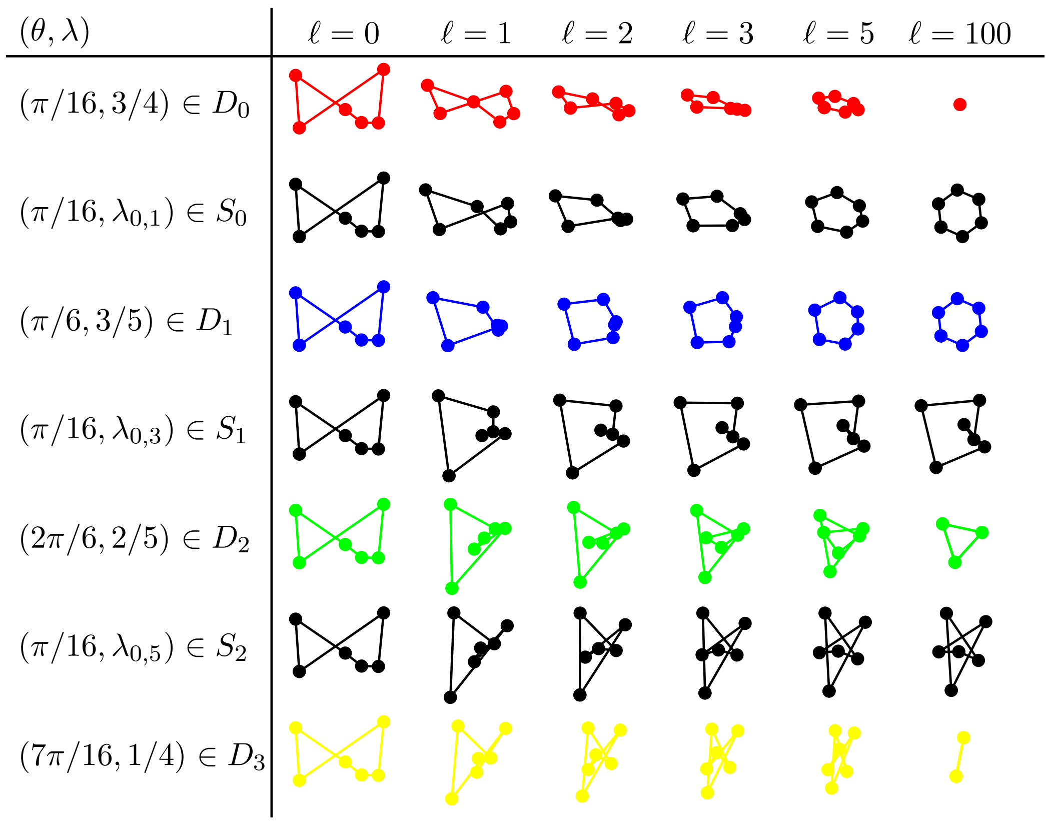


Figure 2.12: Scaled polygons $z_s^{(\ell)}$ for different iteration numbers ℓ and construction parameters $(\theta,\lambda) \in D$

Theorem 2.13. *For $\ell \in \mathbb{N}_0$ let*

$$z_s^{(\ell)} := v_0 + \frac{1}{\eta_{\max}^{\ell}}(z^{(\ell)} - v_0) = v_0 + \sum_{k=1}^{n-1} \left(\frac{\eta_k}{\eta_{\max}}\right)^{\ell} v_k\,, \tag{2.35}$$

denote the polygon $z^{(\ell)} = M^{\ell} z^{(0)}$ scaled with respect to the centroid v_0 by the inverse of the ℓ-th power of the maximal eigenvalue. Then $z^{(\ell)}$ tends to

$$z_s^{(\infty)} := \begin{cases} v_0 & \text{if } (\theta,\lambda) \in D_0 \\ v_0 + v_k & \text{if } (\theta,\lambda) \in D_k, k \in \{1,\dots,\lfloor n/2 \rfloor\} \\ v_0 + v_1 & \text{if } (\theta,\lambda) \in S_0 \\ v_0 + v_k + v_{k+1} & \text{if } (\theta,\lambda) \in S_k, k \in \{1,\dots,\lfloor n/2 \rfloor - 1\} \end{cases}, \tag{2.36}$$

that is $z_s^{(\infty)} = \lim_{\ell\to\infty} z_s^{(\ell)}$.

Proof. The representation (2.36) will be shown by analyzing the eigenvalue quotients $\rho_k := \eta_k/\eta_{\max} \in [0,1]$, $k \in \{1,\dots,n-1\}$, in the sum on the right hand side of (2.35), which is implied by (2.16) and $\eta_0 = 1$. In the case of $\eta_{\max} = \eta_k$ it holds that $\rho_k^\ell = 1$. Hence, the associated eigenpolygon v_k is kept unscaled in the sum. Otherwise $\rho_k < 1$ implies that $\rho_k^\ell v_k$ tends to the zero vector if ℓ tends to infinity. Hence, the limit of the sum on the right hand side of (2.35) is given by the sum of the eigenpolygons belonging to the maximal eigenvalues. The latter are given by Lemma 2.14. □

It should be noticed that in the case $k = 0$ the maximal eigenvalue is given by $\eta_0 = 1$. Hence for $(\theta, \lambda) \in D_0 \cup S_0$ it holds that $z_s^{(\ell)} = z^{(\ell)}$. In particular the unscaled sequence of polygons $z^{(\ell)}$ degenerates for parameter pairs in D_0, converges to a bounded polygon in S_0 and diverges otherwise. Nevertheless, the transformation has a regularizing effect, since the behavior depends on the dominant term.

Theorem 2.13 also demonstrates the advantage of combining the basic transformations M_- and M_+. Since all eigenvalues of $M = M_+ M_-$ are positive, each step of the transformation has only a scaling effect with respect to the eigenpolygons and their associated eigenvalues. Hence scaling with $(1/\eta_{\max})^\ell$ leads to a converging sequence of polygons $z_s^{(\ell)}$. In contrast, the eigenvalues of M_+ and M_- respectively are complex valued which implies that one step of the transformation not only scales each eigenpolygon according to the absolute value of its associated eigenvalue, but also rotates it by an angle given by the argument of the eigenvalue. Nevertheless, as has been shown in Section 2.2 for the special case $\lambda = 1/2$, there exist $2n$ converging subsequences where the limit polygons for odd and even ℓ differ only in a cyclic shift of indices, but not in geometry.

For the presented combined transformation, special cases occur if either one or more of the eigenpolygons in the representation (2.36) of the limit polygon are zero vectors, that is the according coefficients of the representation of the initial polygon $z^{(0)}$ with respect to the Fourier polygons are zero, or transformation parameters are used for which an eigenvalue η_k becomes zero.

For example, let $z^{(0)}$ be a clockwise oriented regular n-gon, $\lambda = 1/2$, and $\theta = (n-2)\pi/(2n)$ be half the interior angle of the regular n-gon. Due to symmetry reasons, the apices of the isosceles triangles constructed in the first substep G_- all coincide with the centroid of $z^{(0)}$ resulting in $z^{(\ell)} = v_0$ for $\ell \in \mathbb{N}$. This can also be seen by the eigenvalues $\eta_k = \csc^2(\pi/n)\sin^2((1+k)\pi/n)$ of M. Since $z^{(0)}$ is a linear combination of the first and last column of the Fourier matrix F, one step of the transformation eliminates the associated eigenpolygon b_{n-1} since $\eta_{n-1} = 0$.

It should also be noticed that the eigenpolygons b_k, $k \in \{\lfloor n/2 \rfloor + 1, \ldots, n\}$, are not contained in the limit polygons given by (2.36). However, due to symmetry reasons these eigenpolygons occur as limit polygons if the similar triangles are constructed to the left of each side.

Chapter 3

Regularizing transformations for polyhedral finite elements

In this chapter regularizing transformations for the most common volumetric finite element types, that is tetrahedra, hexahedra, pyramids, and prisms, are presented. Their regularizing effect is shown numerically, whereby the degree of regularity is measured by the mean ratio quality criterion, which is introduced first.

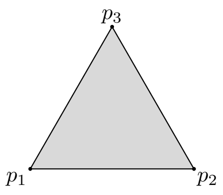
3.1 The mean ratio quality criterion

In order to assess the regularity of mesh elements, the well known mean ratio quality criterion is used, which has been developed in the context of optimization-based smoothing methods [20, 22]. It is based on measuring the distance of an arbitrary simplex in $\mathbb{R}^n$ from a corresponding reference simplex. For completeness, the following description will not only cover the volumetric elements under consideration, but also triangular and quadrilateral planar elements, for which regularizing transformations have been analyzed in the previous chapter.

Let $E := (p_1, \dots, p_{|E|})$ denote an element with $|E|$ nodes, which can be either triangular or quadrilateral in $\mathbb{R}^2$ or tetrahedral, hexahedral, pyramidal, or prismatic in $\mathbb{R}^3$. Except for the apex of the pyramidal element, each node $p_i \in \mathbb{R}^n$, with $n \in \{2, 3\}$, of any element is edge-connected to n other nodes p_j of the same element. Thus each node and its edge-connected neighbors define a simplex in $\mathbb{R}^n$, which will be called node simplex.

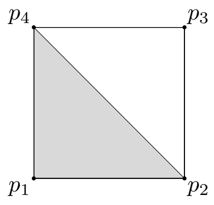
Fig. 3.1 gives examples of such simplices associated to p_1, which are marked gray for all element types under consideration. Here, the given elements are regular and the first two nodes have been set to $p_1 = (0, \dots, 0)$ and $p_2 = (1, 0, \dots)$, respectively. As a prerequisite of the definition of the mean ratio quality criterion,

(a) Triangle



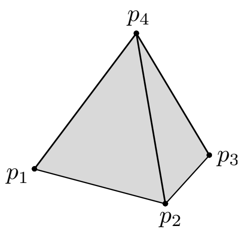
$$W = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{pmatrix}$$

(b) Quadrilateral



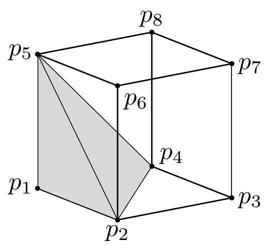
$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(c) Tetrahedron



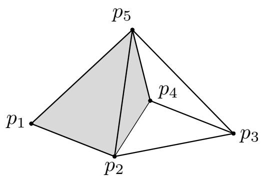
$$W = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{6} \\ 0 & 0 & \frac{\sqrt{2}}{\sqrt{3}} \end{pmatrix}$$

(d) Hexahedron



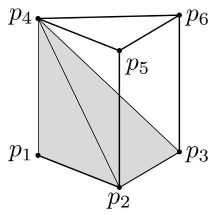
$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(e) Pyramid



$$W = \begin{pmatrix} 1 & 0 & \frac{1}{2} \\ 0 & 1 & \frac{1}{2} \\ 0 & 0 & \frac{\sqrt{2}}{2} \end{pmatrix}$$

(f) Prism



$$W = \begin{pmatrix} 1 & \frac{1}{2} & 0 \\ 0 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 3.1: Regular elements with reference simplices marked gray and associated weight matrices W

specific element validity conditions have to be stated first. Element validity will also be assessed based on the node simplices, which will be rendered more precisely first. In the following, an element of a specific type will be denoted as E^{tri}, E^{quad}, E^{tet}, E^{hex}, E^{pyr}, E^{pri}, respectively. The node indices of the node simplices for the corresponding element type are given by the tuples

$$\begin{aligned}
N^{\mathrm{tri}} &:= \big((1,2,3)\big),\\
N^{\mathrm{quad}} &:= \big((1,2,4),(2,3,1),(3,4,2),(4,1,3)\big),\\
N^{\mathrm{tet}} &:= \big((1,2,3,4)\big),\\
N^{\mathrm{hex}} &:= \big((1,4,5,2),(2,1,6,3),(3,2,7,4),(4,3,8,1),(5,8,6,1),(6,5,7,2),\\
&\qquad (7,6,8,3),(8,7,5,4)\big),\\
N^{\mathrm{pyr}} &:= \big((1,2,4,5),(2,3,1,5),(3,4,2,5),(4,1,3,5)\big),\\
N^{\mathrm{pri}} &:= \big((1,2,3,4),(2,3,1,5),(3,1,2,6),(4,6,5,1),(5,4,6,2),(6,5,4,3)\big).
\end{aligned}$$

In this representation, the kth element of $N \in \{N^{\mathrm{tri}}, N^{\mathrm{quad}}, N^{\mathrm{tet}}, N^{\mathrm{hex}}, N^{\mathrm{pyr}}, N^{\mathrm{pri}}\}$ represents the $n+1$ node indices $N_{k,i}$, $i \in \{1,\dots,n+1\}$, of the node simplex associated with p_k. The number of node simplices, i.e. the number of elements in the tuple N, is denoted as $|N|$. It should be noticed that in the case of E^{tri} and E^{tet} each of the $n+1$ node simplices represents the element itself. Hence it suffices to use only the node simplex associated with p_1 resulting in $|N^{\mathrm{tri}}| = |N^{\mathrm{tet}}| = 1$. Furthermore, the apex p_5 of the pyramidal element is the only node with an incident edge number being not equal to n. However, this node is omitted, since only the four base node tetrahedra will be used in order to define the mean ratio quality number of a pyramidal element. Consequently, for the triangular, tetrahedral as well as the pyramidal element it holds that $|E| \neq |N|$.

Definition 3.1. *Let $E \in \{E^{\mathrm{tri}}, E^{\mathrm{quad}}, E^{\mathrm{tet}}, E^{\mathrm{hex}}, E^{\mathrm{pyr}}, E^{\mathrm{pri}}\}$ denote an arbitrary element with its associated node simplices indices tuple given by $N \in \{N^{\mathrm{tri}}, N^{\mathrm{quad}}, N^{\mathrm{tet}}, N^{\mathrm{hex}}, N^{\mathrm{pyr}}, N^{\mathrm{pri}}\}$. For each node p_k, $k \in \{1,\dots,|N|\}$, let*

$$T_k := \big(p_{N_{k,1}}, \dots, p_{N_{k,n+1}}\big) \tag{3.1}$$

denote associated the node simplex, and

$$D(T_k) := \big(p_{N_{k,2}} - p_{N_{k,1}}, \dots, p_{N_{k,n+1}} - p_{N_{k,1}}\big)$$

the $(n \times n)$-matrix of its spanning edge vectors. The element is called valid if $\det(D(T_k)) > 0$ *holds for all $k \in \{1,\dots,|N|\}$.*

The given validity criterion excludes in particular specific cases of degenerated elements, like for example tetrahedra with collinear nodes, for which the geometric element transformation given later will not be defined. However, the prerequisites of the mean ratio quality criterion or additional prerequisites imposed by applications like the finite element method requiring a positive interior Jacobian are stronger than those of the geometric transformation in order to be well defined.

In the following a definition of the mean ratio quality criterion is given, which is based on measuring the deviation of simplices in an arbitrary valid element from their counterpart in an ideal, i.e. regular element, which is represented by a target matrix W [22].

Definition 3.2. *Let $E \in \{E^{\text{tri}}, E^{\text{quad}}, E^{\text{tet}}, E^{\text{hex}}, E^{\text{pyr}}, E^{\text{pri}}\}$ denote an arbitrary valid element and $N \in \{N^{\text{tri}}, N^{\text{quad}}, N^{\text{tet}}, N^{\text{hex}}, N^{\text{pyr}}, N^{\text{pri}}\}$ its associated indices tuple of node tetrahedra T_k according to* (3.1). *The mean ratio quality number of the element is given by*

$$q(E) := \frac{1}{|N|} \sum_{k=1}^{|N|} \frac{3 \det(S_k)^{2/3}}{\operatorname{trace}(S_k^{\mathrm{t}} S_k)}, \quad S_k := D(T_k) W^{-1}, \tag{3.2}$$

with the element type dependent target simplex weight matrix W given in Fig. 3.1. Here, the columns of W represent the spanning edge vectors of a node tetrahedron in the associated regular element.

It holds that $q(E) \in [0, 1]$ with small quality numbers representing low quality elements and $q(E) = 1$ representing ideal regular elements. In the case of pyramidal elements only the node tetrahedra associated to the four base nodes are used for quality computation, which is in accordance to [137]. Furthermore, In the case of non-simplicial elements, the associated difference matrices W represent non-regular simplices. It should also be noticed that the denominator in (3.2) represents the squared Frobenius norm of the matrix S_k.

3.2 An opposite face normals-based transformation for tetrahedra

In the following a simple and efficient regularizing transformation for tetrahedral elements will be given and analyzed. It is based on deriving new node positions with the aid of the normals of opposing element faces according to the following definition.

Definition 3.3. *Let $E = (p_1, \dots, p_4)$ denote a valid tetrahedral element with the four inside oriented opposite face normals*

$$\begin{aligned} n_1 &:= (p_4 - p_2) \times (p_3 - p_2), \\ n_2 &:= (p_4 - p_3) \times (p_1 - p_3), \\ n_3 &:= (p_2 - p_4) \times (p_1 - p_4), \\ n_4 &:= (p_2 - p_1) \times (p_3 - p_1). \end{aligned}$$

The new nodes p'_i, $i \in \{1, \dots, 4\}$, of the new element E' derived by the opposite face normals transformation are given by

$$p'_i := p_i + \sigma \frac{1}{\sqrt{|n_i|}} n_i \,, \tag{3.3}$$

where $\sigma \in \mathbb{R}_0^+$ denotes a given scaling factor.

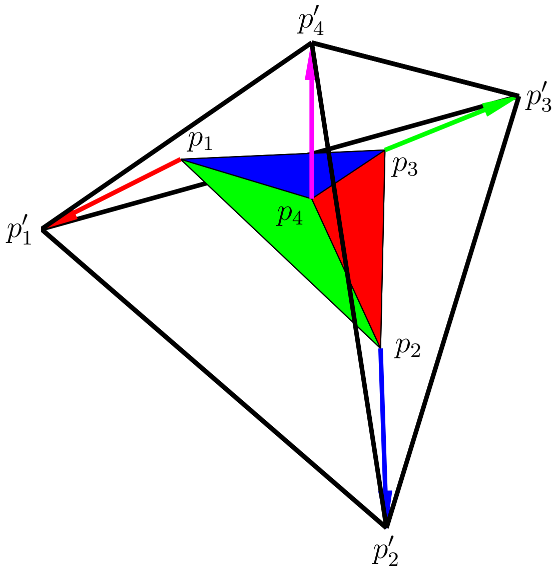


Figure 3.2: Transformation of a tetrahedron using $\sigma = 1$

An initial tetrahedron E and its transformed counterpart E' using $\sigma = 1$ are depicted exemplarily in Fig. 3.2. In this, associated faces and normals are marked by the same color. The edges of the resulting tetrahedron E' are indicated by thick black lines.

Due to the orientation of the normals, the transformation enlarges the tetrahedron. Thereby, the magnification is scalable by the factor σ. In the case of $\sigma = 0$ it holds that $E = E'$. Furthermore, since the normals n_i are scaled by $1/\sqrt{|n_i|}$, the transformation is scale invariant, i.e. for $s > 0$ it holds that $(sE)' = sE'$.

In the case of the example depicted in Fig. 3.2, the mean ratio numbers are given by $q(E) = 0.6795$ and $q(E') = 0.9937$ for the initial and the transformed tetrahedron respectively. The resulting tetrahedra obtained by three consecutively applied transformation steps using $\sigma = 1/10$ as well as the resulting mean ratio values and volumes are depicted in Fig. 3.3.

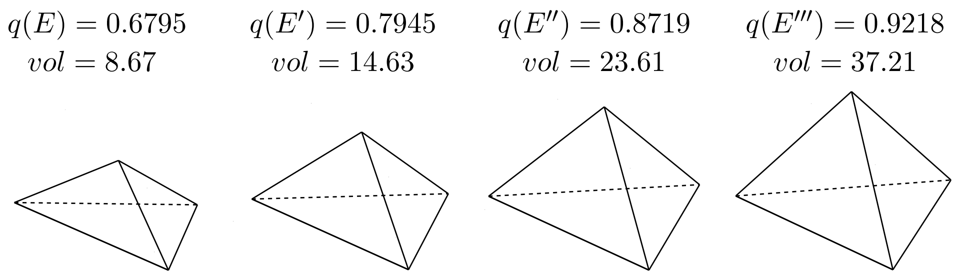


Figure 3.3: Sequence of consecutive tetrahedron transformations using $\sigma = 1/10$

The linear element transformations for planar polygonal elements introduced in Chapter 2 are based on circulant matrices. Hence, limit polygons can be easily determined by analyzing the associated eigenvectors. Since the transformation given by (3.3) is nonlinear and iteratively applying the transformation leads to complex expressions, proving the regularizing effect is not that simple. Nevertheless, it has been substantiated by the following numerical test.

For a given random initial tetrahedron and a fixed value of σ, the transformation has been iteratively applied until a mean ratio number $\geq (1 - 10^{-6})$ has been achieved. In order to investigate the influence of the transformation parameter σ and the initial geometry on the convergence of this transformation cycle, the resulting iteration number has been computed for a broad range of input data. To be precise, a set of 100,000 different initial tetrahedra with random nodes $p_i \in [-1, 1]^3$ and a set of 101 different values of σ equally distributed in $[0, 2]$ have been used. By performing the transformation cycle for each of the 10,100,000 elements of the Cartesian product of these two sets, a strong evidence for the general convergence of the successively applied transformation resulting in regular tetrahedra has been derived. Results of this test with respect to the transformation parameter σ are depicted in Fig. 3.4.

For each specific choice of σ, the upper and lower bound of the corridor marked gray represent the maximal and minimal iteration number of all 100,000 regularization cycles performed. The mean iteration number of all cycles with respect to σ is marked by a thick blue line. The comparatively narrow corridor indicates that the iteration numbers vary only slightly for a given scaling factor σ. That is, the mean ratio number and hence the geometry of the initial tetrahedron has only small influence on the convergence of the transformation. In contrast, the choice

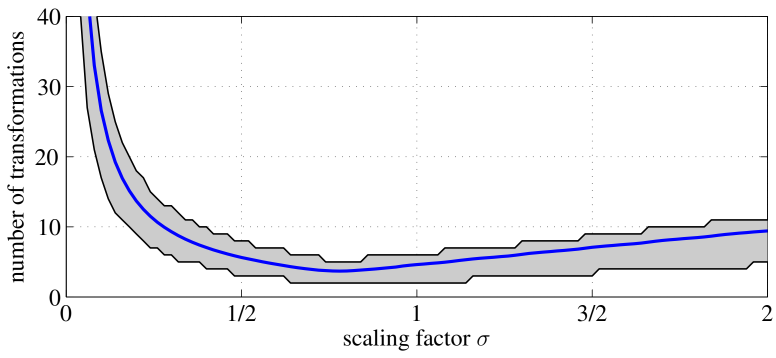


Figure 3.4: Number of transformations depending on the parameter σ

of the scaling factor σ has a significant impact on the resulting iteration number. Whereas the mean iteration number grows exponentially if σ tends to zero, it grows only linearly if σ becomes large. This can be used in order to control the regularization speed by a quality dependent choice of the scaling factor in the geometric element transformation method as described in the following chapter.

The mean iteration number becomes minimal in the case of $\sigma \approx 0.78$ and amounts to 3.69. The minimum of the graph suggests that there might be a specific choice of σ resulting in an analogon to Napoleon's theorem for plane triangles providing a geometric transformation, which regularizes an arbitrary element within one transformation step. This does not hold for the opposite face normals transformation for tetrahedra given by Definition (3.3). That is, there is no specific choice of σ, for which applying the transformation exactly once to any arbitrary tetrahedron results in a regular one. This can be shown by the following counterexample.

Let E be the tetrahedron with the nodes $p_1 = (0,0,0)^{\mathrm{t}}$, $p_2 = (1,0,0)^{\mathrm{t}}$, $p_3 = (0,2,0)^{\mathrm{t}}$, $p_4 = (0,0,3)^{\mathrm{t}}$. According to (3.3), the nodes of the transformed tetrahedron E' are given by $p_1' = -\frac{\sigma}{\sqrt{7}}(6,3,2)^{\mathrm{t}}$, $p_2' = (1+\sqrt{6}\sigma,0,0)^{\mathrm{t}}$, $p_3' = (0,2+\sqrt{3}\sigma,0)^{\mathrm{t}}$, $p_4' = (0,0,3+\sqrt{2}\sigma)^{\mathrm{t}}$ using an arbitrary scaling factor $\sigma \in \mathbb{R}_0^+$. In order to be regular, all edge lengths of the transformed tetrahedron have to be equal. Since the equation $|p_2' - p_3'| = |p_2' - p_4'|$ has the only valid solution $\sigma = \sqrt{2}+\sqrt{3}$, and in contradiction $|p_2' - p_3'| = |p_3' - p_4'|$ implies $\sigma = \frac{1}{2}(\sqrt{2}+\sqrt{6})$, there is no $\sigma \in \mathbb{R}_0^+$ for which the tetrahedron E' obtained by one step of the transformation is regular. However, E can be regularized up to any given tolerance by several steps of the transformation.

Concerning smoothing applications, regularization within one step is not desirable, since usually this changes geometry to rapidly and might invalidate neighboring elements. Hence, by reliably transforming arbitrary elements successively into regular ones with a regularization speed easily controllable by the parameter σ, the transformation (3.3) is perfectly suited for the GETMe smoothing approach described in the following chapter.

It should also be noted, that the transformation given by (3.3) does not preserve the centroid of the tetrahedron, that is $\frac{1}{4}\sum_{i=1}^{4} p_i \neq \frac{1}{4}\sum_{i=1}^{4} p_i'$, since the normals n_i are scaled by $1/\sqrt{|n_i|}$ to ensure the scale invariance of the transformation. In contrast, using the unscaled normals n_i results in a regularizing transformation which preserves the centroid, but is not scale invariant.

3.3 Dual element-based transformations

In the majority of cases mixed volume meshes are assembled of tetrahedra, hexahedra, quadrilateral pyramids, or triangular prisms. Here, a generalization of the opposite face normals transformation for all element types is not straightforward. Therefore, an alternative approach based on dual elements is given in the following.

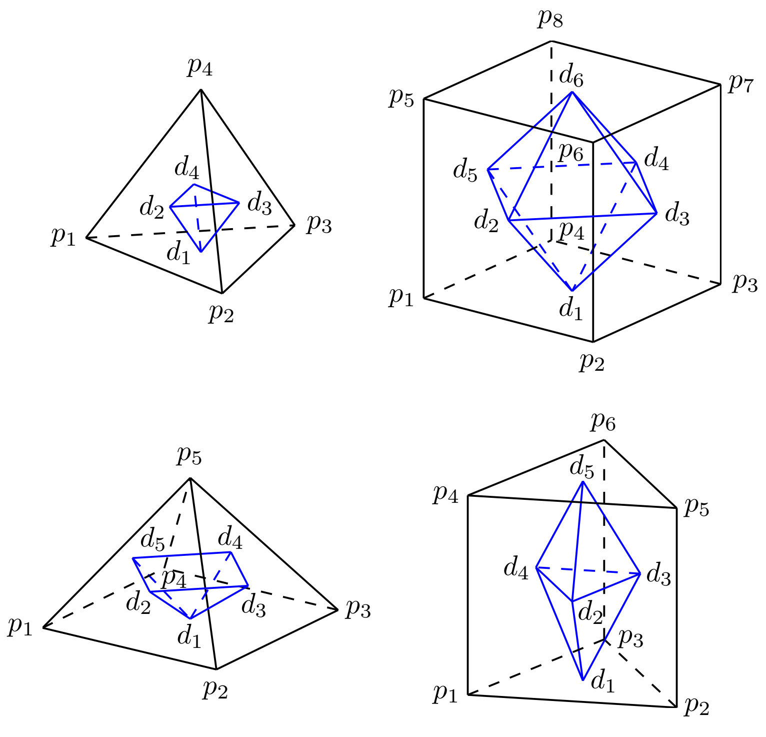


Figure 3.5: Node numbering schemes for all element types and their duals

Node numbering schemes for the element types under consideration as well as their dual elements are depicted in Fig. 3.5. The dual elements, marked blue, consist of the nodes d_k, which will be defined on the basis of the element faces.

The node index tuples of the latter are given by

$$\begin{aligned}
F^{\text{tet}} &:= \big((1,2,3),(1,2,4),(2,3,4),(3,1,4)\big),\\
F^{\text{hex}} &:= \big((1,2,3,4),(1,2,6,5),(2,3,7,6),(3,4,8,7),(4,1,5,8),(5,8,7,6)\big),\\
F^{\text{pyr}} &:= \big((1,2,3,4),(1,2,5),(2,3,5),(3,4,5),(4,1,5)\big),\\
F^{\text{pri}} &:= \big((1,2,3),(1,2,5,4),(2,3,6,5),(3,1,4,6),(4,6,5)\big)\,.
\end{aligned}$$

That is, the kth element of a faces tuple $F \in \{F^{\text{tet}}, F^{\text{hex}}, F^{\text{pyr}}, F^{\text{pri}}\}$ represents the tuple of node indices defining the kth face. The index of the ith node of the kth face will be denoted as $F_{k,i}$. The nodes d_k of a dual element are defined as the arithmetic mean of the not necessarily coplanar face nodes, i.e.

$$d_k := \frac{1}{|F_k|}\sum_{i=1}^{|F_k|} p_{F_{k,i}}\,, \quad k \in \{1,\ldots,|F|\}\,, \tag{3.4}$$

where $|F_k|$ denotes the number of nodes in the kth face and $|F|$ the number of element faces.

Whereas the tetrahedron and the pyramid are self-dual, the dual elements of the hexahedron and the prism are an octahedron and a triangular dipyramid, respectively. In addition, among all dual element faces, the base quadrilateral of the dual pyramid is the only non-triangular face. Node indices of all dual element faces are given by

$$\begin{aligned}
\overline{F}^{\text{tet}} &:= \big((1,2,4),(1,3,2),(1,4,3),(2,3,4)\big),\\
\overline{F}^{\text{hex}} &:= \big((1,2,5),(1,3,2),(1,4,3),(1,5,4),(6,5,2),(6,2,3),\\
&\qquad (6,3,4),(6,4,5)\big),\\
\overline{F}^{\text{pyr}} &:= \big((1,2,5),(1,3,2),(1,4,3),(1,5,4),(2,3,4,5)\big),\\
\overline{F}^{\text{pri}} &:= \big((1,2,4),(1,3,2),(1,4,3),(5,4,2),(5,2,3),(5,3,4)\big).
\end{aligned}$$

The regularizing transformation scheme is based on erecting scaled versions of the dual element faces normals on suitable base points. Here, the outward directed face normals are defined as

$$n_k := \begin{cases} (d_{\overline{F}_{k,2}} - d_{\overline{F}_{k,1}}) \times (d_{\overline{F}_{k,3}} - d_{\overline{F}_{k,1}}) & \text{if } |\overline{F}_k| = 3\,,\\ \frac{1}{2}(d_{\overline{F}_{k,3}} - d_{\overline{F}_{k,1}}) \times (d_{\overline{F}_{k,4}} - d_{\overline{F}_{k,2}}) & \text{if } |\overline{F}_k| = 4\,, \end{cases} \tag{3.5}$$

where $k \in \{1,\ldots,|\overline{F}|\}$ and $\overline{F}$ denoting a dual element faces tuple taken from $\{\overline{F}^{\text{tet}}, \overline{F}^{\text{hex}}, \overline{F}^{\text{pyr}}, \overline{F}^{\text{pri}}\}$. That is, in the case of triangular faces, the normal is determined by the cross product of two vectors spanning the face. In the case of the quadrilateral base of the pyramid, the normal is defined as the cross product of the two diagonals of the quadrilateral scaled by 1/2. The base points, on which the normals will be erected, are defined with the aid of the dual element faces centroids

$$c_k := \frac{1}{|\overline{F}_k|} \sum_{i=1}^{|\overline{F}_k|} d_{\overline{F}_{k,i}}\,, \quad k \in \{1, \dots, |\overline{F}|\}\,. \tag{3.6}$$

For the triangular faces of the duals of non-Platonic elements, i.e. pyramids and prisms, the point

$$a_k(\tau) := (1-\tau) d_{\overline{F}_{k,1}} + \tau \frac{1}{2}\left(d_{\overline{F}_{k,2}} + d_{\overline{F}_{k,3}}\right),$$

with $\tau \in \mathbb{R}$ is used. It is located on the line connecting the triangular face apex and the midpoint of the opposing side. The following definition specifies the choice of τ with respect to a normals scaling factor $\sigma > 0$. The latter will be used in order to control the regularizing effect of the single element transformation.

Definition 3.4. *Let $E \in \{E^{\text{tet}}, E^{\text{hex}}, E^{\text{pyr}}, E^{\text{pri}}\}$ denote an arbitrary element with $|E|$ nodes p_k and dual element face normals n_k. Furthermore, let $\sigma \in \mathbb{R}^+$ denote an arbitrary normals scaling factor. The nodes p'_k of the transformed element $E' = (p'_1, \dots, p'_{|E|})$ are given by*

$$p'_k := b_k + \frac{\sigma}{\sqrt{|n_k|}} n_k, \quad k \in \{1, \dots, |E|\}, \tag{3.7}$$

with element type dependent base points

$$b_k := \begin{cases} c_k & \text{if } E \in \left\{E^{\text{tet}}, E^{\text{hex}}\right\} \vee \left(E = E^{\text{pyr}} \wedge k = 5\right), \\ a_k(\tau),\ \tau := \dfrac{1}{2} + \sigma & \text{if } E = E^{\text{pyr}} \wedge 1 \le k \le 4, \\ a_k(\tau),\ \tau := \dfrac{4}{5}\left(1 - \dfrac{\sqrt{2}\sigma}{\sqrt[4]{39}}\right) & \text{if } E = E^{\text{pri}}. \end{cases} \tag{3.8}$$

A basic prerequisite for regularizing element transformations is that regular elements are transformed into regular elements. Due to symmetry reasons, this holds for tetrahedral and hexahedral elements for the specific choice of $b_k = c_k$ and arbitrary $\sigma > 0$. However, this does not hold for the non-Platonic pyramidal and prismatic elements. Hence, for these elements an additional degree of freedom had to be included. It is represented by the σ-dependent parameter τ defining the position of the base points b_k of triangular faces chosen on the line connecting the apex and the midpoint of the opposing side. The specific choice of τ given

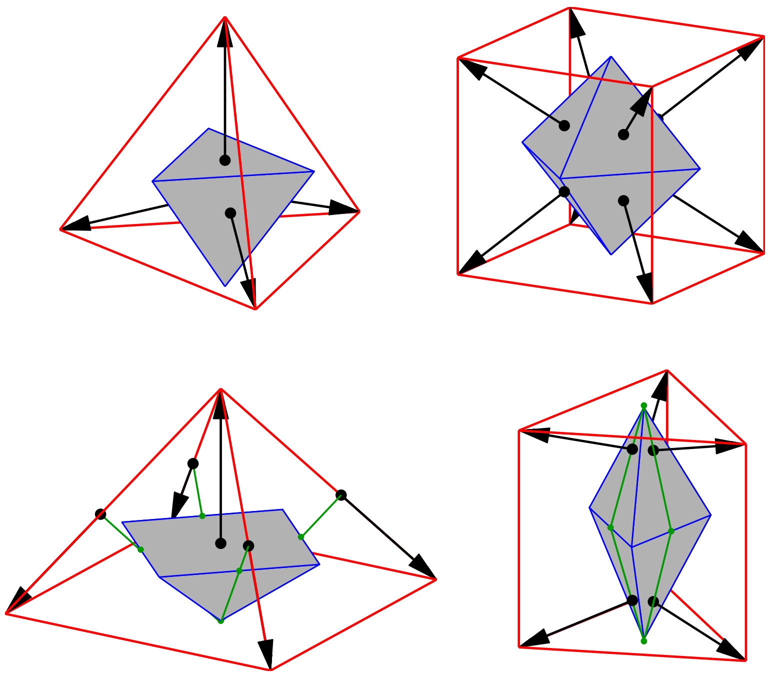


Figure 3.6: Transformed element construction by erecting normals on the dual element

in (3.8) has been determined by the basic regularity prerequisite. That is, by solving an equation based on the equality of transformed element edge lengths.

The construction of the transformed element is depicted in Fig. 3.6 for the specific choice $\sigma = 1$. Here, the faces of the dual elements are marked gray with blue edges. The base points b_k are indicated by black markers, the normals n_k by black arrows, and the resulting transformed elements E' red. In the case of the pyramidal and prismatic element, the line connecting the triangular face apex with the midpoint of the opposite side is marked green. Here, the resulting base point b_k may lie outside the associated dual element face as can be seen in the case of the pyramidal element.

In the following, basic properties of the dual elements-based transformation will be analyzed.

Lemma 3.1. *The transformation given by Definition 3.4 is invariant with respect to translation, rotation, and scaling for all element types.*

Proof. It has to be shown that for arbitrary scaling factors $\zeta > 0$, rotation matrices R (i.e. matrices with $\det R = 1$ and $R^{-1} = R^{\mathrm{t}}$) and translation vectors $t \in \mathbb{R}^3$ it holds that $(\zeta R p_k + t)' = \zeta R p_k' + t$. This will be accomplished by applying the geometric transformation given by Definition 3.4 to the nodes $\tilde{p}_k := \zeta R p_k + t$ of an element $E \in \left\{E^{\mathrm{tet}}, E^{\mathrm{hex}}, E^{\mathrm{pyr}}, E^{\mathrm{pri}}\right\}$ with the corresponding faces tuple F.

According to (3.4) the nodes $\tilde{d}_k$ of the dual element associated to the element with nodes $\tilde{p}_k$ are given by

$$\begin{aligned}\tilde{d}_k &:= \frac{1}{|F_k|}\sum_{i=1}^{|F_k|}\tilde{p}_{F_{k,i}} = \frac{1}{|F_k|}\sum_{i=1}^{|F_k|}(\zeta R p_{F_{k,i}} + t) = \zeta R\left(\frac{1}{|F_k|}\sum_{i=1}^{|F_k|} p_{F_{k,i}}\right) + t \\ &= \zeta R d_k + t\,,\end{aligned}$$

with $k \in \{1, \ldots, |F|\}$. Using (3.6) it can be shown analogously that the centroid $\tilde{c}_k$ of the kth dual element face with node indices given by $\overline{F}_k$ can be written as

$$\tilde{c}_k := \frac{1}{|\overline{F}_k|}\sum_{i=1}^{|\overline{F}_k|}\tilde{d}_{\overline{F}_{k,i}} = \zeta R c_k + t, \qquad \text{for } k \in \{1, \ldots, |\overline{F}|\}.$$

According to (3.8), the base node $\tilde{b}_k$ is either the centroid $\tilde{c}_k$ of the kth dual element face or given by the linear combination

$$\begin{aligned}\tilde{a}_k(\tau) &:= (1-\tau)\tilde{d}_{\overline{F}_{k,1}} + \frac{\tau}{2}\left(\tilde{d}_{\overline{F}_{k,2}} + \tilde{d}_{\overline{F}_{k,3}}\right) \\ &= (1-\tau)(\zeta R d_{\overline{F}_{k,1}} + t) + \frac{\tau}{2}\left(\zeta R(d_{\overline{F}_{k,2}} + d_{\overline{F}_{k,3}}) + 2t\right) \\ &= \zeta R\left((1-\tau)d_{\overline{F}_{k,1}} + \frac{\tau}{2}\left(d_{\overline{F}_{k,2}} + d_{\overline{F}_{k,3}}\right)\right) + t = \zeta R a_k(\tau) + t\,,\end{aligned}$$

which implies $\tilde{b}_k = \zeta R b_k + t$ for $k \in \{1, \ldots, |E|\}$. That is, the normal base nodes of the scaled, rotated, and translated element can be obtained by scaling, rotating, and translating the normal base nodes of the initial element.

For the normals $\tilde{n}_k$ of triangular dual element faces with nodes $\tilde{d}_{\overline{F}_{k,i}}$ it holds that

$$\begin{aligned}\tilde{n}_k &:= \left(\tilde{d}_{\overline{F}_{k,2}} - \tilde{d}_{\overline{F}_{k,1}}\right) \times \left(\tilde{d}_{\overline{F}_{k,3}} - \tilde{d}_{\overline{F}_{k,1}}\right) \\ &= \left(\zeta R(d_{\overline{F}_{k,2}} - d_{\overline{F}_{k,1}})\right) \times \left(\zeta R(d_{\overline{F}_{k,3}} - d_{\overline{F}_{k,1}})\right) \\ &= \zeta^2 \det(R)\left(R^{-1}\right)^{\mathrm{t}}\left[(d_{\overline{F}_{k,2}} - d_{\overline{F}_{k,1}}) \times (d_{\overline{F}_{k,3}} - d_{\overline{F}_{k,1}})\right] = \zeta^2 R n_k\,.\end{aligned}$$

For normals of quadrilateral faces $\tilde{n}_k = \zeta^2 R n_k$ follows analogously.

These representations of $\tilde{b}_k$ and $\tilde{n}_k$ finally imply

$$\begin{aligned}(\zeta R p_k + t)' &= \tilde{p}'_k = \tilde{b}_k + \frac{\sigma}{\sqrt{|\tilde{n}_k|}}\tilde{n}_k = \zeta R b_k + t + \frac{\sigma}{\zeta\sqrt{|n_k|}}\zeta^2 R n_k \\ &= \zeta R\left(b_k + \frac{\sigma}{\sqrt{|n_k|}}n_k\right) + t = \zeta R p'_k + t\,,\end{aligned}$$

thus proving the invariance property of the transformation as stated by the lemma. □

The last step in the proof of Lemma 3.1 also gives the motivation for the choice of the normalization factor $1/\sqrt{|n_k|}$ in the definition (3.7) of p'_k, since it ensures the scale invariance of the regularizing element transformation.

For an arbitrary element $E \in \{E^{\text{tet}}, E^{\text{hex}}, E^{\text{pyr}}, E^{\text{pri}}\}$ with $|E|$ nodes p_k let

$$q := \frac{1}{|E|} \sum_{k=1}^{|E|} p_k \tag{3.9}$$

denote the centroid of the initial element. Furthermore, let

$$q' := \frac{1}{|E|} \sum_{k=1}^{|E|} p'_k = \underbrace{\frac{1}{|E|} \sum_{k=1}^{|E|} b_k}_{=:q'_b} + \underbrace{\frac{\sigma}{|E|} \sum_{k=1}^{|E|} \frac{1}{\sqrt{|n_k|}} n_k}_{=:q'_n} \tag{3.10}$$

denote the centroid of the transformed element and its partition into the mean base node q'_b as well as the mean normal q'_n. The following lemma considers the relation between the centroids of initial and transformed elements.

Lemma 3.2. *For $E \in \{E^{\text{tet}}, E^{\text{hex}}, E^{\text{pri}}\}$ it holds that $q = q'_b$. In addition, for $E = E^{\text{hex}}$ it holds that $q = q'$, i.e. the transformation preserves the centroid of hexahedral elements.*

Proof. The hexahedral case will be shown first by evaluating the two sums q'_b and q'_n according to (3.10) separately. For q'_b successively substituting $b_k := c_k$ by the representations of the octahedron nodes, centroids and normals yields

$$\frac{1}{8}\sum_{k=1}^{8} c_k = \frac{1}{8}\sum_{k=1}^{8}\left(\frac{1}{3}\sum_{i=1}^{3}\left(\frac{1}{4}\sum_{j=1}^{4} p_{F^{\text{hex}}_{F^{\text{hex}}_{k,i},j}}\right)\right) = \frac{1}{8\cdot 3\cdot 4}\sum_{k=1}^{8} 3\cdot 4\cdot p_k = \frac{1}{8}\sum_{k=1}^{8} p_k\,.$$

The simplification of the triple sum is based on the fact that each node p_k is involved in three facet centroids of the hexahedron and with each of this in four facet centroids of the octahedron.

In order to prove $q = q'$ it remains to show that $q'_n = (0,0,0)^{\text{t}}$, which holds since the normals of opposing octahedron faces pairwise sum up to the zero vector. That is $n_1 = -n_7$, $n_2 = -n_8$, $n_3 = -n_5$, and $n_4 = -n_6$, as will be shown exemplarily for the first identity. The representation of the octahedron normals implies

$$\begin{aligned}
n_1 &= (d_2 - d_1) \times (d_5 - d_1)\\
&= +\frac{1}{16}\big[(p_5 + p_6 - p_3 - p_4) \times (p_5 + p_8 - p_2 - p_3)\big]\\
&= -\frac{1}{16}\big[(p_2 + p_3 - p_5 - p_8) \times (p_3 + p_4 - p_5 - p_6)\big]\\
&= -\big((d_3 - d_6) \times (d_4 - d_6)\big) = -n_7\,,
\end{aligned}$$

since the cross product is compatible with scalar multiplication and anti-commutative. The other identities can be deduced analogously.

In the case of tetrahedral elements successively substituting the expressions (3.6) and (3.4) in the representation of the base nodes $b_k = c_k$ implies

$$q_b' = \frac{1}{4}\sum_{k=1}^{4} c_k = \frac{1}{4}\sum_{k=1}^{4}\left(\frac{1}{3}\sum_{i=1}^{3}\left(\frac{1}{3}\sum_{j=1}^{3} p_{F^{\text{tet}}_{\overline{F}^{\text{tet}}_{k,i},j}}\right)\right) = \frac{1}{36}\sum_{k=1}^{4} 9p_k = q\,.$$

Here, it has been used that each node p_k is involved in the three adjacent element face centroids d_k and with each of this in three dual element face centroids c_k.

Finally, in the case of prismatic elements the base nodes are given by $b_k = a_k(\tau)$ resulting in

$$\begin{aligned} q_b' &= \frac{1}{6}\sum_{k=1}^{6} a_k(\tau) = \frac{1}{6}\sum_{k=1}^{6}\left[(1-\tau)d_{\overline{F}_{k,1}} + \frac{\tau}{2}\left(d_{\overline{F}_{k,2}} + d_{\overline{F}_{k,3}}\right)\right] \\ &= \frac{1}{6}\left[3(1-\tau)(d_1+d_5) + 2\tau(d_2+d_3+d_4)\right] \\ &= \frac{1}{6}\left[(1-\tau)(p_1+\cdots+p_6) + \tau(p_1+\cdots+p_6)\right] = q\,. \end{aligned}$$

Since $q_b' = q$ has been shown for arbitrary τ, this holds in particular for the specific choice of τ given in (3.8). $\square$

Whereas Lemma 3.2 implies $q_n' = (0,0,0)^{\mathrm{t}}$ for arbitrary hexahedra, this does not generally hold for tetrahedral, pyramidal and prismatic elements as can be shown by simple counterexamples. However, centroid preservation can easily be achieved by an additional translation step after transforming the element. According to Lemma 3.2 the required translation vector is given by $-q_n'$ in the case of tetrahedral and prismatic elements and by $q - q'$ in the case of pyramidal elements. For the latter even $q = q_b'$ does not hold in general as can also be shown by simple counterexamples.

Due to its geometric construction and depending on the choice of the normals scaling factor σ given in Definition 3.4, the size of a transformed element can differ significantly from the size of the initial element. This can be avoided by an element scaling step, which can be combined with the centroid preservation step described in the previous paragraph. Let $Q = (q,\ldots,q)$ and $Q' = (q',\ldots,q')$ denote $3 \times |E|$ matrices with each column representing the initial and transformed element centroid q and q', respectively. For an arbitrary scaling factor $\zeta > 0$ the scaled transformed element E_s' preserving the initial element centroid q is given by

$$E_s' := Q + \zeta(E' - Q')\,. \tag{3.11}$$

A natural choice of ζ applicable to all element types is, for example, the mean length of all edges of E divided by the mean length of all edges of E'. This choice of ζ will be denoted as mean edge length preserving scaling. Other scaling schemes could for example be based on preserving the mean volume of all node tetrahedra T_k.

Valid initial elements of all types under consideration defined by randomly chosen nodes and their mean ratio quality numbers can be seen in the leftmost column of Fig. 3.7. In addition, each row contains the elements obtained by successively applying the same element transformation using $\sigma = 1$ and mean edge length preserving scaling to prevent the successive growth of the elements.

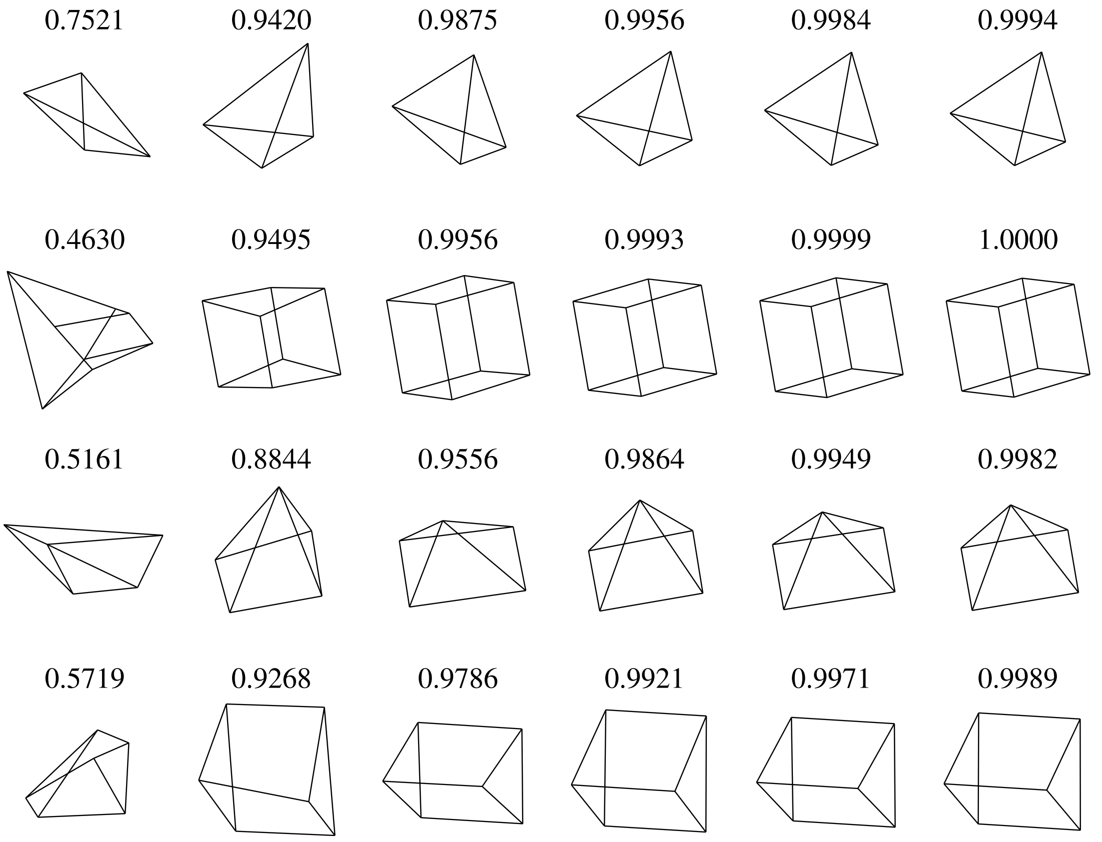


Figure 3.7: Elements and their mean ratio quality numbers obtained by successively applying the regularizing transformation using $\sigma = 1$ and mean edge length preserving scaling

Mean ratio quality numbers are successively improved resulting in nearly regular elements after applying the transformation five times, as can be seen by the elements depicted in the rightmost column. In particular, the first transformation step leads to a considerable improvement of element regularity and with this in element quality. In the context of mesh smoothing, such a rapid change of shape is not always favorable since it also increases the risk of invalid neighbor element generation. Therefore, relaxation is applied, that is the linear combination

$$E'_r := (1 - \varrho)E + \varrho E'_s \tag{3.12}$$

of the initial element E and the transformed and scaled element E'_s is used, where $\varrho \in (0, 1]$ represents the relaxation parameter. The smaller ϱ gets, the smaller are the geometric changes compared to the initial element E. For the choice $\varrho = 1$ it holds that $E'_r = E'_s$.

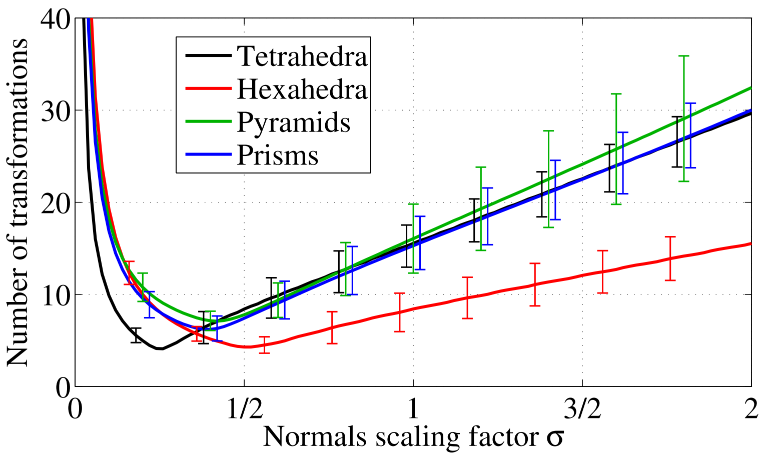
Due to its recursive nature and the involvement of cross products and vector normalization, proving the regularizing effect of the non-linear transformation given by Definition 3.4 is not an easy task. Therefore, it has been substantiated by numerical tests, which also provide more insight into the regularizing behavior of the transformation with respect to the scaling parameter σ.

The tests are based on generating 100,000 random initial elements of each type and taking 101 equidistant values for $\sigma \in [0, 2]$. For each pair (E_j, σ_k) taken from the Cartesian product of all elements and all σ-values, the geometric transformation using the scaling factor σ_k has been iteratively applied starting with the initial element E_j until the mean ratio quality number of the resulting element deviated less than 10^{-6} from the ideal value one. After each iteration step the resulting element has been scaled with ζ being set to the inverse of the arithmetic mean of all edge lengths to avoid numerical instabilities caused by the successive element growth, which would have occurred otherwise. Due to the scaling invariance of the transformation according to Lemma 3.1 this has no influence on the speed of regularization. In addition, relaxation has been disabled by the specific choice $\varrho = 1$. For each element type this test resulted in 10,100,000 transformation cycles, each represented by its number of iterations, thus providing an indication of the speed of regularization.

Results of this test are depicted in Fig. 3.8a. Here, for each element type and specific choice of σ_k the according graph gives the average iteration number of all 100,000 transformation cycles performed. In addition, for selected values of σ the standard deviation is marked by error markers. As can be seen, all graphs show a similar behavior. To be precise, the average iteration number increases if σ tends to zero, since the transformation tends to become a node averaging scheme without regularizing effect. For $\sigma \in (1/2, 2)$, that is after reaching its minimum, the average iteration number depends almost linearly on σ. Thus, the parameter σ provides an easy control for the speed of regularization.

Due to the usage of random initial nodes, the probability of an initial element being invalid increases with its complexity, i.e. with its number of nodes.

(a) Arbitrary initial elements



(b) Valid initial elements

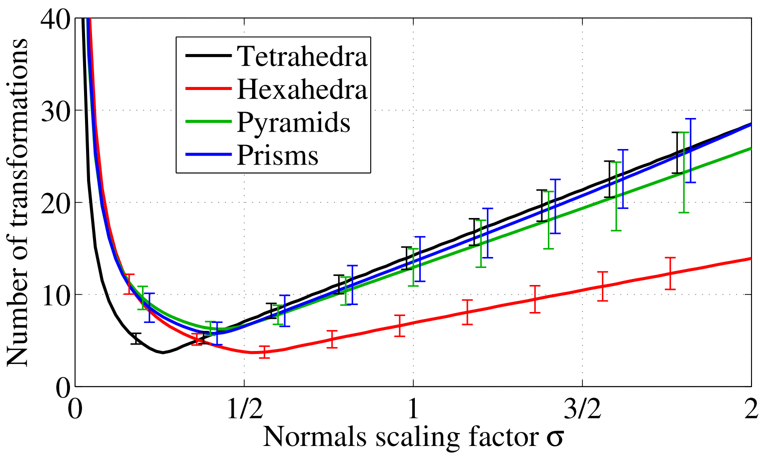


Figure 3.8: Regularization speed with respect to scaling parameter σ

This is reflected by the valid initial element percentage amounting to 50.15% for tetrahedral, 0.37% for hexahedral, 9.36% for pyramidal, and 5.33% for prismatic elements. That is, in the majority of cases random generated elements are invalid. However, the transformation also reliably regularizes invalid elements, as can be seen by the average iteration numbers given in Fig. 3.8a. Furthermore, as is shown by the standard deviation error markers the dispersion of iteration numbers is moderate.

Regularization results are even more uniform, if only valid elements are used as is common in the case of mesh smoothing. Results based on using 100,000 valid initial elements for each element type are depicted in Fig. 3.8b. Compared to the test case of arbitrary initial elements the average iteration number becomes smaller while preserving its qualitative behavior with respect to σ. Furthermore, the standard deviation is decreased significantly. That is, for a given element type and normals scaling factor σ_k elements are regularized quite uniformly. This is an important fact with respect to the mesh smoothing approach presented below, assuring a consistent smoothing result.

The transformation of hexahedral elements, not only differs in the centroid preservation property, but also in its increased speed of regularization as can be seen in both cases. Hence, for mixed mesh smoothing it is advisable to use element type dependent values for σ to obtain a uniform regularization for all elements under consideration.

Chapter 4

GETMe smoothing

Whereas in the previous two chapters single element transformations have been considered, this chapter discusses how to use such transformations in order to smooth entire finite element meshes. This results in the definition of the geometric element transformation method, which is highly effective and efficient although being conceptually comparably simple.

4.1 Introduction

Let M be a conforming mesh with $n_p \in \mathbb{N}$ nodes $p_i \in \mathbb{R}^n$, where $n \in \{2, 3\}$ denotes the dimension and with $i \in I := \{1, \dots, n_p\}$ representing the index set of all nodes. The latter are connected by $n_e \in \mathbb{N}$ valid mesh elements $E_j = (p_{j_1}, \dots, p_{j_\ell})$, where $j \in J := \{1, \dots, n_e\}$. Each element E_j is uniquely defined by its node indices $j_1, \dots, j_\ell \in I$ with ℓ denoting the element type specific number of nodes. GETMe smoothing, in its presented form, is based on using the mean ratio quality criterion for transformation and termination control. Due to the requirements of this quality criterion, the initial mesh has to be valid, that is, all elements have to be valid according to Definition 3.1. Otherwise, untangling techniques, similar to those presented in [110, 124], have to be applied first as is the case in mean ratio-based optimization methods.

By using the mean ratio quality $q(E_j)$ of a single mesh element E_j introduced in Equation (3.2), minimal and mean mesh quality numbers for the entire mesh are given by

$$q_{\min} := \min_{j \in J} q(E_j) \quad \text{and} \quad q_{\text{mean}} := \frac{1}{n_e} \sum_{j=1}^{n_e} q(E_j) \,. \tag{4.1}$$

If boundary nodes are kept fixed during the smoothing process, the element with the lowest quality might consist entirely of boundary nodes. Since in this case $q_{\min}$

cannot be improved, $q^*_{\min}$ denoting the minimal element quality of all elements with at least one interior node will be used instead. This number gives the lowest quality of all improvable elements.

In the following a basic GETMe smoothing approach as well as an advanced version using concepts of adaptivity will be introduced. Both are founded on a two-phase smoothing approach. These phases can be distinguished by the number of element transformations performed within one smoothing step as is illustrated in Fig. 4.1 on the example of a planar polygonal mesh. Here, the initial nodes are indicated by black square symbols, initial elements by black edges. Transformed elements and the associated nodes are marked blue, new node positions obtained by one smoothing step are marked by red points.

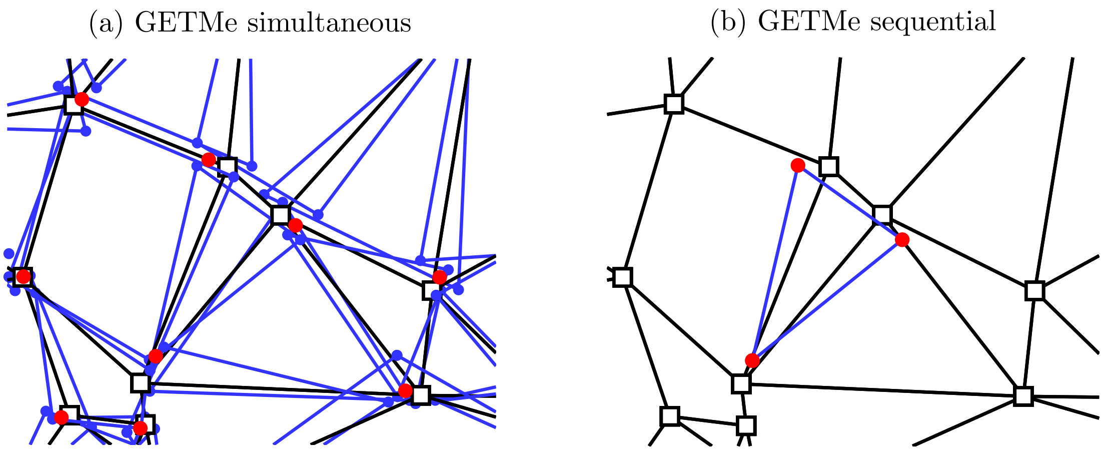


Figure 4.1: New node computation in GETMe simultaneous by transformed element nodes averaging and by direct worst element transformation in GETMe sequential

Fig. 4.1a depicts a simultaneous approach in which all elements E_j are transformed, scaled, and relaxed first. For each node p_i and an attached element E_j this results in a new temporary node $p'_{i,j}$. Then, the new mesh node positions are derived as weighted averages. An alternative approach of mesh smoothing based on regularizing element transformations is depicted in Fig. 4.1b. In this, only the improvable element with the lowest quality $q^*_{\min}$ is transformed and the obtained new element nodes are set directly, which also affects all neighboring elements. This is iteratively applied which results in a sequential approach of single element improvements.

4.2 Simultaneous GETMe smoothing

Before defining the single steps of the simultaneous GETMe algorithm, a more detailed sketch of its procedure will be given also introducing some additional notation. This scheme is based on transforming all mesh elements E_j simultaneously. Let $J(i) \subseteq J$ denote the index set of all mesh elements adjacent to the node p_i. For each node p_i and its adjacent elements E_j, $j \in J(i)$, this results in $|J(i)|$ new temporary nodes $p'_{i,j}$. Out of this, new node positions p'_i are derived as quality weighted means according to

$$p'_i := \frac{\sum_{j \in J(i)} w_j p'_{i,j}}{\sum_{j \in J(i)} w_j} \quad \text{with} \quad w_j := \big(1 - q(E_j)\big)^\eta, \tag{4.2}$$

where η denotes a prescribed weight exponent. Due to its geometric approach the validity of the resulting new mesh elements cannot be guaranteed so far. Therefore, a simple repair step for invalid elements is applied subsequently. All steps are repeated until the resulting mesh improvement is below a given tolerance. This results in the simultaneous GETMe smoothing algorithm summarized in Fig. 4.2.

```
Input  : Initial valid mesh
Output: Smoothed valid mesh
1 for Iter := 1 to MaxIter do
2     ResetTemporaryNodesAndWeights();
3     AddTransformedElementNodesAndWeights();
4     ComputeNewNodes();
5     IterativeInvalidElementResetting();
6     if improvement of $q_{\text{mean}}$ by last iteration below threshold then
7         break;
8     end
9 end
```

Figure 4.2: Algorithmic description of GETMe simultaneous smoothing

In the following, a more detailed description of the sub-functions of simultaneous GETMe smoothing is given:

- `ResetTemporaryNodesAndWeights:` In addition to the current mesh node coordinates given by p_i, old coordinates p_i^{old}, and the nominator $\hat{p}_i$ and denominator $\hat{w}_i$ in the representation of p'_i according to Equation (4.2) are used. In this sub-function, these additional variables are initialized to $p_i^{\text{old}} := p_i$, $\hat{p}_i := (0, 0, 0)$, and $\hat{w}_i := 0$.

- `AddTransformedElementNodesAndWeights:` This sub-function determines $\hat{p}_i$ and $\hat{w}_i$. For this purpose, each element E_j is transformed using one of the element type specific transformation schemes defined in the previous two chapters, scaled with respect to its centroid according to (3.11), and relaxed according to (3.12) using a fixed relaxation value $\varrho \in (0, 1]$. In the case of transformations not preserving the centroid, the transformed element is also shifted in order to preserve the centroid. This results in the new temporary nodes $p'_{i,j}$ and associated weight w_j, which are involved in Equation (4.2). These are added to the corresponding nominator and denominator variable $\hat{p}_i$ and $\hat{w}_i$, respectively.

  In the case of polygonal element transformations according to Equation (2.4), the parameters λ and θ are usually fixed prescribed values. In the case of volumetric element transformations, the normals scaling factor σ is chosen within a element type dependent interval $[\sigma_{\min}^{\text{type}}, \sigma_{\max}^{\text{type}}]$ depending on the actual element quality according to

$$\sigma_j := \sigma_{\min}^{\text{type}} + (\sigma_{\max}^{\text{type}} - \sigma_{\min}^{\text{type}})\big(1 - q(E_j)\big) .$$

- `ComputeNewNodes:` For each node p_i of the mesh compute its new position according to Equation (2.4) by dividing $\hat{p}_i$ by $\hat{w}_i$ and assign the resulting new mesh node to p_i. If boundary nodes have to preserved, such nodes are not modified. Otherwise projection techniques can be applied in order to preserve the shape of the mesh. This also holds in the case of surface meshes.

- `IterativeInvalidElementResetting:` Since the previous step might invalidate elements, like in the case of Laplacian smoothing, nodes of invalid elements are reset to their old position stored in p_i^{old}. This is iterated until no invalid element remains in the mesh. This loop is guaranteed to terminate, since the initial mesh is valid.

As has been shown numerically in Section 3.3, the average convergence rate of the dual element-based transformation for volumetric elements not only depends on the normals scaling factor σ_j but also on the element type. Therefore, element type dependent ranges for σ_j are used in `AddTransformedElementNodesAnd-Weights` to synchronize the speed of regularization. The concrete choice of σ_j is based on the individual element quality $q(E_j)$, which allows low quality elements to be smoothed more carefully than high quality elements. Within the mesh smoothing context, the element growth, centroid movement, and rapid change of shape caused by the element transformation are undesirable and hence also corrected in this sub-function.

In AddTransformedElementNodesAndWeights the nominator and denominator of the new node representation Equation (2.4) are computed. Again, quality weighted averaging puts an emphasis on low quality elements. This can additionally be enforced by an appropriate choice of the user defined exponent $\eta \geq 0$. If all elements adjacent to p_i are regular, the denominator in the representation of p_i' becomes zero. However, this is no problem, since from a local point of view, the original node position is already optimal and hence kept unchanged. Whereas according to the previous sub-function element centroid changes are prevented on a single element level, they are enabled on the complete mesh level due to weighted node averaging.

It should also be noted that according to AddTransformedElementNodesAndWeights new node coordinates are not immediately updated, since this would influence the computation of neighboring nodes. Therefore, new node computation and applying the new coordinates are conducted in ComputeNewNodes, ensuring that the computation does not depend on the numbering scheme of nodes and elements. This is of particular interest for parallelized implementations of the GETMe approach, in order to obtain reproducible results.

Due to the geometry driven approach, the steps accomplished so far may lead to the generation of invalid elements. Such elements are identified and removed in IterativeInvalidElementResetting. For reasons of simplification and due to good results in various numerical tests, this is done by iteratively resetting the associated nodes to their previous position p_i^{old}.

A likewise simple approach is used for the termination control of the algorithm. It is based on the number of iterations performed so far and the mean mesh quality number of the current and previous iteration. Smoothing is terminated if the mean mesh quality improvement obtained by one step of GETMe simultaneous smoothing drops below a given threshold.

4.3 Sequential GETMe smoothing

Although the sequential approach is likewise based on improving element quality by applying the regularizing element transformation, the way in which transformed element nodes are handled is much simpler. Here, the elements with the lowest quality are selected iteratively and transformed within the mesh. That is, new node coordinates are set directly. This is repeated until specific termination criteria are met. In order to allow an additional control over element selection, each element quality number $q(E_j)$ is corrected by a quality penalty value $\pi_j \geq 0$, which is initialized to zero. A proper control of this penalty parameter also avoids an infinite loop of picking, transforming and resetting the same element due to invalidated neighboring elements. This results in the sequential GETMe algorithm as given in Fig. 4.3.

```
Input  : Initial valid mesh
Output: Smoothed valid mesh
for Iter := 1 to MaxIter do
    TransformAndSetWorstQualityElement();
    InvalidElementAndPenaltyHandling();
    if no $q^*_{\min}$ improvement within the last $n$ iterations then
        break
    end
end
```

Figure 4.3: Algorithmic description of GETMe sequential smoothing

The role of the sub-functions of this algorithm are described in more detail in the following:

- `TransformAndSetWorstQualityElement:` The index j of the element with lowest corrected quality $q(E_j) + \pi_j$ is determined first and all its nodes are stored according to $p_i^{\text{old}} := p_i$. Subsequently E_j is transformed using fixed transformation parameters, scaled while restoring its centroid according to (3.11) and relaxed according to (3.12) using a fixed parameter value $\varrho \in (0, 1]$ resulting in the new temporary nodes which directly replace the current nodes p_i of the element.
- `InvalidElementAndPenaltyHandling:` In the case that any of the neighboring elements becomes invalid, revert to the initial node positions. Increase quality penalty value π_j of element E_j by fixed value $\Delta\pi_{\text{i}} > 0$ if an invalid element was generated, by $\Delta\pi_{\text{r}} > 0$ if the same element was picked in previous step, and decrease by $\Delta\pi_{\text{s}} > 0$ if transformation was successful.

In `TransformAndSetWorstQualityElement` the element E_j with the lowest quality number is determined first, which is an $\mathcal{O}(1)$ operation if a min heap of quality numbers and associated element indices is used. Subsequently, E_j is transformed and adjusted similar to the simultaneous approach. However, since only one element is transformed, there is no need for a quality dependent control of the transformation and averaging parameters. Hence, it suffices to use a fixed set of conservatively chosen σ^{type} values to slightly improve element quality while reducing the risk of invalid element generation.

Since only one element is transformed, its new nodes can immediately be set after the original node coordinates have been stored for invalid element handling purposes. Likewise handling of invalid elements becomes simple, since only the element E_j and its direct neighbor elements are affected by the node updates. In the case that one of these elements becomes invalid, the initial node coordinates are restored in `InvalidElementAndPenaltyHandling`.

Without further modifications, this could result in an infinite loop of picking, transforming and reverting the same element and its nodes. This is prevented by a simple element quality penalty mechanism based on adjusting the quality penalty value π_j associated to the selected element E_j. In this, π_j is increased by $\Delta\pi_{\mathrm{i}}$ if invalid elements have been generated. This leads to a successively increased corrected quality number $q(E_j) + \pi_j$ of E_j until a different element is picked first in `TransformAndSetWorstQualityElement`. The increase of the penalty term in case of a repeated selection of the same element can be accelerated by the choice of $\Delta\pi_{\mathrm{r}} > 0$. Finally, if an element E_j has been successfully transformed and applied, its quality penalty value π_j is decreased by $\Delta\pi_{\mathrm{s}}$. Using a min heap for determining the lowest quality element also requires the update of changed element qualities, which is an $\mathcal{O}(\log n_e)$ operation for the selected element E_j itself as well as its neighboring elements if node updates have been successful. Hence, the selection of the elements, which have to be transformed, can be handled efficiently.

4.4 GETMe smoothing

By deriving new node positions as weighted means of transformed element nodes for the complete mesh, the simultaneous GETMe approach focuses on the overall mesh quality. In contrast, transforming only low quality elements in the case of the sequential approach puts an emphasis on improving the lowest element quality. These individual strengths with respect to quality and runtime behavior are combined by applying GETMe simultaneous first, which improves overall mesh quality and with this problematic parts of the mesh containing the element with the lowest quality. Therefore, the subsequently applied GETMe sequential approach can improve the remaining low quality elements more effectively. The resulting combined approach is simply denoted as GETMe smoothing.

As numerical tests have shown, this basic GETMe smoothing approach already results in quality numbers at least comparable to those of a state of the art global optimization approach within significantly shorter runtimes. Results of these tests are given in Chapter 5.

4.5 The GETMe adaptive approach

The GETMe approach described in the previous section has been further improved with the following aspects in mind: improving smoothing quality, reducing the number of control parameters, preserving simplicity, and increasing its amenability to parallel implementation.

Similar to the GETMe approach, smoothing by the new GETMe adaptive approach is performed in two stages. The first is geared towards improving q_{mean}, the second towards improving q^*_{min}. However, in contrast to GETMe smoothing, these two stages are integrated into one smoothing loop. Furthermore, both stages use the same weighted node averaging scheme given by

$$p'_i := \frac{\sum_{j \in J(i)} w_j p'_{i,j}}{\sum_{j \in J(i)} w_j}, \quad \text{with } w_j := \sqrt{\frac{\sum_{n \in N(j)} q(E_n)}{|N(j)| q(E_j)}}\,, \tag{4.3}$$

in order to compute new node positions. Here $N(j)$ denotes the index set of the neighbor elements of E_j, i.e. elements, which share at least one node with E_j. The number of such neighbor elements is given by $|N(j)|$. Compared to GETMe simultaneous smoothing using the averaging scheme according to Equation (4.2), the first stage of GETMe adaptive smoothing differs in the choice of the weights w_j, as can be seen by Equation (4.3), where the quality of a single element is related to the mean quality of its direct neighbors. This puts an emphasis on weights of lower quality elements. In GETMe sequential only the worst element is transformed and the resulting new element nodes are set directly, affecting all neighboring elements. In contrast, both stages of GETMe adaptive smoothing use the weighted node averaging scheme according to Equation (4.3) in order to provide a more balanced result. The algorithmic description given in Fig. 4.4 provides an overview of the adaptive approach, which will be described in detail in the following.

In GETMe adaptive, only elements with a mean ratio quality number below a given threshold q_t are transformed. This threshold is set to one in line 1 of the algorithm, which enforces all elements to be transformed during the first stage. Furthermore a state variable, indicating the q_{mean} oriented stage, and an associated table of node relaxation values are initialized. The role of the relaxation values will be discussed later and specific choices for the parameters involved will be given in Section 5.5.

The following sub-functions are applied:

- `ResetTemporaryNodesAndWeights`: For each node initialize a temporary node sum $\hat{p}_i := (0, 0, 0)$ and weight sum $\hat{w}_i := 0$.
- `AddTransformedElementNodesAndWeights`: For each mesh element E_j with $q(E_j) \leq q_t$, compute the associated weight w_j according to Equation (4.3), apply the geometric transformation to E_j using fixed transformation parameters, scale E'_j with respect to its centroid in order to preserve the sum of all element edge lengths and add the resulting weighted nodes $w_j p'_{i,j}$ to

```
Input  : Initial valid mesh
Output: Smoothed valid mesh
q_t := 1;
State := MeanCycleRunning;
SetNodeRelaxationValueTable(State);
for Iter := 1 to MaxIter do                         /* Main smoothing loop */
    ResetTemporaryNodesAndWeights();
    AddTransformedElementNodesAndWeights($q_t$);
    AddUntransformedElementNodesAndWeights($q_t$);
    ComputeNewNodes();
    IterativeNodeRelaxation();
    if State = MeanCycleRunning and $\Delta q_{\mathrm{mean}} < tol$ then
        State := MinCycleStart;
        SetNodeRelaxationValueTable(State);
    end
    if State = MinCycleRunning then
        if no $q^*_{\min}$ improvement in last iteration then
            NoMinImproveCounter ++;
        end
        if NoMinImproveCounter > MaxNoMinImproveCounter then
            State := MinCycleStart;
        end
    end
    if State = MinCycleStart then
        if no $q^*_{\min}$ improvement in last min cycle then break;
        State := MinCycleRunning;
        NoMinImproveCounter := 0;
        $q_t$ := DetermineTransformationThreshold();
    end
end
```

Figure 4.4: Algorithmic description of the GETMe adaptive smoothing

the temporary node sums $\hat{p}_i$ and the weight w_j to the associated temporary weight sums $\hat{w}_i$.

- `AddUntransformedElementNodesAndWeights:` Let I_T denote the index set of non-fixed nodes, which belong to at least one transformed element. For each untransformed element E_j with $q(E_j) > q_t$ and at least one node p_i with $i \in I_T$ compute the associated weight w_j according to Equation (4.3) and add the weighted, but untransformed coordinates $w_j p_i$ of the nodes of E_j to the temporary node sums $\hat{p}_i$ and the weight w_j to the associated temporary weight sums $\hat{w}_i$.

- `ComputeNewNodes:` For each $i \in I_T$ with $\hat{w}_j > 0$ replace the contents of $\hat{p}_i$ by the weighted coordinates $(1/\hat{w}_i)\hat{p}_i$ resulting in the new node coordinates p_i' according to Equation (4.3). In the case of $\hat{w}_j = 0$ use $\hat{p}_i := p_i$.

- `IterativeNodeRelaxation:` Set the index into the table of relaxation values of each node to $r_i := 1$ and apply $p_i' := p_i$ for all $i \notin I_T$. For all $i \in I_T$ and a given table of relaxation values $R = (\varrho_1, \ldots, \varrho_k)$ compute the associated new coordinates as

$$p_i' := (1 - \varrho_{r_i})p_i + \varrho_{r_i}\hat{p}_i \,, \tag{4.4}$$

  Subsequently, run the following iterative relaxation process until no invalid element remains in the mesh: For each invalid element mark the associated nodes and for each marked node increase the relaxation counter and recompute p_i' according to Equation (4.4). Here, the last entry ϱ_k in the table R of descending relaxation values equals 0, and the increase of r_i is stopped if k is reached. This assures that if relaxation cannot avoid the generation of invalid elements, the nodes are reseted to their original valid position like in the case of the GETMe approach. After termination of the relaxation loop set the new mesh node coordinates to p_i'.

The subsequent lines 10 to 26 of the algorithm control the two smoothing stages. Here, the first if-statement of this block controls the termination of the q_{mean}-oriented first smoothing stage, in which all elements are transformed due to the choice $q_t := 1$. This stage, indicated by setting the state variable to `MeanCycleRunning`, is terminated in case the q_{mean}-improvement (denoted as Δq_{mean}) of two consecutive iterations drops below a prescribed threshold `tol`, which is usually set to 10^{-4}. If this is the case, the state is changed to `MinCycleStart` and an alternative table of relaxation values used in `IterativeNodeRelaxation` is set. Specific choices used for the tables of relaxation values are given in the numerical examples section.

The second stage, which is geared towards improving $q^*_{\min}$, is divided into smoothing cycles consisting of an adaptive number of smoothing iterations. Here, at the beginning of each cycle (cf. line 22 of the algorithm), smoothing is terminated if the previous cycle did not result in an improvement of $q^*_{\min}$. Subsequently, the no improvement counter is reset and a new element quality threshold q_t is determined. This is done by `DetermineTransformationThreshold` in which the quality numbers of all mesh elements are sorted in ascending order. After that, q_t is set to the quality value of a prescribed position in this sorted vector. Usually, this position is set to a fixed percentage of all elements times the number of elements.

After determining the new node positions within one $q^*_{\min}$-oriented smoothing cycle, $q^*_{\min}$ is checked in lines 14f. If there is no improvement, then the no improvement counter is increased by one and the cycle is terminated as soon as this counter reaches a prescribed limit.

Due to their common approach of using geometric element transformations in combination with weighted transformed node averaging, GETMe smoothing as well as GETMe adaptive smoothing are generally applicable. As has been already shown by various examples of GETMe smoothing this includes structured and unstructured surface and volume meshes consisting of triangular, quadrilateral, tetrahedral, hexahedral, pyramidal, and prismatic elements [27–31, 35]. From an algorithmic point of view, GETMe adaptive differs from GETMe smoothing in the following points:

- Incorporation of two smoothing stages within one smoothing loop instead of applying two separate loops.

- Relaxation is applied to both previous and new node positions instead of involving previous and transformed elements. Furthermore, the relaxation parameter is adjusted on a nodal basis, which also replaces the iterative invalid element node resetting scheme.

- During the $q^*_{\min}$ oriented stage, GETMe adaptive transforms more than one element per iteration, which is more suitable for parallel mesh smoothing. Furthermore, both stages use the same node averaging approach. However, nodes of elements with $q(E) > q_t$ are directly used without transformation and scaling.

- Iterations of the second stage are organized in cycles. The transformation quality threshold q_t is updated at the beginning of each smoothing cycle.

- Fixed element transformation parameters are used instead of quality adaptive transformation parameters. The adjusted weights in the transformed element nodes averaging scheme are based on neighboring element quality ratios.

- The exponent η of the node weights w_j given in Equation (4.2) and the penalty parameters of GETMe sequential are removed. This eliminates the need for a minimal element quality heap required by the sequential substep of GETMe.

4.6 Implementation and parallelization

Results of the GETMe approach are given in the next chapter using a straightforward C++ implementation [154]. This first experimental implementation incorporates an object-oriented data structure, which provides enhanced topology information for nodes and elements facilitating implementational flexibility but also leading to significantly increased memory requirements. Furthermore, the interchange of information between node and element objects leads to an additional runtime overhead. Nevertheless, GETMe smoothing turned out to be fast and effective if compared with other smoothing methods like smart Laplacian smoothing and a global optimization-based approach.

In order to make one step forward towards demonstrating the true efficiency of geometry-based smoothing approaches, the GETMe adaptive smoothing is implemented aiming at improving runtime and memory profile. Although such an implementation could have been based on C++, the C programming language [155] has been chosen instead, since it also builds a good foundation for future developments involving GPU-based computations using OpenCL [156], CUDA [157], or OpenACC [158], which are mainly C oriented. As has been recently shown, combined with domain decomposition methods such approaches open a new era in scientific computing [159]. In the following, some key aspects of this improved implementation will be discussed.

Mesh nodes and elements are stored in arrays containing the node coordinates and the node indices of the elements, respectively. Furthermore, since the weights w_j according to Equation (4.3) involve quality numbers of neighbor elements, a neighbor element index table is also used. In addition to the current node coordinates p_i, GETMe adaptive smoothing also uses arrays in order to store the temporary node coordinate sums $\hat{p}_i$, the new node coordinates p'_i, the weight sums w_i, the element quality numbers $q_j := q(E_j)$, and the indices r_i into the table of relaxation values for all $i \in \{1, \dots, n_N\}$ and $j \in \{1, \dots, n_E\}$, where n_N and n_E denote the number of mesh nodes and elements. Since the mean ratio quality criterion is expensive to evaluate, entries of the element quality array are only updated in the case of element node coordinate changes.

First, all values $\hat{p}_i$, w_i are initialized to zero in `ResetTemporaryNodesAndWeights`. After that, in `AddTransformedElementNodesAndWeights` each element is transformed and scaled and the associated weighted new node coordinates and weights are successively added to the corresponding variables $\hat{p}_i$, w_i. Here, the weights are determined using the tabulated element quality numbers q_j. Similarly, `AddUntransformedElementNodesAndWeights` adds untransformed but weighted nodes and the corresponding weights to $\hat{p}_i$ and w_i. Then, the unrelaxed new node coordinates according to Equation (4.3) are computed by `ComputeNewNodes` and stored in $\hat{p}_i$.

During the $q^*_{\min}$ oriented stage of the GETMe adaptive approach, these sub-functions operate only on a subset of the mesh, which is defined by the elements with a quality number below the quality threshold q_t. This also holds for `IterativeNodeRelaxation`, which iteratively determines the final new node coordinates p'_i. Since the relaxation step does not require the re-evaluation of the weights w_i, the element quality vector can already be filled with the quality numbers of the elements with respect to the new node coordinates p'_i. Here, non positive entries, indicate invalid elements whose nodes require an additional node relaxation step using an adjusted individual relaxation parameter. The relaxation loop is terminated when all elements become valid. Then all current mesh nodes p_i are set to the new coordinates p'_i.

As can be seen, the loops of all these sub-functions are amenable to parallelization. In the current C implementation of GETMe adaptive parallelization was realized by using the OpenMP API version 3.1 [160]. Here, simply adding `#pragma omp parallel for` directives suffice for the loops to be executed in parallel. In this context, reading and updating data like $\hat{p}_i$ and w_i by simultaneous threads requires synchronization by the use of atomic operations. Results of the parallelized version of GETMe adaptive smoothing following this approach are given in Section 5.5.

Further potential runtime improvements can be achieved by avoiding thread synchronization caused by atomic operations using thread private dynamically allocated memory, which requires an additional implementational effort. However, for the current version, runtime optimization of GETMe adaptive by modifying data handling on an implementational level has not been applied.

The OpenMP API supports shared memory multiprocessing programming, where concurrently accessing a large amount of memory by different threads on the same system often leads to a significant bottleneck. As an alternative, distributed computing approaches can be applied. Here, the mesh is partitioned and smoothing of the submeshes is conducted on different systems. GETMe adaptive smoothing is also suitable for this type of parallelization, since its local smoothing approach, incorporating only information of direct neighbor elements, results in a low submesh interface communication overhead.

4.7 Incorporating alternative quality criteria

In the previous, mesh quality was assessed using the mean ratio quality criterion, which was introduced in the context of optimization-based smoothing methods [20, 22]. By measuring the distance of an arbitrary valid element from a prescribed reference element, the mean ratio criterion not only satisfies basic requirements, such as being well defined for all element types under consideration and the invariance under affine transformations, but it also provides flexibility by the choice of the reference shape. Furthermore, since it is normalized, element quality can be easily incorporated as a control parameter. Due to these advantages, and in order to allow an equitable comparison with the results of the mean ratio-based global optimization approach of Mesquite [137], GETMe smoothing, in its presented form, has adopted the mean ratio quality criterion for smoothing control.

However, depending on the application, additional requirements might be imposed, which can be measured by more specialized quality metrics. One example is the warpage of quadrilateral faces of volumetric elements, for which an exemplary modification of the sequential GETMe approach will be considered in the following. The resulting algorithm focuses on improving the lowest element quality with respect to the mean ratio and the warpage criterion. This shows that GETMe smoothing is not only flexible with respect to mesh element types, but also with respect to quality metrics and smoothing objectives.

Let $Q := (p_0, \dots, p_3)$ denote a quadrilateral face of a volumetric element. The normal of the triangle defined by a node p_k and its connected neighbors is given by $n_k := \big(p_k - p_{(k+3) \bmod 4}\big) \times \big(p_{(k+1) \bmod 4} - p_k\big)$, $k \in \{0, \dots, 3\}$. According to [161], for $\hat{n}_k := \frac{1}{\|n_k\|} n_k$ the warpage $w(Q) \in [0, 2]$ of Q is defined as

$$w(Q) := 1 - \min\left\{(\hat{n}_0 \cdot \hat{n}_2)^3, (\hat{n}_1 \cdot \hat{n}_3)^3\right\} .$$

Here, $w(Q) = 0$ indicates non-warped quadrilaterals consisting of coplanar face nodes and large values indicate strongly warped low quality quadrilaterals.

Let w_{max} denote the maximal warpage of all quadrilateral faces in a given mesh and w_{mean} the arithmetic mean of the warpage values of all unique quadrilateral faces. Since iteratively applying the geometric transformation according to Definition 3.4 leads to regular elements, the warpage of quadrilateral element faces tends to zero. Hence, GETMe smoothing is also suitable for mesh smoothing with respect to the warpage criterion. In the case of the sequential substep, only the worst element selection scheme has to be adjusted as is described in the following.

For a given element $E \in \{E^{\mathrm{hex}}, E^{\mathrm{pyr}}, E^{\mathrm{pri}}\}$ let I_Q denote the index set of all its quadrilateral faces Q_i. The warpage of an element E is defined as

$$w(E) := \begin{cases} \max_{i \in I_Q} w(Q_i) & \text{if } E \in \{E^{\text{hex}}, E^{\text{pyr}}, E^{\text{pri}}\}\,, \\ 0 & \text{otherwise.} \end{cases}$$

This element quality number and the mean ratio criterion are not compatible, since large values of $w(E) \in [0, 2]$ indicate low quality elements, whereas large values of $q(E) \in [0, 1]$ indicate high quality elements. Therefore, $w(E)$ is transformed by $(1 - w(E)/2)$ and element quality in the modified GETMe sequential substep is assessed by the combined criterion

$$q_\gamma(E) := \min\big(q(E), \gamma(1 - w(E)/2)\big)\,.$$

Here, the fixed weight $\gamma > 0$ allows either to balance the two quality criteria or to put an emphasis on one of them. For example, large values of γ focus on mean ratio quality improvements, whereas small values of γ focus on warpage quality improvements. In all cases, low values of q_γ indicate low quality elements. Consequently, the element with the lowest penalty corrected $q_\gamma(E)$ value in the mesh is transformed in order to improve its quality. If the lowest combined quality value is defined by a warped quadrilateral face connecting two elements, the one with the lower mean ratio quality number is selected for transformation. Results obtained for such a q_γ-based GETMe sequential approach applied to the GETMe smoothed meshes of the previous examples are given in Section 5.4.4.

Chapter 5

Numerical mesh smoothing results

This chapter provides a broad overview and an in-depth analysis of numerical results obtained by smoothing meshes of various types. Results are given for all variants of GETMe smoothing, some variants of Laplacian smoothing, and a state of the art global optimization-based method. Synthetical as well as real world models and meshes are considered. Partly based on results of [27–32] published during the last four years, this chapter also documents the progress in the development of GETMe smoothing variants both with respect to algorithmic aspects including default parameter sets and implementational details. In Section 5.1 exemplaric smoothing examples in terms of mesh types and fundamental properties of different smoothing algorithms will be given first. The subsequent Sections 5.2 to 5.5 focus either on characteristics of specific mesh types or specific GETMe variants.

5.1 Exemplary mesh smoothing results

In this section, a first overview of smoothing results for generic meshes will be presented. Three examples are considered: a random planar domain, a surface mesh of a tensile structure and volumetric meshes of a prestressed concrete bridge part. Results of GETMe smoothing are compared to those of other smoothing methods, which are described in detail in Section 1.3.2 and Section 1.3.3.

5.1.1 Planar polygonal meshes

Three meshes of a random planar domain are considered, which differ in element types and mesh resolution. They are depicted on the left side of Fig. 5.1. The first is a mesh generated by Delaunay-triangulation of the nodes of a slightly distorted regular grid. According to the theory of Delaunay-triangulations, slightly distorting the initial nodes results in a unique triangulation, since no four points lie on the same circle. The resulting triangular mesh consists of 1838 nodes and 3060 elements. The second mesh considered is a quadrilateral mesh consisting of 6562 nodes and 6111 elements. Finally, a mixed triangular quadrilateral mesh has been generated by splitting randomly selected quadrilaterals of an alternative quadrilateral mesh along their diagonals into two triangles each. The resulting mesh consists of 8152 nodes and 12,100 elements in total. Here, the number of triangles amounts to 8732 and the number of quadrilaterals amounts to 3368. In order to improve the smoothing potential, all meshes have also been distorted by randomly chosen node movements, which do not invalidate the mesh. The associated distorted meshes are depicted on the right side of Fig. 5.1.

All undistorted initial meshes as well as the distorted initial meshes have been smoothed by smart Laplacian smoothing, a state of the art global optimization based approach and GETMe smoothing consisting of applying consecutively the simultaneous as well as the sequential substep. In the case of GETMe simultaneous, mean element edge length preserving scaling and no relaxation, i.e. $\varrho = 1$, have been applied. Furthermore, by using the exponent $\eta = 0$, the weighted node averaging scheme simplifies to the arithmetic mean, which further reduces the numerical complexity. The element transformation parameters have been set to $\lambda = 7/20$ and $\theta = \pi/12$. In the case of the GETMe sequential substep likewise mean element edge length preserving scaling was used. Furthermore, the relaxation parameter has been set to $\varrho = 1/100$, and the penalty parameters $\Delta\pi_i = 10^{-4}$, $\Delta\pi_r = 10^{-5}$, and $\Delta\pi_s = -10^{-3}$ were applied in all cases. Transformation parameters have been set to $\lambda = 9/20$ and $\theta = 5\pi/36$. Since one iteration of GETMe sequential smoothing consists of transforming a single element, the comparatively expensive evaluation of mesh quality, which is used for termination control, was only assessed after a cycle of 100 consecutive single element transformations each. Smoothing results are given in Table 5.1.

Results have been obtained by a straightforward C++ implementation of GETMe smoothing and smart Laplacian smoothing. They are compared to the results of the feasible Newton-based global optimization approach of Mesquite [137], which is also implemented in C++. Runtimes have been measured on a notebook with an Intel® Core$^{\text{TM}}$ i5-540M CPU (dual core, 3MB cache, 2.53GHz), 4GB RAM, 64bit Linux operating system with kernel 2.6.34.4, and the GNU C++ compiler version 4.5.1.

It can be seen that GETMe smoothing due to the incorporation of the sequential approach focusing on improving worst element quality leads to superior $q^*_{\min}$ quality numbers for all meshes. Although global optimization focuses on improving the mean mesh quality, worst element quality numbers are also comparably

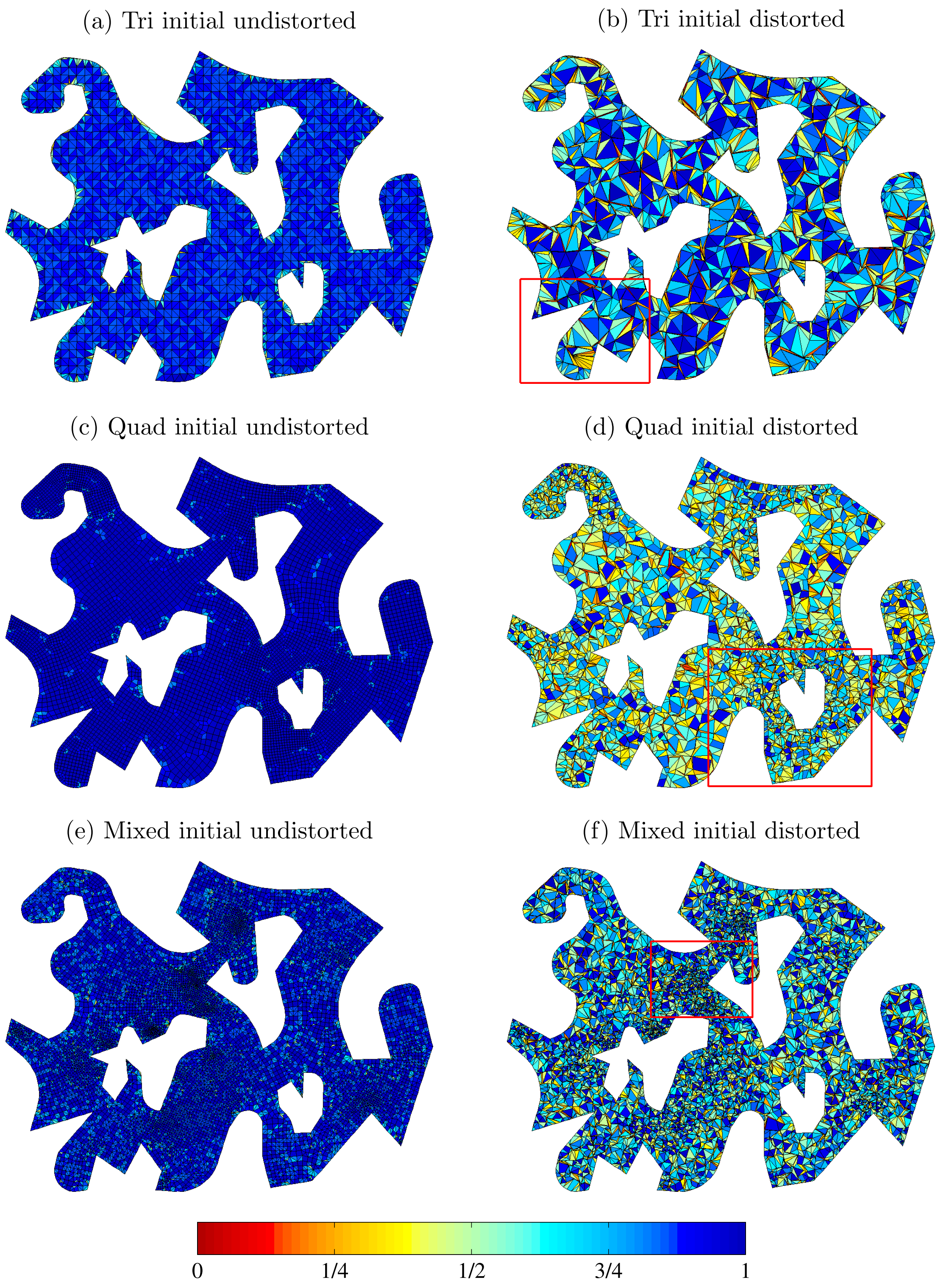

(a) Tri initial undistorted
(b) Tri initial distorted
(c) Quad initial undistorted
(d) Quad initial distorted
(e) Mixed initial undistorted
(f) Mixed initial distorted
0
1/4
1/2
3/4
1

Figure 5.1: Initial planar meshes. Mesh elements are colored according to their mean ratio value

Table 5.1: Planar mixed mesh smoothing results

Mesh	Method	Iter	Time (s)	$q^*_{\min}$	q_{mean}
Tri	Initial undistorted	–	–	0.0034	0.8193
	Smart Laplace	8	0.02	0.1032	0.9182
	Global Optimization	24	0.06	0.5044	0.9190
	GETMe	16/4700	0.04	0.5646	0.9157
	Initial distorted	–	–	0.0000	0.3897
	Smart Laplace	15	0.03	0.0042	0.9140
	Global Optimization	100	0.28	0.5045	0.9190
	GETMe	29/3800	0.07	0.5646	0.9155
Quad	Initial undistorted	–	–	0.6195	0.9712
	Smart Laplace	1	0.01	0.6157	0.9713
	Global Optimization	6	0.20	0.6546	0.9719
	GETMe	5/1600	0.03	0.7234	0.9711
	Initial distorted	–	–	0.0003	0.3896
	Smart Laplace	26	0.26	0.0109	0.9466
	Global Optimization	261	4.33	0.6546	0.9719
	GETMe	31/1200	0.20	0.7234	0.9684
Mixed	Initial undistorted	–	–	0.3052	0.8801
	Smart Laplace	6	0.11	0.2006	0.9236
	Global Optimization	16	0.67	0.4576	0.9411
	GETMe	18/1800	0.29	0.5048	0.9354
	Initial distorted	–	–	0.0000	0.4499
	Smart Laplace	11	0.22	0.0037	0.9141
	Global Optimization	93	2.06	0.4576	0.9411
	GETMe	32/1900	0.54	0.5059	0.9354

high. In contrast, the $q^*_{\min}$ numbers achieved by smart Laplacian smoothing are inferior in all examples. This is also reflected by the zoomed sections of the smoothed versions of the distorted meshes depicted in Fig. 5.2. As can be seen in the case of smart Laplacian smoothing, clusters of very low quality elements remain. Here, smart Laplacian smoothing was not able to improve these clusters since this would have led to invalid elements or a deterioration of local mesh quality, which is prevented by its smoothing control mechanisms.

As can be seen in Table 5.1, even in the case of the undistorted quadrilateral and mixed mesh, applying smart Laplacian smoothing leads to a decreased worst element quality number, since smoothing control is based on the local mean mesh quality. This is reflected by the good q_{mean}-values obtained. Since global optimization is geared towards improving the mean mesh quality number by

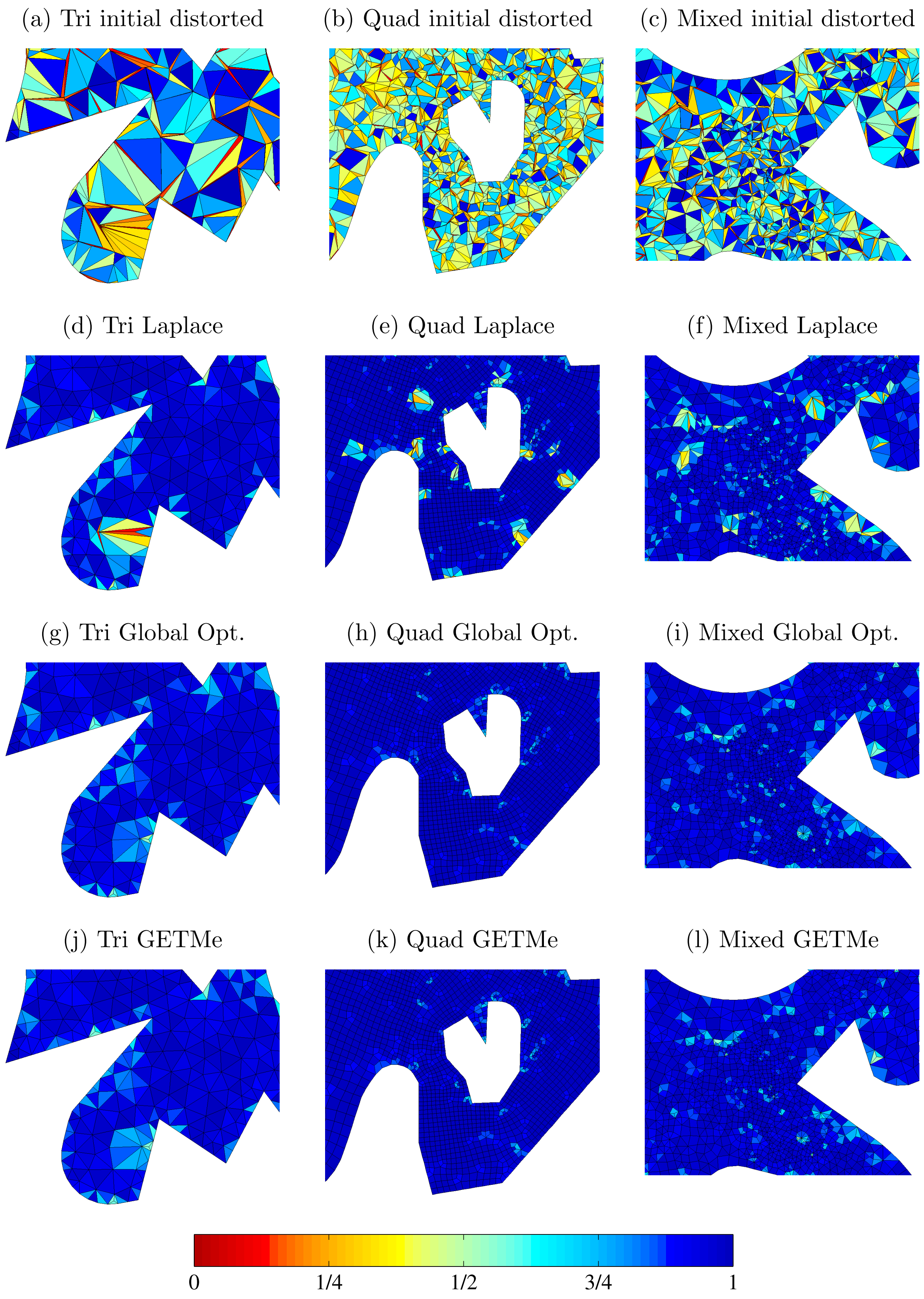


Figure 5.2: Sections of distorted initial meshes and their smoothed counterparts

mathematical optimization, the results achieved are on a high level. Although following a geometric approach, results obtained by GETMe smoothing are on a comparable level.

By comparing the results for the undistorted mesh with those for the distorted mesh, it is also visible that with respect to mesh quality global optimization as well as GETMe smoothing are particularly stable. That is, distorting the mesh does not influence the resulting mesh quality. In contrast, smart Laplacian smoothing is less robust as is reflected by the particularly low $q^*_{\min}$ values and decreased q_{mean} values in the case of the distorted meshes.

Whereas global optimization is stable with respect to mesh quality, it is not stable with respect to smoothing time. For example, this approach required 261 iterations to smooth the distorted quadrilateral mesh, whereas only 6 iterations where required in the case of the undistorted mesh. With respect to smoothing time, smart Laplacian smoothing and GETMe smoothing are more stable, that is, distorting the mesh leads only to a moderate increase of iteration numbers and smoothing time. Although GETMe smoothing incorporates a element transformation approach, the smoothing time per iteration is low, since the number of quality evaluations is lower if compared to smart Laplacian smoothing. The latter requires the quality to be evaluated for each element attached to a node before and after the node is updated, whereas element quality has to be evaluated only once per element per iteration during GETMe smoothing. In contrast, global optimization leads to a significantly increased numerical effort due to its optimization approach incorporating a feasible Newton-based optimization.

5.1.2 Tensile structure surface mesh

The second model considered is a tensile structure depicted in Fig. 5.3. It represents a pavilion roof with two peaks.

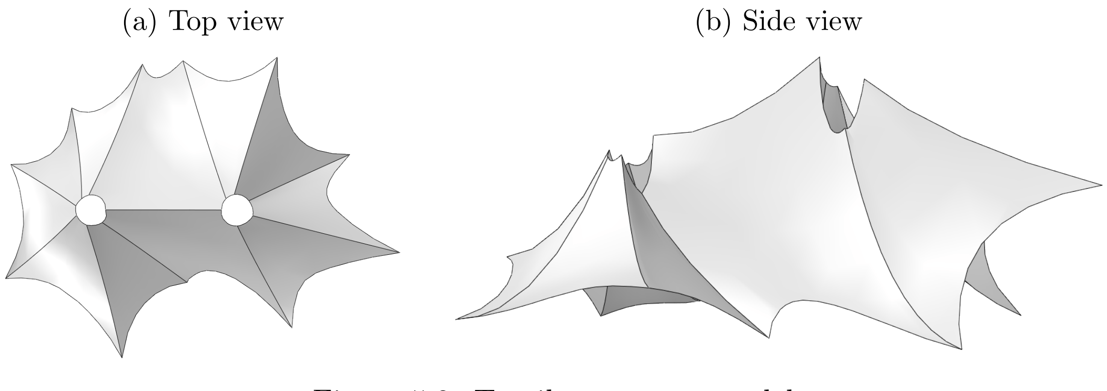


Figure 5.3: Tensile structure model

In the case of surface mesh smoothing additional techniques have to be incorporated in order to preserve the model. This includes constraint node movements for boundary nodes and nodes, which are part of feature lines given by the edges of the model. Movements of all other nodes are restricted to the model surface. This is realized by simply backprojecting new node positions onto a tessellated version of the initial mesh and its boundary and feature edges, before the iterative invalid element resetting step is applied. Both, the smart Laplacian smoothing algorithm and the GETMe smoothing algorithm have been enhanced accordingly. In contrast, the third party software Mesquite follows a different approach and would have required an extensive implementational effort in order to incorporate shape preservation techniques. Thus it is omitted in the following test.

A quadrilateral mesh consisting of 23,101 nodes and 22,676 elements has been generated. As can be seen by the mean quality number $q_{\text{mean}} = 0.9814$, the mesh is of particularly high overall quality. In addition, the minimal element quality number amounting to $q^*_{\min} = 0.2467$ is comparably good. Results of smart Laplacian smoothing and GETMe smoothing for the undistorted initial mesh as well as distorted initial mesh are provided by Table 5.2. They have been obtained by using the same test system, implementation and parameter set as described in the previous section.

Table 5.2: Tensile structure smoothing results

Method	Iter	Time (s)	$q^*_{\min}$	q_{mean}
Initial undistorted	–	–	0.2467	0.9814
Smart Laplace	11	3.05	0.3383	0.9872
GETMe	12/42100	1.73	0.4224	0.9857
Initial distorted	–	–	0.0046	0.4983
Smart Laplace	40	11.48	0.0555	0.9855
GETMe	40/63500	5.26	0.4224	0.9858

As can be seen, performing similar iteration numbers in both cases, GETMe smoothing is significantly faster compared to smart Laplacian smoothing due to the lower number of element quality evaluations. In the case of GETMe, the iteration number of the simultaneous as well as the sequential substep is given. Although the iterations of the sequential substep are high, computing times are low, since only one element is transformed per iteration. Again, GETMe leads to superior minimal element quality numbers. As in the case of the planar example, smart Laplacian smoothing is able to improve the undistorted mesh but shows significant deficiencies in the lowest quality element of the distorted mesh. This is also reflected by the low quality element clusters as depicted in Fig. 5.4.

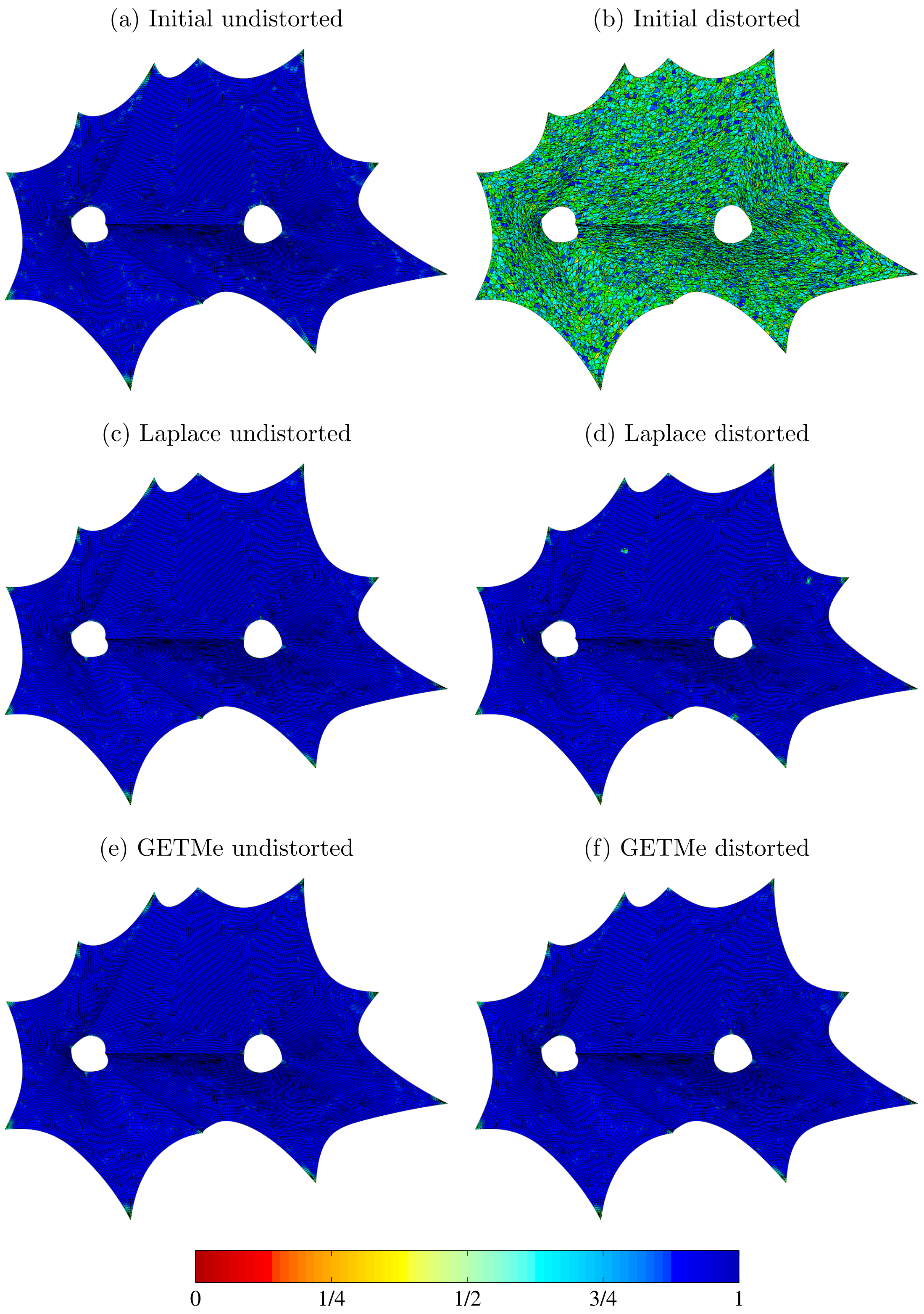


Figure 5.4: Tensile structure meshes with elements colored according to their mean ratio value

Although smart Laplacian smoothing results in particularly good mean mesh quality numbers, in this example, its instability represents a major drawback with respect to industrial applicability. In contrast, GETMe smoothing is fast and stable. That is, its results with respect to mesh quality are almost identical for both, the distorted as well as the undistorted mesh. Furthermore, the particularly good minimal element quality number is favorable in applications like the finite element method, where mesh quality has an impact on solution quality and efficiency. This will also be demonstrated by the examples presented in Chapter 6 discussing the effect of mesh smoothing on finite element solutions in more detail.

5.1.3 Prestressed concrete bridge part volumetric mesh

The part of a prestressed concrete bridge shown in Fig. 5.5 is considered as volumetric model. It has been meshed by tetrahedra resulting in 53,978 nodes and 262,201 elements. In addition, a hexahedral mesh has been generated consisting of 307,530 nodes and 260,496 elements.

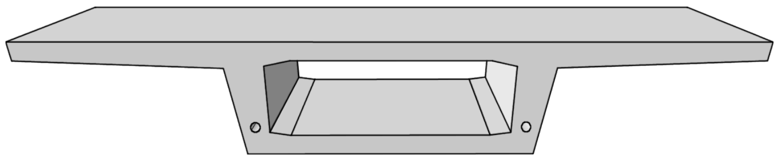


Figure 5.5: Prestressed concrete bridge part model

The quality initial meshes as well as their distorted counterparts have been smoothed by the original Laplacian smoothing approach, volume-weighted Laplacian smoothing, smart Laplacian smoothing, the global optimization-based approach, which is provided by the shape improvement wrapper of Mesquite, and the GETMe adaptive approach. For the latter the maximum number of iterations has been set to 1000 and the q_{mean} improvement tolerance to 10^{-4}. Tetrahedral elements have been transformed by using the opposite face normal transformation according to Definition 3.3. In the case of hexahedral elements, the dual element-based transformation according to Definition 3.4 has been applied. In both cases, the fixed element transformation parameter $\sigma = 3/2$ was used. The tables of relaxation values have been set to $R = (1, 1/4, 1/16, 0)$ and $R = (1/2, 1/4, 1/10, 1/100, 0)$ in the case of the q_{mean} and q^*_{min} oriented smoothing stage, respectively. Each q^*_{min} oriented smoothing cycle has been terminated after five consecutive iterations without q^*_{min} improvement. Cross sections of the distorted initial meshes and their smoothed counterparts are depicted in Fig. 5.6.

The effect of mesh distortion is obvious for the initial meshes, which are shown in Fig. 5.6a and Fig. 5.6g. The same holds for the quality improving effect of all mesh smoothing methods depicted in the other subfigures. However,

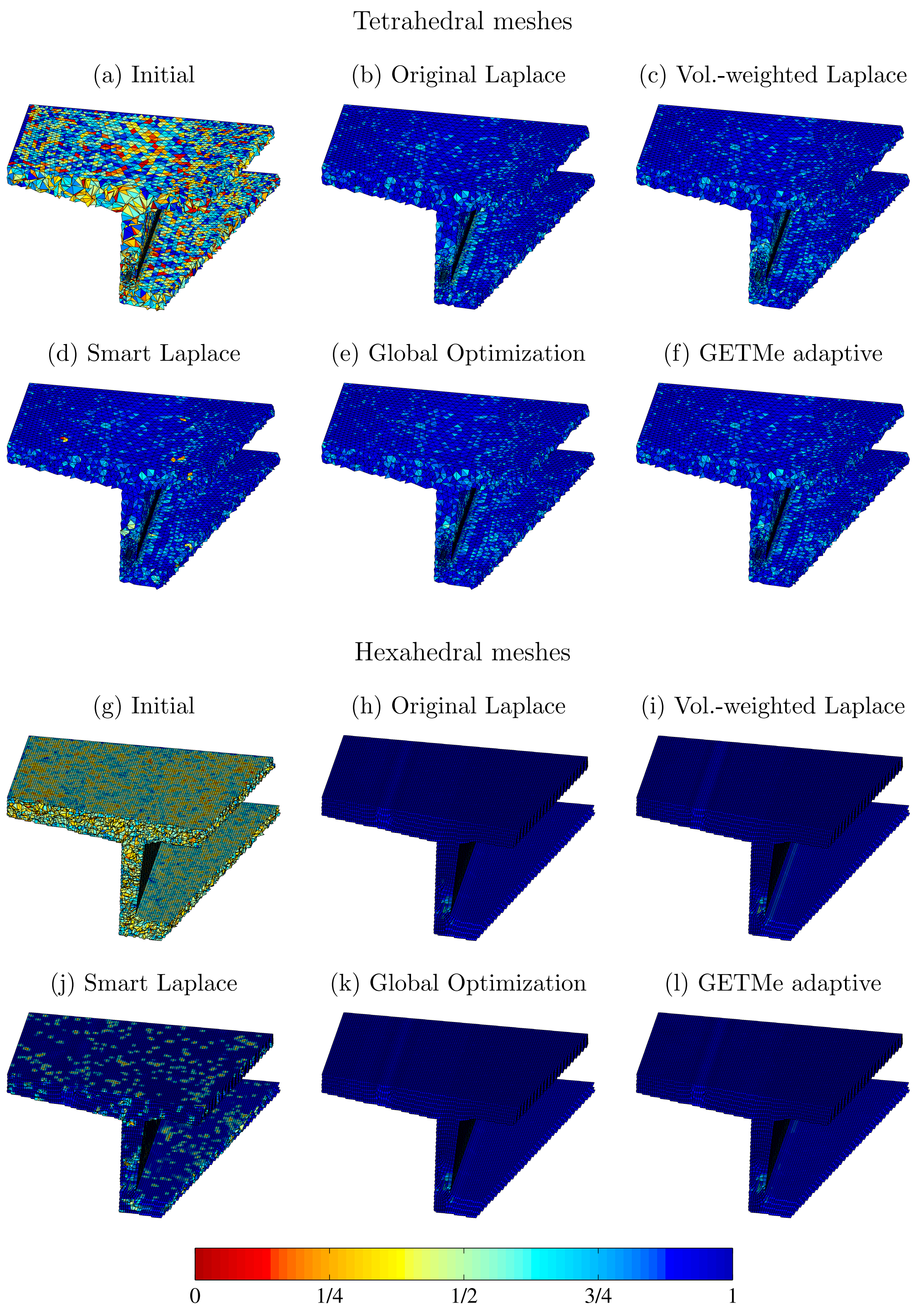


Figure 5.6: Cross sections of smoothed pretensioned concrete bridge part meshes with elements colored according to their mean ratio value

low quality clusters remain in the case of smart Laplacian smoothing due to the quality controlled restriction of node movements in order to avoid the generation of invalid elements. Although not obvious from Fig. 5.6, such elements occur in the case of original and volume-weighted Laplacian smoothing of the tetrahedral mesh. As can also be seen, due to their increased regular topology, the hexahedral meshes are of higher quality compared to the tetrahedral meshes.

Resulting smoothing times, iteration numbers, and mesh quality numbers for all meshes and smoothing methods are given in Table 5.3. These have been obtained by C++ implementations of original Laplacian smoothing, volume-weighted Laplacian smoothing and global optimization. In the case of smart Laplacian smoothing and GETMe adaptive, C implementations were applied. All programs have been compiled using the GNU Compiler Collection version 4.7.1 [162]. Computations were accomplished on a personal computer with an Intel® Core™ i7-870 CPU (quad core, 8 MB cache, 2.93 GHz), 16 GB RAM, and a 64 bit Linux operating system.

Smoothing results are provided by Table 5.3, which is divided into two sections containing the results for the tetrahedral and hexahedral meshes, respectively. Each of these sections again is divided into two blocks containing the results for the undistorted and the distorted meshes. As can be seen in the case of the tetrahedral mesh, original Laplacian smoothing and volume-weighted Laplacian smoothing led to invalid elements both for the undistorted and distorted initial mesh. However, iteration numbers of these approaches are not affected by mesh distortion. Due to local mean quality controlled node movements, smart Laplacian smoothing avoids the generation of invalid elements. Nevertheless, $q^*_{\min}$ is decreased in the case of the undistorted initial mesh or on an unacceptable level in the case of the distorted mesh. In contrast, global optimization as well as GETMe adaptive smoothing lead to superior mesh quality results.

The advantage of GETMe adaptive smoothing over global optimization becomes particularly obvious by comparing the results for the undistorted and the distorted meshes. Although quality numbers are comparable in both cases, the runtime of global optimization is significantly affected by mesh distortion, that is to say by the quality of the initial mesh. This is caused by the convergence properties of the mathematical optimization process, which depends on the quality of the start value and hence the initial mesh configuration. These effect is also particularly visible for the hexahedral mesh, where mesh distortion leads to an increase of smoothing time by factor 20.2 in the case of global optimization. In contrast, smoothing time increases only by factor 2.2 in the case of GETMe adaptive. The beneficial effect of the two stage smoothing approach of GETMe adaptive is reflected in both cases by the superior results for $q^*_{\min}$.

Table 5.3: Prestressed concrete bridge part mesh smoothing results

Mesh	Method	Iter	Time (s)	$q^*_{\min}$	q_{mean}
Tet	Initial undistorted	–	–	0.2715	0.8486
	Original Laplace	17	1.66	invalid	invalid
	Volume-weighted Laplace	17	4.38	invalid	invalid
	Smart Laplace	5	0.58	0.1910	0.8741
	Global Optimization	12	3.63	0.4791	0.8774
	GETMe adaptive	215	2.19	0.5564	0.8766
	Initial distorted	–	–	0.0002	0.4678
	Original Laplace	17	1.66	invalid	invalid
	Volume-weighted Laplace	16	4.13	invalid	invalid
	Smart Laplace	14	1.62	0.0087	0.8650
	Global Optimization	65	19.76	0.4790	0.8774
	GETMe adaptive	214	3.36	0.5498	0.8765
Hex	Initial undistorted	–	–	0.6475	0.9626
	Original Laplace	10	0.78	0.6384	0.9635
	Volume-weighted Laplace	10	11.54	0.5624	0.9634
	Smart Laplace	3	2.91	0.6463	0.9638
	Global Optimization	5	11.02	0.6783	0.9642
	GETMe adaptive	219	3.29	0.7082	0.9639
	Initial distorted	–	–	0.0472	0.4771
	Original Laplace	20	1.56	0.6211	0.9634
	Volume-weighted Laplace	15	17.30	0.5647	0.9633
	Smart Laplace	24	23.21	0.0479	0.8954
	Global Optimization	126	222.62	0.6783	0.9642
	GETMe adaptive	245	8.02	0.7096	0.9635

5.2 Tetrahedral mesh smoothing examples

In this section several examples of tetrahedral meshes smoothed by the geometric element transformation method will be given. Whereas the first example considers a rather synthetic mesh in order to give a first comparison and to discuss basic properties of the methods under consideration, a modular hip endoprosthesis mesh with a more adverse topological configuration is examined in the second example. In the third example, various graded meshes of a stub axle model are smoothed representing another real world application. Finally, results are given for a variety of meshes in order to further substantiate the potential of GETMe-based smoothing.

In the given examples, results of the following smoothing methods will be compared: Smart Laplacian smoothing, which was terminated if two consecutive q_{mean} values deviated less than 10^{-6} or neither q_{min} nor q_{mean} have been improved for the last 50 iterations. The mesh with the best q_{mean} value and the associated iteration number have been used for comparison. Global optimization-based smoothing was performed by using the shape improvement wrapper included in the mesh quality improvement toolkit Mesquite version 1.1.7 [137]. Default settings, as well as the default termination criterion, have been applied. Hence, smoothing usually stopped at a preliminary state. Therefore, applying the shape improvement wrapper has been repeated until the convergence or deterioration of the average mean ratio value. As for all methods, results of the best mesh obtained during the whole process have been taken for comparison.

The GETMe approach has been applied using the same termination criteria as in the case of smart Laplacian smoothing for the simultaneous smoothing substep. The preceding sequential substep has been terminated if 5000 consecutive element transformations neither improved q_{min} nor q_{mean} or q_{mean} degraded more than 1% compared to the initial value generated by the simultaneous substep. For all examples, the parameters of the simultaneous substep have been consistently set to $\sigma_{\text{min}} = \sigma_{\text{max}} = 40$, $\varrho = 0.1$, mean volume preserving scaling and the averaging exponent $\eta = 0$. For the subsequent sequential substep $\sigma_{\text{min}} = \sigma_{\text{max}} = 0.0001$, $\varrho = 0.75$ and edge length sum preserving scaling have been applied.

Using the smart variant in the case of Laplacian smoothing is essential, since ordinary Laplacian smoothing, where node updates are performed independent of quality improvements, often leads to meshes with inverted elements. Laplacian smoothing is popular due to its simple approach, but results in meshes of inferior quality if compared to state of the art global optimization-based methods like that provided by Mesquite. Hence, results of the latter have been used as quality benchmark with a special focus on the mean element quality q_{mean} since the optimization approach maximizes this quality number.

In the case of GETMe smoothing, even better results can be obtained by using individual parameters. Nevertheless, for the sake of simplicity and better comparability, only the unified set of parameters and termination criteria described before have been used. Furthermore, the particular choices of η in the case of the simultaneous substep and $\sigma_{\min} = \sigma_{\max}$ further simplify the method, since node averaging and scaling are performed without respect to quality, hence reducing computational complexity.

5.2.1 Sphere mesh

The first example considers a tetrahedral mesh of the unit sphere. It has been generated by Delaunay tessellation of 258 surface nodes obtained by subdividing the faces of an octahedron three times with subsequent back-projection on the sphere and 346 inner nodes representing a distorted regular grid. The resulting initial tetrahedral mesh consists of 3108 elements of which 1745 elements do not intersect the boundary. On average, each node has 20.6 incident tetrahedra, where the number of incident elements varies within a range of 4 to 42. The average number of neighbors per tetrahedron amounts to 72.0 within a range of 33 to 112. Due to the smoothing approach, topology is never changed and boundary nodes have been kept fixed, hence these numbers are preserved.

The complete model and the lower hemisphere of the initial mesh are depicted in the upper row of Fig. 5.7. Thereby, each tetrahedron E_j is colored according to its mean ratio quality number $q(E_j)$. As can be seen by the colorbar, which is depicted below the initial meshes, reddish colors indicate elements of bad quality, whereas bluish colors mark elements of good quality. The lower hemispheres of the meshes obtained by applying smart Laplacian smoothing, global optimization, and GETMe using the default configuration described in the beginning of this section are depicted in the lower row of Fig. 5.7.

Table 5.4: Sphere mesh quality statistics

method / criterion	$q_{\min}$	q_{mean}
Initial	0.0010	0.4888
Smart Laplace	0.0010	0.5915
Global Optimization	0.3545	0.7657
GETMe	0.4177	0.7701

As can be seen by the resulting quality numbers given in Table 5.4, the geometry-based approach of smart Laplacian smoothing only leads to a moderate improvement of the mean element quality number q_{mean}. Furthermore, this approach fails to improve the worst element quality number $q_{\min}$, which is a characteristic weakness of smart Laplacian smoothing. Since the global optimization

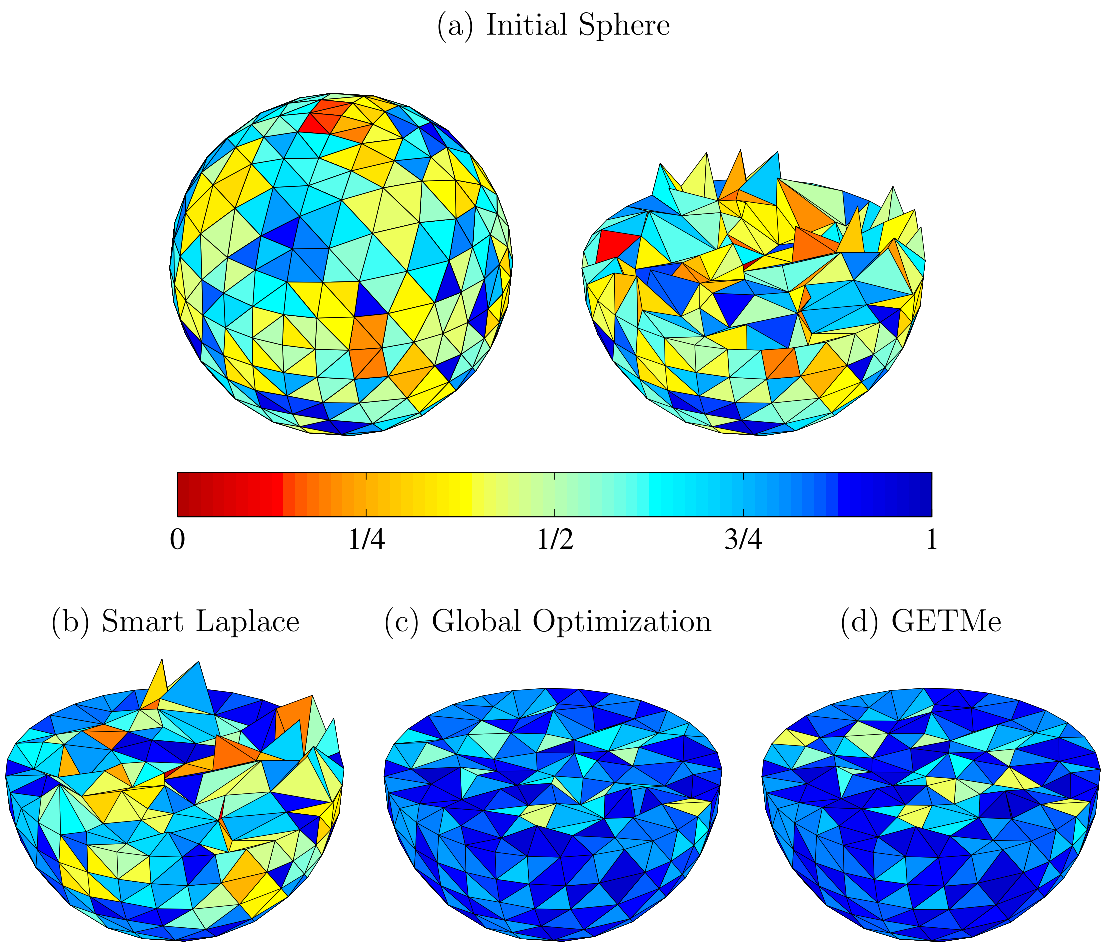


Figure 5.7: Complete initial sphere mesh and its lower hemisphere (a) with elements colored according to their mean ratio quality number and smoothed meshes (b)–(d)

approach is geared towards improving q_{mean} the mean element quality is considerably improved. Nevertheless, the average mean quality number 0.7779 obtained by the simultaneous substep of the GETMe approach is slightly larger. Subsequently applying the sequential GETMe substep in order to further improve the worst element quality number leads to a slight decrease of q_{mean}, since the transformation of low quality elements also affects the quality numbers of neighboring elements. Hence, the transformation parameters $\sigma_{\min} = \sigma_{\max} = 0.0001$ have been chosen conservatively in order to obtain a moderate change of geometry. As can be seen by the final results of the geometry-based GETMe approach given in the last row of Table 5.4, this leads to convincing results for $q_{\min}$ as well as q_{mean}.

In the case of global optimization-based smoothing a total of 41 feasible Newton iterations have been performed within 3 calls of the shape improvement wrapper of Mesquite. Smart Laplacian smoothing terminated after 26 iterations in which 239 node updates had to be discarded in order to avoid the generation of invalid elements. In contrast, only 6 updates which would have led to invalid elements have been discarded in the case of GETMe smoothing. However, the iteration numbers 168 of the simultaneous substep and 159,800 of the sequential substep are significantly higher if compared to Laplacian smoothing. Thereby, the number of element transformations applied during the sequential substep corresponds to that performed within about 51 steps of simultaneous smoothing approach. This occurs because a tight tolerance for termination control has been used and mesh quality increases rather moderately after a steep ascent during the first iterations.

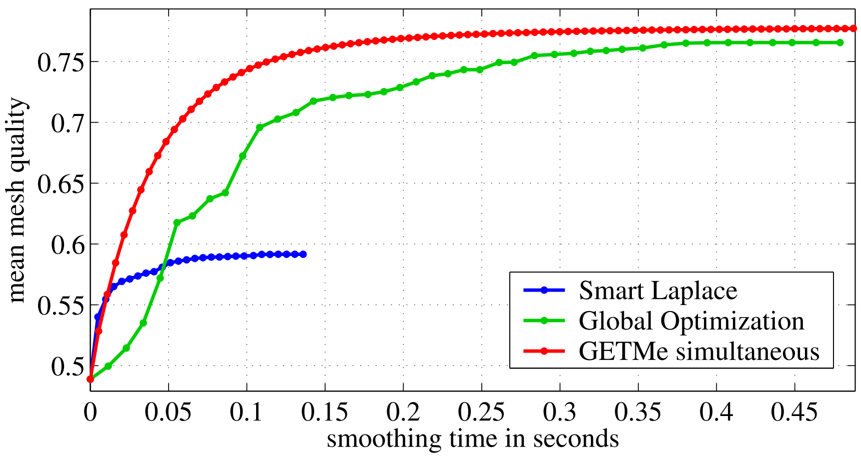


Figure 5.8: Sphere mesh mean quality improvement with respect to smoothing time

This is depicted in Fig. 5.8 showing the mean mesh quality q_{mean} with respect to the accumulated smoothing time in seconds. Each marker represents the results after an iteration of the associated smoothing method, or, in the case of the first marker, the initial mean mesh quality 0.4888. Only the iteration times of the main smoothing loop of each program has been recorded in order diminish the runtime tampering caused for example by different file i/o strategies, file formats, and mesh initialization procedures.

A straightforward C++ implementation of GETMe smoothing and smart Laplacian smoothing is compared to the results of the feasible Newton based global optimization approach of Mesquite, which is also implemented in C++. Runtimes have been measured on a notebook with an Intel® Core$^{\mathrm{TM}}$2 Duo CPU T7250 (2MB cache, 2.00GHz, 800MHz FSB), 2GB RAM, Linux operating system with kernel 2.6.22.19, and the GNU C++ compiler version 4.2.1. Since smart Laplacian smoothing uses the same data structures as GETMe smoothing, both methods resulted in a peak memory usage of 3.9MB during smoothing the sphere example. To simplify matters, the data structure stores redundant proximity information to a large extent. Hence, the memory consumption of this first implementation is larger than the 2.4MB used by the global optimization-based approach of Mesquite.

The closer the iteration markers in Fig. 5.8 the faster the iterations of a smoothing method. For the given example, iteration runtimes average to 0.0052s, 0.0054s, and 0.0117s in the case of smart Laplacian smoothing, GETMe simultaneous smoothing, and global optimization respectively. In contrast, the total smoothing runtime of GETMe simultaneous amounts to 0.91s caused by the large number of iterations, to 0.48s in the case of global optimization and to 0.14s in the case of smart Laplacian smoothing.

Nevertheless, due to the efficiency of the first smoothing steps, GETMe simultaneous smoothing reaches or exceeds the final q_{mean} value of smart Laplacian smoothing already after 4 iterations being 6.1 times faster and the final q_{mean} value of global optimization after 33 iterations being 2.5 times faster if compared to the respective runtimes. Although the following 135 simultaneous GETMe iterations continuously further increased q_{mean} due to the tight tolerance, the overall q_{mean} improvement of these following steps only amounts to 1.5%. Hence, in practice alternative termination criteria avoiding such inefficient iterations can lead to similar results within a fraction of the number of GETMe simultaneous steps accomplished in this and the following examples.

The subsequent applied sequential GETMe smoothing substep took in total 4.0s. Hence, sequentially transforming the same number of elements as within one step of the simultaneous approach took about 0.0778s. Although being simpler, the sequential approach is therefore slower due to the min heap handling in order to determine the worst quality element after each iteration and the comparably expensive mesh quality assessment after 100 iterations each. Furthermore, since the transformation parameters are chosen conservatively in order to be applicable to a broad range of meshes, this results in a slow $q_{\min}$ convergence. A speedup of factor 10.5 could for example be achieved by setting $\sigma_{\min} = \sigma_{\max} = 0.001$ resulting in 16,000 sequential iterations with a total runtime of 0.4s and a similar worst element quality of $q_{\min} = 0.4170$.

The element quality histograms for the sphere meshes are depicted in Fig. 5.9. As can be seen, smart Laplacian smoothing (blue markers) leads only to a moderate change in the quality distribution, if compared to the initial mesh marked black. In contrast, the global optimization approach (green markers) and the GETMe approach (red markers) have massive impact on the histograms. Thereby, for larger quality numbers the histogram of GETMe mainly reflects the results of the simultaneous substep, whereas for smaller numbers it is mainly af-

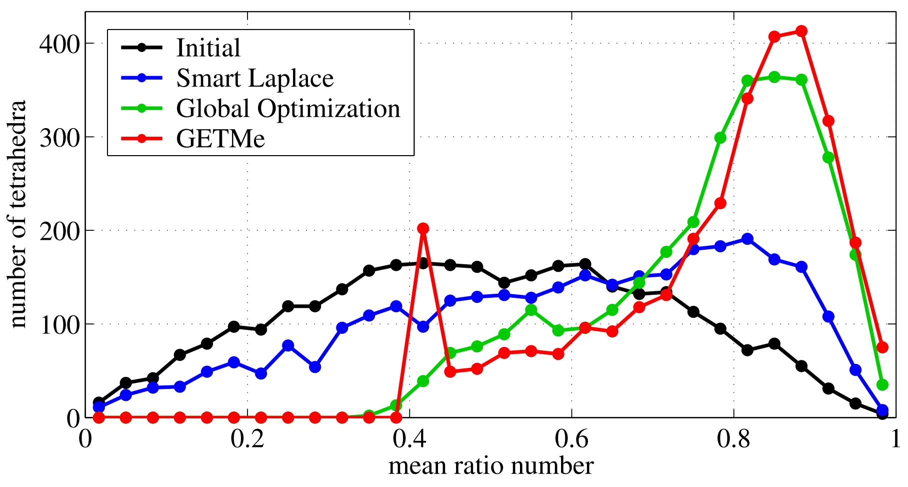


Figure 5.9: Sphere mesh element quality histogram

fected by the sequential substep. In particular, the peak of elements with quality numbers near $q_{\min}$ is characteristic for the sequential substep due to the approach of successively gently improving the worst elements, which leads to this accumulation.

5.2.2 Modular hip endoprosthesis

The second tetrahedral mesh example considers the modular hip endoprosthesis depicted on the left of Fig. 5.10. It was developed by NIKI Ltd. [163, 164] within the research project SKELET funded by the 3rd European Framework program. The complete initial mesh used for smoothing is depicted in the middle of Fig. 5.10 and a cross section with elements colored according to their mean ratio quality number is depicted on the right.

The mesh consists of 2669 nodes and 13,192 tetrahedral elements. Again, each of the 1228 boundary nodes has been kept fixed during the smoothing process. On average, each node has 19.8 adjacent tetrahedra where the numbers range from 2 to 56. The average number of neighbors per tetrahedron amounts to 72.5 within a range from 16 to 140. In contrast to the sphere example of the previous subsection, random inner nodes have been used instead of regular inner nodes. This results in more adverse topological configurations, which impose a problem for pure smoothing methods. The improved meshes obtained by smart Laplacian smoothing, the global optimization-based approach and GETMe are depicted in Fig. 5.11.

In the case of GETMe 243 iterations of simultaneous smoothing have been applied first. In the subsequent 253,400 sequential smoothing steps have been

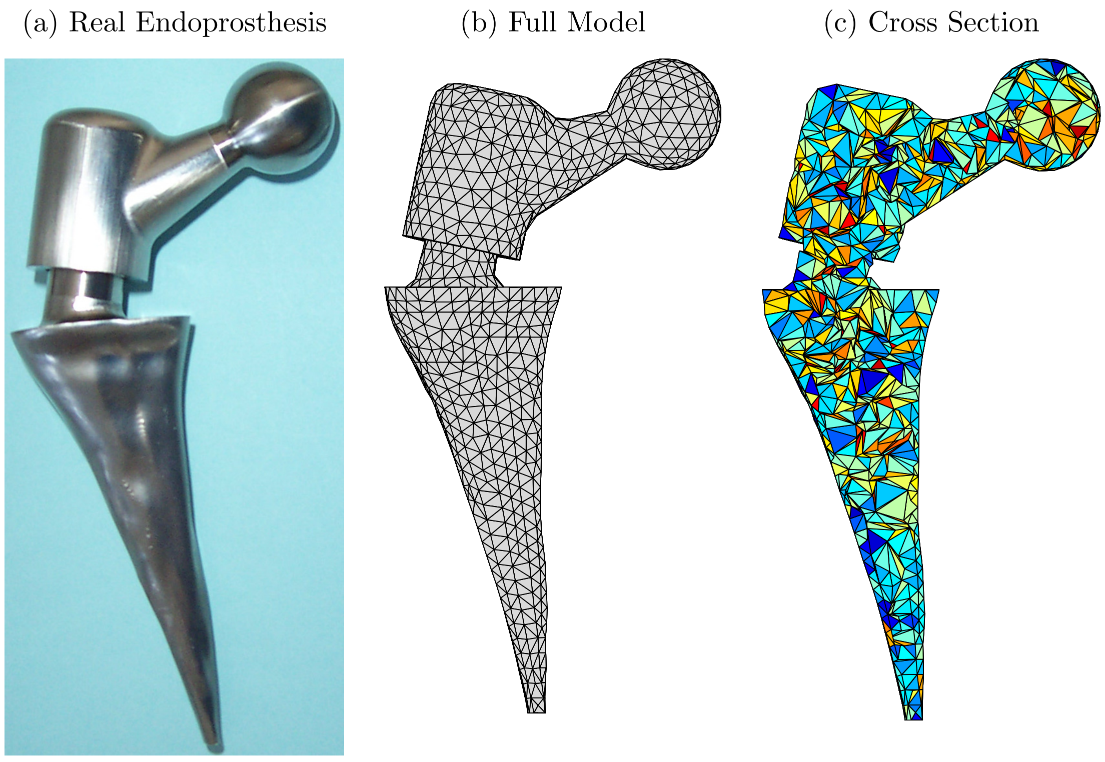


Figure 5.10: Modular hip endoprosthesis and initial model

performed. The resulting quality numbers are given in Table 5.5. Since the mesh contains fixed low quality elements consisting of four boundary nodes, the quality number q^*_{min} of the worst tetrahedral element with at least on inner node is given instead of q_{min}. That is, q^*_{min} represents the quality number of the worst improvable element. In the case of smart Laplacian smoothing q^*_{min} is smaller than the minimal fixed element quality number given by 0.0253. In contrast, for GETMe and global optimization-based smoothing it holds that $q^*_{\text{min}} > q_{\text{min}}$. In particular, this implies that the overall minimal quality numbers q_{min} obtained by these two methods are the same.

Table 5.5: Modular hip endoprosthesis mesh quality statistics

method / criterion	q^*_{min}	q_{mean}
Initial	0.0010	0.4748
Smart Laplace	0.0022	0.5640
Global Optimization	0.2265	0.7354
GETMe	0.2353	0.7433

Compared to the 15 iterations performed by smart Laplacian smoothing and

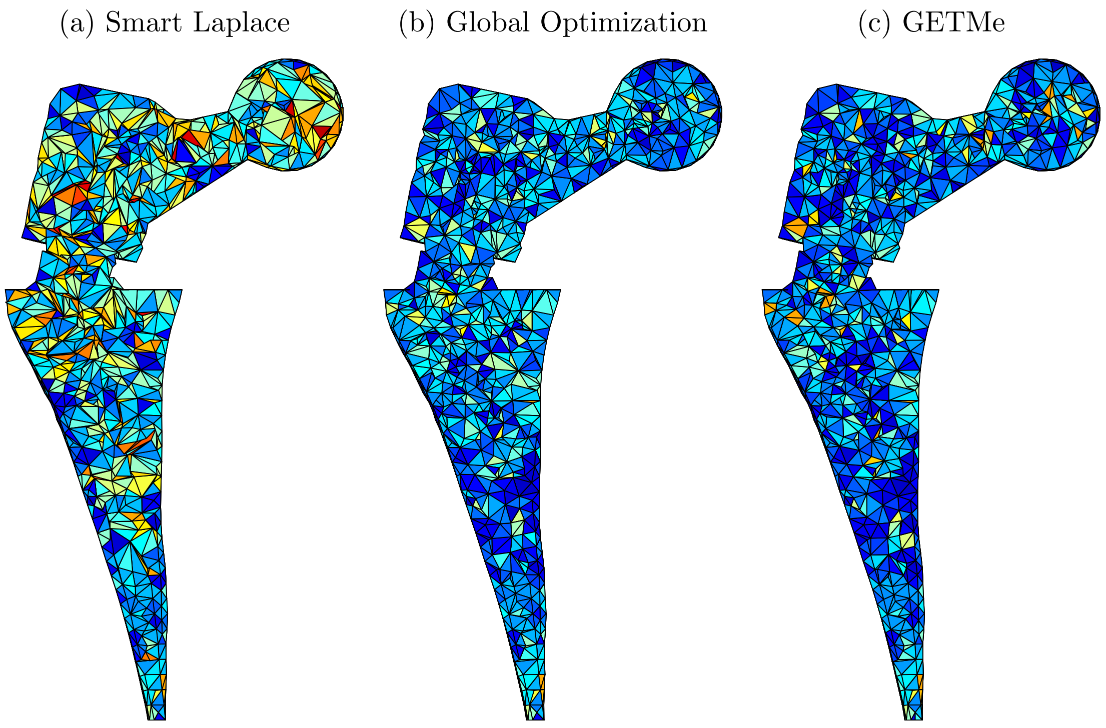


Figure 5.11: Smoothed modular hip endoprosthesis models

the 52 iterations of the feasible Newton approach, the number of iterations needed in the case of the simultaneous GETMe substep is significantly higher. Nevertheless, once again the first few steps of GETMe smoothing are highly efficient. For example 3 steps of the simultaneous GETMe substep suffice to yield the mean quality number 0.5649, which is better than the final result achieved by smart Laplacian smoothing, and 16 iterations result in $q_{\text{mean}} > 0.7$. The following steps are of decreasing efficency as is depicted in Fig. 5.12. The number of element transformations performed in the subsequent sequential GETMe smoothing substep equals that of about 19 simultaneous iterations.

The average smoothing iteration runtimes amount to 0.0285s, 0.0435s, and 0.0499s in the case of smart Laplace, GETMe simultaneous, and global optimization using a peak memory of 12.4MB and 5.7MB respectively. At this, GETMe simultaneous achieves the final q_{mean} results 3.0 and 1.6 times faster than smart Laplacian smoothing and global optimization respectively. In total, simultaneous GETMe took 10.57s and the subsequent sequential approach 6.02s.

All numbers refer to non-parallelized implementations. Compared to this, a first simple parallelized version of GETMe simultaneous obtained by the OpenMP [160] directive "`#pragma omp parallel for`" applied to major element loops, like the element transformation and node averaging loops, resulted in a runtime

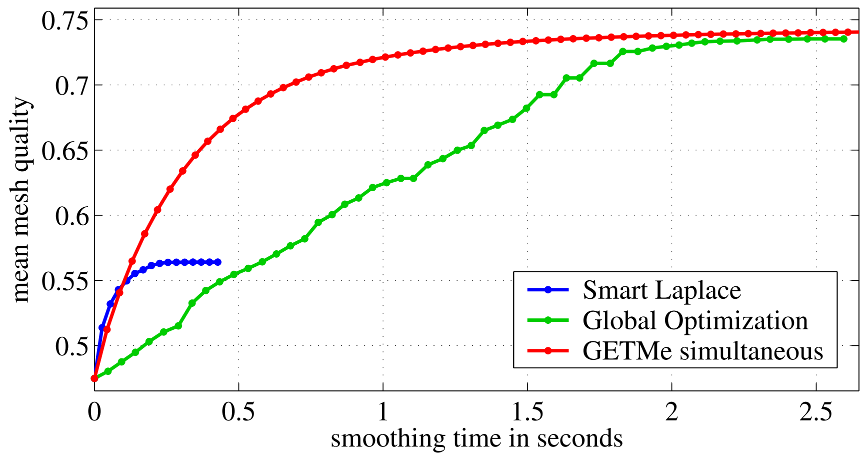


Figure 5.12: Modular hip endoprosthesis quality improvement with respect to smoothing time

of 6.95s. This already yields a good speedup factor of 1.52 on the test system named before with a theoretical speedup limit of 2. Hence GETMe smoothing offers a good potential for parallelization techniques.

On average 17.1 invalid elements per iteration step occurred during smart Laplacian smoothing. Due to the adverse topological configurations, invalid element removal techniques had also to be applied in the case of the simultaneous GETMe smoothing substep. However, the average number 3.8 of invalid elements per iteration step is significantly lower. Furthermore, the subsequent sequential smoothing approach did not lead to any degenerated elements.

Again, smart Laplacian smoothing led only to insufficient improvements for both quality numbers. This can also be seen by the quality histogram depicted in Fig. 5.13. In particular, the remaining number of low quality elements prohibits the use of such meshes for finite element applications. In contrast, the results obtained by two cycles of the shape improvement wrapper of Mesquite are of high quality both with respect to the worst and mean quality number. Nevertheless, GETMe is able to further improve both numbers.

As can be seen by the quality table and the histogram, GETMe leads to a better value for $q^*_{\min}$, but also to a higher number of elements with quality numbers near this value if compared to the global optimization results. This comes due to the fact that the worst element and deadlock handling of GETMe sequential was geared towards aggressively improving the worst element leaving all elements with higher quality numbers unimproved. Alternative control mechanisms could be applied, which improve elements within a broader range around $q^*_{\min}$. However, this usually comes at the expense of a higher computational complexity and a further degradation of the high quality mean number obtained by the simulta-

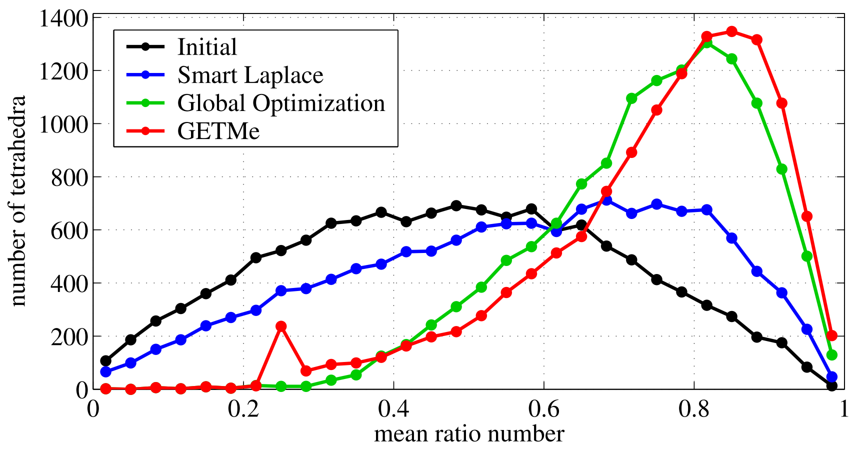


Figure 5.13: Modular hip endoprosthesis element quality histogram

neous GETMe substep, since successively transforming low quality elements also affects the quality of the neighboring elements.

5.2.3 Stub axle

Further aspects of the presented smoothing approaches will be analyzed by various meshes of the stub axle model depicted on the left of Fig. 5.14. The STEP file of the underlying geometry model is provided courtesy of INPG by the AIM@SHAPE shape repository [165]. The model of genus 17 consists of 249 faces bounded by spline curves. On the left of Fig. 5.14 these faces are marked by different colors.

Three graded meshes with 110,524, 274,607, and 400,128 tetrahedral elements have been generated using the NETGEN mesh generator version 4.9.9 [166], of which the finest is depicted on the right of Fig. 5.14. The surface meshes have only been slightly improved resulting in various low quality elements near the boundary. Since boundary nodes are kept fixed, this makes smoothing more difficult. In addition, the volume mesh optimizer provided by NETGEN has not been applied. The ratio of the maximal and minimal average edge length of all tetrahedra amount to 660, 1140, and 441 respectively for the three meshes. Furthermore, in all three cases the maximal and minimal element volume ratios are of order 10^{10}. Hence, all meshes are strongly graded.

In addition to the three initial meshes generated by NETGEN, also distorted variants have been generated and smoothed. That is, for each initial mesh, nodes have been randomly moved preserving the validity of elements. This yields distorted initial meshes of lower quality with a different geometry, but the same topology if compared to their undistorted counterparts. Investigating the differ-

(a) Stub axle model (b) Tetrahedral mesh

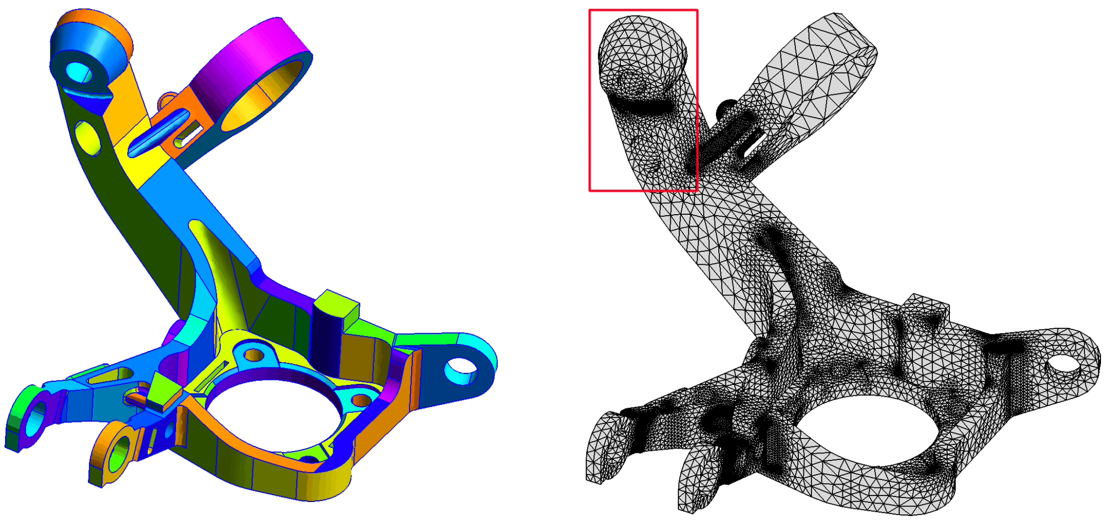


Figure 5.14: Stub axle model and graded mesh with 400,128 elements

ences of the smoothing results for the undistorted as well as the distorted meshes gives an indication of geometrical robustness of the smoothing methods.

Table 5.6: Stub axle meshes smoothing results

Mesh		Initial		Smart Laplace		Global Opt.		GETMe	
tets	dist.	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}
110,524	no	0.0003	0.6792	0.0025	0.7169	0.0311	0.7296	0.0293	0.7306
110,524	yes	0.0010	0.5350	0.0037	0.6884	0.0311	0.7296	0.0294	0.7300
274,607	no	0.0157	0.6890	0.0057	0.6975	0.0169	0.7058	0.0174	0.7132
274,607	yes	0.0003	0.4890	0.0004	0.6498	0.0169	0.7058	0.0173	0.7105
400,128	no	0.0448	0.7175	0.0000	0.7667	0.2089	0.7772	0.2680	0.7806
400,128	yes	0.0001	0.5284	0.0002	0.7481	0.2089	0.7772	0.2678	0.7805

Results for all six initial meshes and the three smoothing approaches are given in Table 5.6. In this, the first column gives the number of tetrahedral elements for each mesh. The second column indicates either if the initial mesh has been additionally distorted (yes) or not (no). As can be seen by the quality numbers of the initial meshes, distorting leads to a significant decrease of the mean element quality number q_{mean}, but not necessarily to a deterioration of the quality number q^*_{min} of all tetrahedra with at least one inner, hence modifiable node. The latter is used instead of the overall minimal element quality number q_{min} since completely fixed boundary elements exist. In all cases, GETMe smoothing yields the best mean quality number if compared to smart Laplacian smoothing and the global optimization approach.

As can also be seen by the quality numbers given in Table 5.6, distorting the mesh not only has an impact on the quality numbers of the initial mesh, but also on the results obtained by smart Laplacian smoothing. In all cases, the mean element quality number deteriorates remarkably, whereas the minimal element quality number has been improved in two cases. In contrast, distorting the initial mesh has no impact at all on the results of the global optimization-based approach, and only a slight impact on the results of GETMe smoothing.

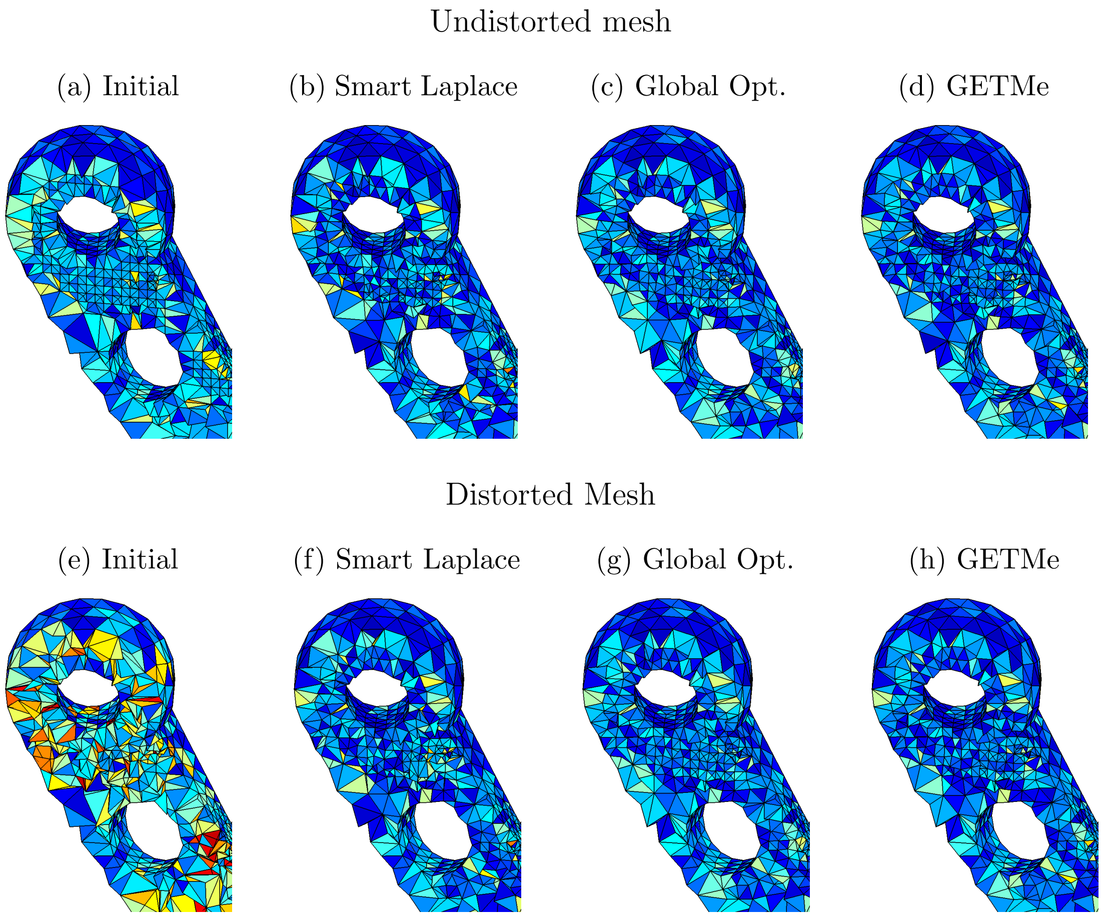


Figure 5.15: Cross sections of initial and smoothed stub axle meshes in the case of the undistorted initial mesh (upper row) and the distorted initial mesh (lower row)

This can also be seen by the cross sections of the meshes resulting for the stub axle part marked by a red rectangle on the right of Fig. 5.14. Such cross sections for the initial meshes with 400,128 elements as well as that obtained by all smoothing methods are depicted in Fig. 5.15. Whereas the meshes of smart

Laplacian smoothing for the undistorted as well as the distorted mesh partly differ considerably, in the case of global optimization and GETMe smoothing one mesh resembles the other. This indicates that, even using a geometry-based approach, GETMe smoothing is significantly less interference-prone with respect to geometrical changes than smart Laplacian smoothing.

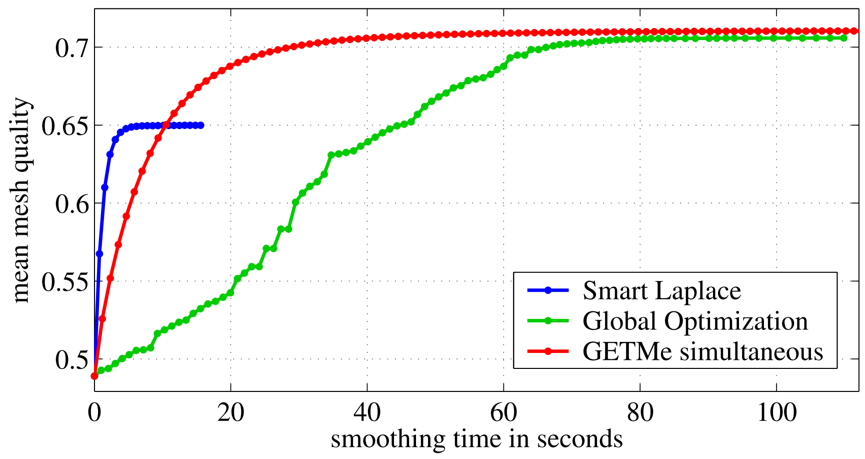


Figure 5.16: Distorted mid size stub axle mean quality improvement with respect to smoothing time

Fig. 5.16 shows exemplarily the mean quality development with respect to smoothing runtime for the distorted stub axle model with 274,607 elements. Smart Laplacian smoothing terminated after 20 iterations each taking on average 0.7792s, global optimization after 95 iterations with an average runtime of 1.1566s and a peak memory usage of 87.9MB. In contrast, GETMe simultaneous terminated after 127 iterations with an average runtime of 1.1718s and a peak memory usage of 229.4MB. The subsequent GETMe simultaneous approach performed 22,900 iterations within 3.62s.

5.2.4 Mesh variety test

In order to further substantiate the high mesh quality obtained by GETMe based smoothing, the method has also been applied to the dozen tetrahedral meshes used by Klingner and Shewchuk in [12, 167]. The initial meshes range from very low quality random meshes to high quality meshes generated by state of the art mesh generators, as well as from small meshes of about 1000 tetrahedral elements to large meshes of about 100,000 tetrahedral elements. Four of these meshes are depicted in Fig. 5.17.

The initial quality values as well as the results obtained by global optimization-based and GETMe smoothing are given in Table 5.7. The columns "Improve-

(a) TFire (b) Rand1 (c) Tire (d) StGallen

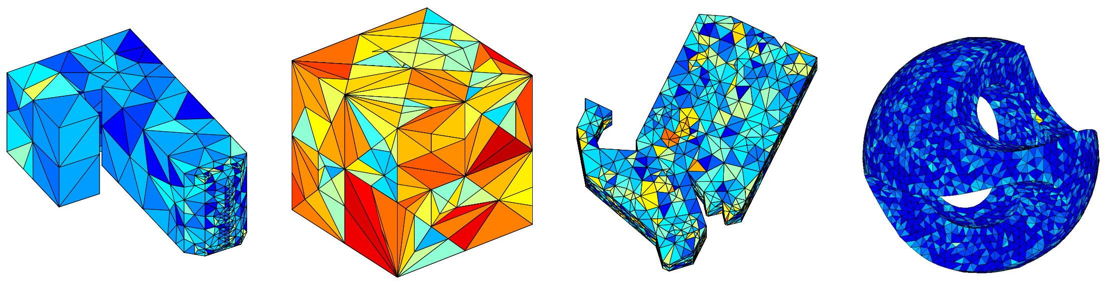


Figure 5.17: Selected initial meshes by Klingner and Shewchuk

ment" contain the relative improvement with respect to the initial values in percent. Since Klingner and Shewchuk use a topology modifying approach to improve the worst tetrahedra, results cannot be compared to those given in their publication. Furthermore, the meshes contain low quality elements with no inner nodes. These cannot be fixed by boundary nodes preserving smoothing methods as used here. Hence, again the quality number $q^*_{\min}$ of the worst element with at least one inner node is given instead of the overall worst element quality number $q_{\min}$.

As expected, the rates of improvement achieved by both methods depend on the quality of the initial meshes as well as their topological configuration. In particular, mesh quality can even decrease in the event of high quality initial meshes, as can be seen by the global optimization results for the meshes Cube1k or Cube10k. This comes due to the fact, that the objective function used is geared towards optimizing q_{mean} instead of the worst element quality $q^*_{\min}$. Consequently, the mean mesh quality has been improved in all cases.

Because of the combined approach of GETMe smoothing applying sequential smoothing as a final step, the worst element quality is improved in all cases. However, due to the termination criteria this comes at the expense of a q_{mean} deterioration up to 1% of its initial value. This is also the reason why the mean quality number q_{mean} decreases for the meshes Cube1k and Cube10k, if compared to the initial mesh. The best mean mesh quality numbers obtained by the simultaneous GETMe substep amount to 0.8879, 0.8944, 0.9047, and 0.9007 in the case of Cube1k, Cube10k, Dragon, and StGallen respectively. Hence, the simultaneous GETMe approach leads in nine of twelve cases to equal or greater mean quality numbers if compared to the global optimization approach. However, since the final GETMe mean quality numbers differ in none of the twelve cases by more than 0.01 from the values obtained by global optimization, both methods yield basically similar results with respect to q_{mean}.

Since unified parameters have been used in all cases, this demonstrates the

Table 5.7: Mean ratio numbers and improvements for different tetrahedral meshes and smoothing methods

Mesh		Initial		Global Opt.		Improvement		GETMe		Improvement	
name	tetrahedra	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}	q^*_{min}	q_{mean}
P	926	0.0795	0.7573	0.3656	0.7630	359.75 %	0.75 %	0.3942	0.7655	395.68 %	1.08 %
TFire	1104	0.4049	0.7595	0.4894	0.7884	20.87 %	3.81 %	0.5224	0.7861	29.02 %	3.50 %
Cube1k	1184	0.6542	0.8875	0.6467	0.8879	-1.15 %	0.05 %	0.7655	0.8796	17.00 %	-0.89 %
House2	1389	0.0980	0.7645	0.3695	0.7723	276.95 %	1.02 %	0.4010	0.7738	309.12 %	1.21 %
Rand1	5104	0.0231	0.4128	0.1683	0.4855	628.85 %	17.59 %	0.1999	0.4938	765.67 %	19.61 %
Tire	11,098	0.0442	0.7413	0.2801	0.8109	534.11 %	9.39 %	0.3611	0.8110	717.57 %	9.40 %
Cube10k	11,660	0.5800	0.8938	0.5484	0.8944	-5.46 %	0.06 %	0.7087	0.8935	22.18 %	-0.04 %
Rand2	25,704	0.0076	0.3345	0.0828	0.4247	986.21 %	26.96 %	0.1022	0.4278	1240.69 %	27.90 %
Dragon	32,959	0.3588	0.8962	0.4867	0.9047	35.65 %	0.95 %	0.4886	0.9046	36.17 %	0.95 %
Cow	42,053	0.3831	0.8227	0.4104	0.8478	7.13 %	3.05 %	0.4645	0.8470	21.25 %	2.96 %
StGallen	50,391	0.3501	0.8897	0.4878	0.9005	39.31 %	1.21 %	0.5663	0.9004	61.74 %	1.20 %
StayPuft	102,392	0.0843	0.8128	0.2481	0.8256	194.24 %	1.58 %	0.3347	0.8242	297.00 %	1.41 %

broad applicability and power of GETMe smoothing. In addition, even better results can be obtained by using individually adjusted parameter sets or by advanced combined approaches consisting of not only one GETMe simultaneous and GETMe sequential cycle but multiple ones. However, the results also show that in order to be applicable for arbitrary meshes, smoothing methods should be combined with topology modifying approaches as described for example in [12, 99, 168].

5.3 Hexahedral mesh smoothing examples

In this section, results obtained by applying GETMe smoothing will be presented for a selection of all-hexahedral meshes covering different aspects. This includes a nonuniform mesh with regular topology, an unstructured mesh generated by element subdivision, a mesh obtained by sweeping techniques, as well as a complex real world mesh taken from a CFD application.

Results will be compared to those of smart Laplacian smoothing. In order to be independent of the node numbering scheme, new node positions are sequentially computed and assessed but stored separately and applied simultaneously afterwards. The algorithm is terminated if two consecutive mean mesh quality values deviate less than 10^{-6}. The mesh with the best mean mesh quality is used for comparison. Results obtained by the shape improvement wrapper of the mesh quality improvement toolkit Mesquite version 1.1.7 [137] additionally serve as a quality benchmark. Since the wrapper returns the mesh of the last iteration, which is not necessarily the best mesh, all intermediate mesh qualities generated within one wrapper call have been evaluated. As in the case of smart Laplacian smoothing, the mesh with the best overall mean quality is used for comparison.

In the case of GETMe smoothing, a unified parameter set has been used for all examples in order to demonstrate the general applicability of this approach. It has been determined by assessing the results obtained by a systematic parameter variation using various meshes resulting from different mesh generation approaches. For the simultaneous GETMe substep these parameters are given by a quality dependant choice of $\sigma \in [0.5, 0.6]$, average element edge length preserving element scaling, the relaxation coefficient $\varrho = 0.75$, and the exponent $\eta = 0.5$. The same termination criterion has been used as in the case of smart Laplacian smoothing. For the subsequently applied sequential GETMe smoothing substep parameters have been set to the quality independent factor $\sigma = 0.01$, average element edge length preserving scaling, and the relaxation parameter $\varrho = 0.01$. Quality penalty difference values have been set to $\Delta\pi_i = 0.0002$, $\Delta\pi_r = 0.0004$, and $\Delta\pi_s = -0.0002$, and mesh quality has been assessed after each hundredth sequential GETMe step. Smoothing was terminated if $q_{\min}$ did not improve within the last ten evaluations.

5.3.1 Nonuniform grid

The first example considers a simple structured nonuniform mesh with nodes obtained by taking the Cartesian product of 13 non-equidistant x-values, 4 equidistant y-values, and 11 non-equidistant z-values. This results in the initial mesh depicted in the upper row of Fig. 5.18 consisting of 572 nodes and 360 hexahedral elements with quality numbers ranging from 0.0010 to 0.5714. The initial mean mesh quality amounts to 0.0943. Due to the regular topology of the mesh each of the 198 inner nodes is shared by exactly eight adjacent elements.

Cross sections of the initial mesh as well as the smoothed meshes are depicted in the lower two rows of Fig. 5.18. As can be seen on the right of the middle row, smart Laplacian smoothing leads to a rather moderate change in geometry and with that in quality. In contrast, the global optimization-based approach leads to an unnatural prolongation of the inner elements with respect to the y-axis as can be seen on the left of the lower row. A cross section of the GETMe smoothed mesh using the unified parameter set is depicted on the lower right. Quality numbers of all meshes are given in Table 5.8.

Table 5.8: Nonuniform grid smoothing results

Method / Criterion	q_{min}	q_{mean}
Initial	0.0010	0.0943
Smart Laplace	0.0059	0.1083
Global Optimization	0.0054	0.1199
GETMe	0.0204	0.1199

In the case of smart Laplacian smoothing q_{mean} became maximal for the last of the 30 iterations performed in total, whereas global optimization yielded the best results after 17 iterations in the first call of the shape improvement wrapper performing a total of two cycles and 51 iterations. In the case of GETMe smoothing 35 iterations of the simultaneous approach and 1000 iterations of the sequential approach have been performed. Since the sequential approach transforms only one element per iteration the computational effort is significantly lower compared to that of one simultaneous GETMe iteration transforming all elements. Thus, the total number of element transformations performed during all sequential GETMe smoothing iterations corresponds to the number of element transformations performed in about 2.8 iterations of the simultaneous approach.

GETMe smoothing can even lead to better results if individual parameters are used. For example, by using adjusted parameters for the sequential substep, the

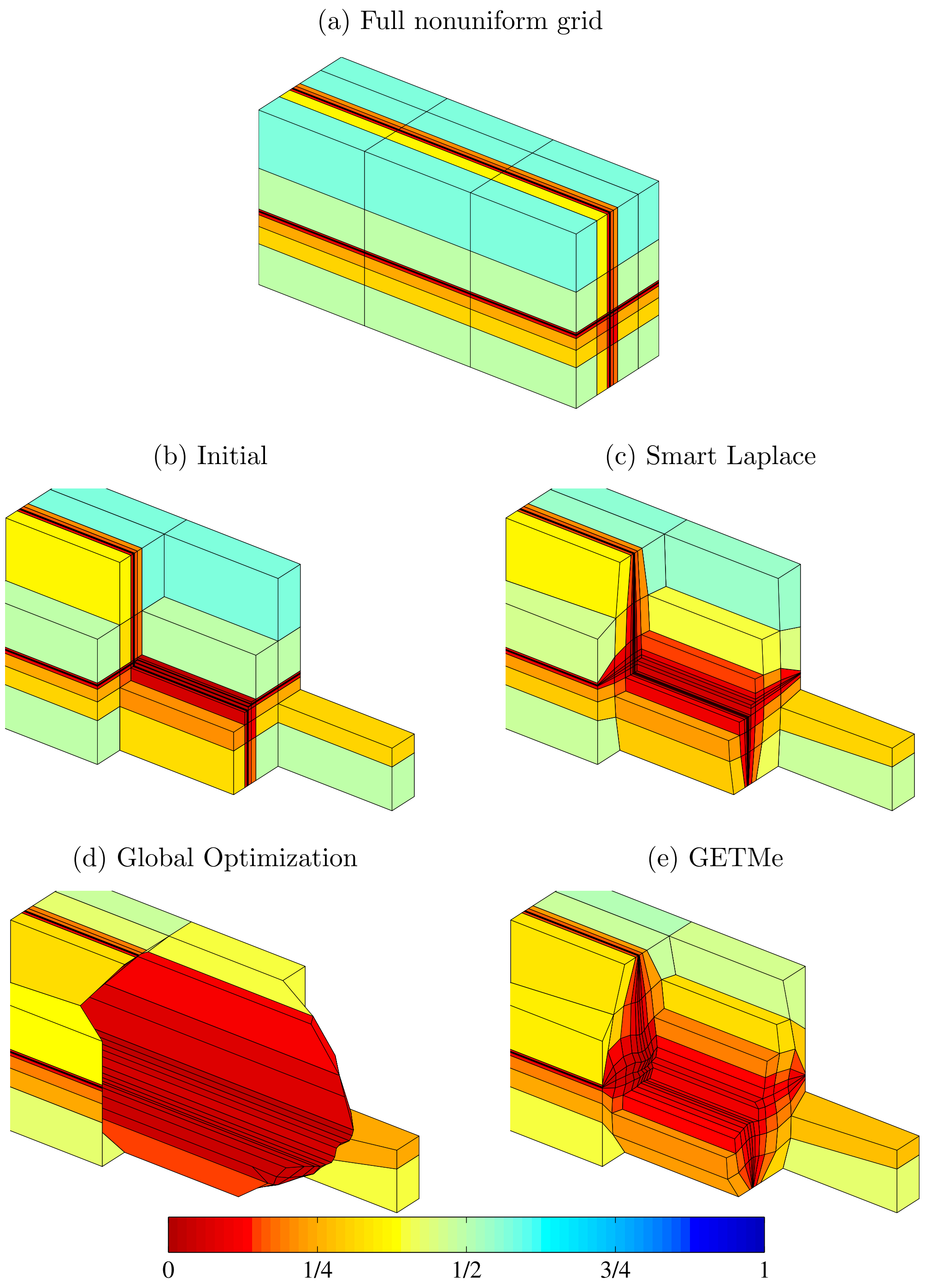


Figure 5.18: Full nonuniform grid and cross sections of smoothed meshes with elements colored according to their mean ratio quality numbers

minimal element quality can be significantly improved to 0.0634 with a moderate decrease of the mean mesh quality to 0.1165.

GETMe smoothing not only leads to convincing results with respect to quality numbers, but also with respect to smoothing runtime as is depicted in Fig. 5.19. In this mean mesh quality numbers are shown with respect to smoothing runtime whereby each marker indicates the results after one smoothing step. Hence, the horizontal distance between to markers denotes the runtime of one smoothing step. In the case of sequential GETMe smoothing markers indicate the results obtained after 100 sequential element transformations. Results of smart Laplacian smoothing are marked blue, those of global optimization green, and the two phases of GETMe smoothing are colored red and magenta respectively.

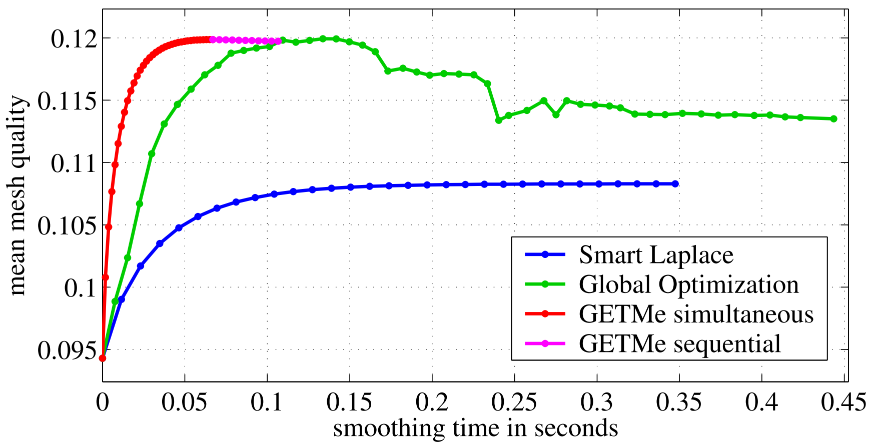


Figure 5.19: Nonuniform grid mean quality with respect to smoothing runtime

A straightforward C++ implementation of GETMe smoothing and smart Laplacian smoothing has been compared to the results of the feasible Newton-based global optimization approach of Mesquite version 1.1.7 [137], which is also implemented in C++. Runtimes have been measured on a notebook with an Intel® Core$^{\text{TM}}$2 Duo CPU T7250 (2MB cache, 2.00GHz, 800MHz FSB), 2GB RAM, 32bit Linux operating system with kernel 2.6.31.12, and the GNU C++ compiler version 4.4.1. Only the pure smoothing time was measured in order to exclude runtime distorting effects caused by different data formats, initialization procedures, and so on.

On average, one iteration of smart Laplacian smoothing, global optimization and GETMe simultaneous smoothing took 0.0116s, 0.0087s, and 0.0019s respectively. Hence, in the given example simultaneous GETMe smoothing is considerably faster than the other two methods. In the case of smart Laplacian smoothing this comes due to the fact that the number of element quality evaluations is significantly larger than in the case of GETMe simultaneous smoothing, since smart

Laplacian smoothing has to assess the quality of all elements adjacent to the node, which has to be updated. Hence the quality number of each inner element has therefore to be evaluated eight times due to its number of changed element nodes, whereas in the case of GETMe simultaneous smoothing the mean ratio number has to be evaluated only once per element. According to its definition the evaluation of the element mean ratio number is computationally expensive resulting in a significant drawback of smart Laplacian smoothing of hexahedral meshes. Using standard Laplacian smoothing without quality evaluation would lead to drastic shorter runtimes but usually also to the generation of inverted elements or quality deterioration.

5.3.2 Subdivision mesh

The mesh considered in the second example has been generated by repeatedly subdividing the elements of an initial cube using some of the refinement templates described in [87]. To be precise, the templates T_2, T_4, and T_8 depicted in Fig. 5.20 in their exploded view have been used. They subdivide a cube into 5, 13, and 27 hexahedral elements respectively. Here, the different mean ratio numbers of the hexahedra of T_2 are given by 0.6357, 0.7048, and 0.8399 respectively. Those of T_4 range from 0.5400 to 1.0000.

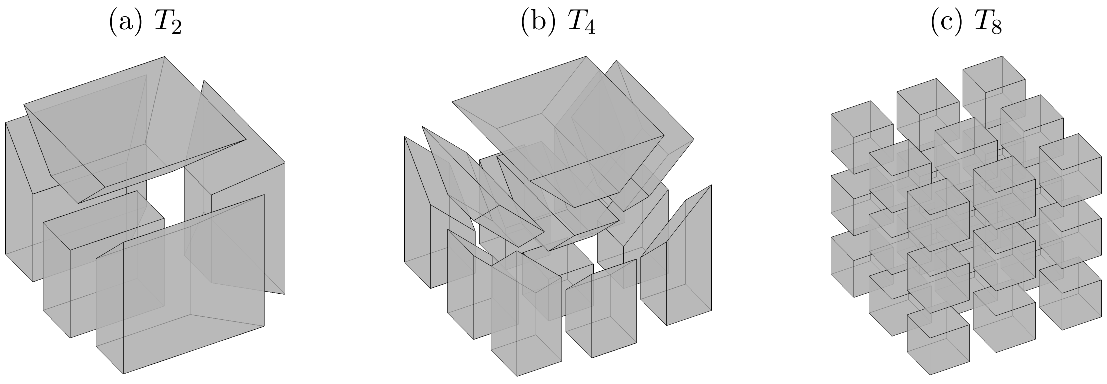


Figure 5.20: Hexahedral refinement templates

In order to generate the initial mesh, a cube of edge length 1000 has been cut in half for each coordinate direction resulting in eight equally sized cubes. Subsequently, these cubes have been refined by applying the refinement templates according to the following scheme:

$$\text{upper layer:} \quad \begin{array}{c|c} T_2 & \text{-} \\ \hline T_4 & T_2 \end{array} \qquad \text{lower layer:} \quad \begin{array}{c|c} T_4 & T_2 \\ \hline T_8 & T_4 \end{array}$$

After that, the eight elements forming the lower outer cube of the template T_8 have been recursively subdivided following the same subdivision scheme. Applying this construction scheme ten times results in a cube with a mesh successively refined towards one of its nodes. By using this mesh as octants of a cube with edge length 2000, a hexahedral mesh refined towards the center of the cube has been generated. Since the subdivision templates comprise elements of already good quality, inner nodes have been additionally distorted by element validity preserving random node movements in order to decrease the minimal and average element quality numbers of the subdivided mesh. The resulting initial mesh is depicted in Fig. 5.21.

Because of the templates T_2 and T_4 the mesh is unstructured and consists of 6165 nodes and 5984 hexahedral elements. Here, the number of adjacent elements per node range from 1 in the case of outer corner nodes to 16 in the case of specific inner nodes and average to 7.8 elements per node. More than 48% of the inner nodes are irregular, that is they do not have exactly eight adjacent hexahedral elements. The minimal and average element quality numbers of the initial mesh are given by 0.1370 and 0.4979 respectively. Due to the iterated subdivision scheme, the mesh is strongly graded towards the center of the model.

Table 5.9: Subdivision mesh smoothing results

Method / Criterion	$q_{\min}$	q_{mean}
Initial	0.1370	0.4979
Smart Laplace	0.1721	0.7906
Global Optimization	0.5662	0.8395
GETMe	0.6619	0.8254

Cross sections of the initial and smoothed meshes are depicted in the lower two rows of Fig. 5.21 and the according mesh quality numbers are given in Table 5.9. As can be seen in the case of smart Laplacian smoothing, although the mean mesh quality has been considerably improved, low quality elements remain in the mesh. Furthermore, the minimal element quality number $q_{\min} = 0.1721$ is well below the undistorted minimal element quality number given by 0.5400. In contrast, global optimization-based and GETMe smoothing, not only lead to better results with respect to q_{mean}, but also with respect to the minimal element quality $q_{\min}$.

This is also reflected by the element quality histogram depicted in Fig. 5.22 using 30 equidistant quality bins. Whereas smart Laplacian smoothing reduced the number of elements with $q(H_j) < 0.6619$ from 5476 to 1427, global optimization reduced this number to 296. In the case of the simultaneous GETMe substep resulting in a mesh with $q_{\min} = 0.6357$ and $q_{\text{mean}} = 0.8256$, 95 hexahedra with $q(H_j) < 0.6619$ remain. Finally, the sequential GETMe substep was able

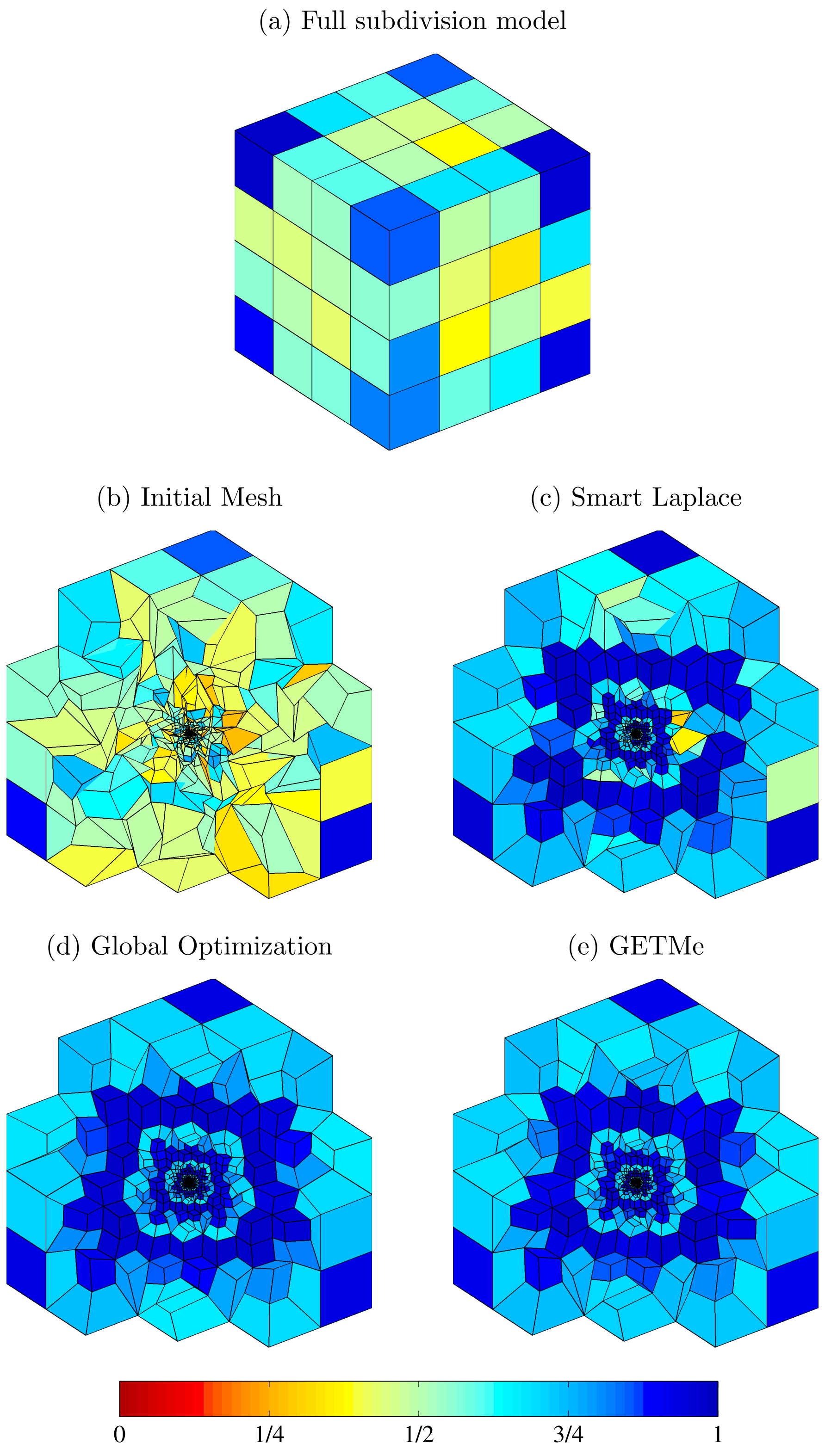


Figure 5.21: Full subdivision model (a) and cross sections of smoothed meshes with elements colored according to their mean ratio quality number (b)–(e)

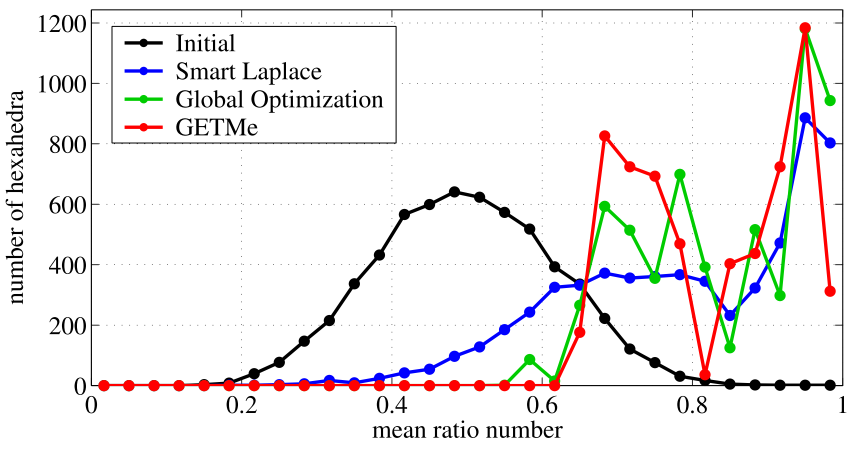


Figure 5.22: Subdivision mesh element quality histogram

to improve all these elements, resulting in no such elements remaining. In practice, this is particularly important, since finite element solution accuracy can be significantly affected by low quality elements. Here, GETMe smoothing not only leads to good q_{min} values but also reduces the number of low quality elements on a broad range of low quality numbers more effectively and efficiently.

Smart Laplacian smoothing and GETMe simultaneous smoothing led to best results in the last of a total of 27 iteration steps in each case. In contrast, the average number of 1.5 inverted elements, which had to be handled per GETMe simultaneous iteration step, is substantially lower than the 65 inverted elements per iteration step in the case of smart Laplacian smoothing. The subsequently applied 1400 sequential GETMe smoothing steps generated no inverted elements at all. Mesquite performed a total of 77 iterations within four shape improvement wrapper calls of which the 71st iteration led to the mesh with the best mean quality. Here, the fourth call was terminated prematurely after 15 iterations, since otherwise it would have accomplished more than 1000 iterations without further improvements.

Mesh quality improvement with respect to smoothing runtime is depicted in Fig. 5.23. Again, GETMe smoothing leads to a fast and substantial improvement within the first steps with respect to q_{min} as well as q_{mean}. The total smoothing runtimes amount to 5.3s, 16.6s, and 1.1s in the case of smart Laplacian, global optimization-based, and GETMe smoothing. In particular, the simultaneous GETMe smoothing substep obtained the mean mesh quality number 0.8 after 4 iterations taking in total 0.16s, which is about 20 times faster than the 23 iterations needed in the case of global optimization-based smoothing taking 3.19s in total. Smart Laplacian smoothing was not able to achieve this mean mesh quality level at all.

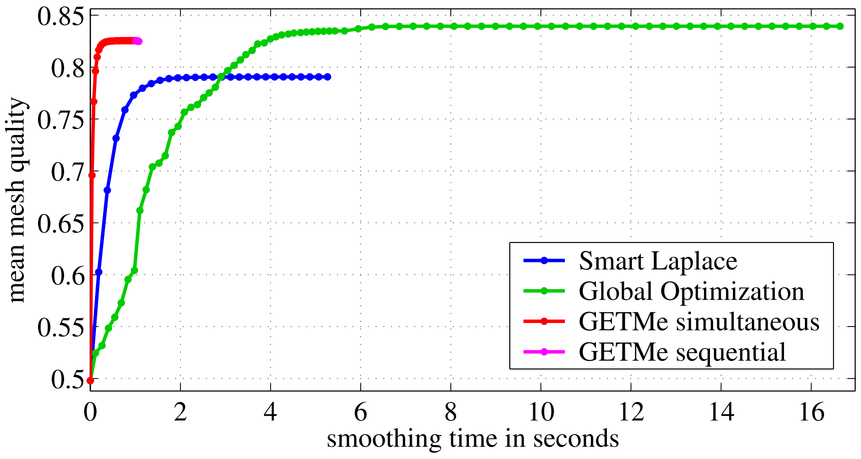


Figure 5.23: Subdivision mesh mean quality with respect to smoothing runtime

5.3.3 Sweep mesh

Sweeping techniques, where meshes are generated by projecting a 2D mesh along a path, are widely used in the generation of all-hexahedral meshes. In this subsection, smoothing results for a basic sample mesh created by this approach are presented. The mesh has been generated by rotating the unstructured mesh of quadrilateral elements depicted on the left of Fig. 5.24. The resulting full model is depicted on the right. Here, the overall mesh quality has been additionally deteriorated by element validity preserving random inner node movements.

The initial 2D sweeping mesh consists of 817 quadrilateral elements, which have been rotated along a coordinate axis generating 15 layers of hexahedral elements. The resulting unstructured hexahedral mesh consists of 14,352 nodes and 12,255 hexahedral elements. On average 6.8 elements are adjacent to a mesh node, where individual adjacency numbers range from 1 to 12. Due to the additional distortion of the initial mesh, the minimal and mean mesh quality numbers given by $q_{\min} = 0.0788$ and $q_{\text{mean}} = 0.4452$ are comparatively low as can also be seen by the cross section of the initial mesh depicted on the upper left of Fig. 5.25.

Results obtained by all smoothing methods are given in Table 5.10. As can be seen, Laplacian smoothing was not able to improve the worst element to the same extend as the other two methods. In particular, some clusters of low quality elements remain as can be seen on the upper right of Fig. 5.25. In contrast, the results of global optimization and GETMe based smoothing are balanced with respect to both quality numbers.

In the case of GETMe smoothing using the unified parameter set resulted in 52 simultaneous iteration steps generating a total of 27 invalid elements, which have

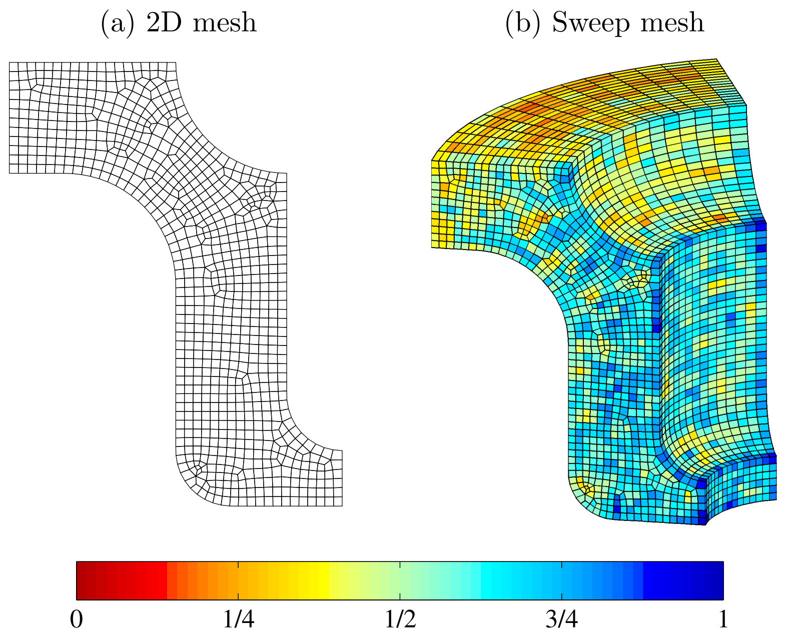


Figure 5.24: Unstructured quadrilateral 2D mesh and resulting full 3D model after sweeping and inner node distortion

Table 5.10: Sweep mesh smoothing results

Method / Criterion	q_{min}	q_{mean}
Initial	0.0788	0.4452
Smart Laplace	0.1323	0.7036
Global Optimization	0.4029	0.7227
GETMe	0.4125	0.7235

been handled by the successive note resetting approach. The subsequently applied sequential approach performed 5600 iteration steps. Smart Laplacian smoothing performed 55 iterations leading in total to 1828 invalid elements, which is about 68 times more as in the GETMe case. Global optimization based smoothing performed in total 62 feasible Newton iterations within four optimization cycles. In doing so, the best mean mesh quality number was achieved after 49 iterations. Subsequent iterations further improved q_{min}, but deteriorated q_{mean}. The best results of GETMe simultaneous smoothing and smart Laplace have been achieved by the last iteration step in each case.

Again, GETMe smoothing can even lead to better results if individual parameters are used. For example, the mean mesh quality can be further improved to 0.7266 by the simultaneous substep and the minimal element quality to 0.5038 by the sequential smoothing substep.

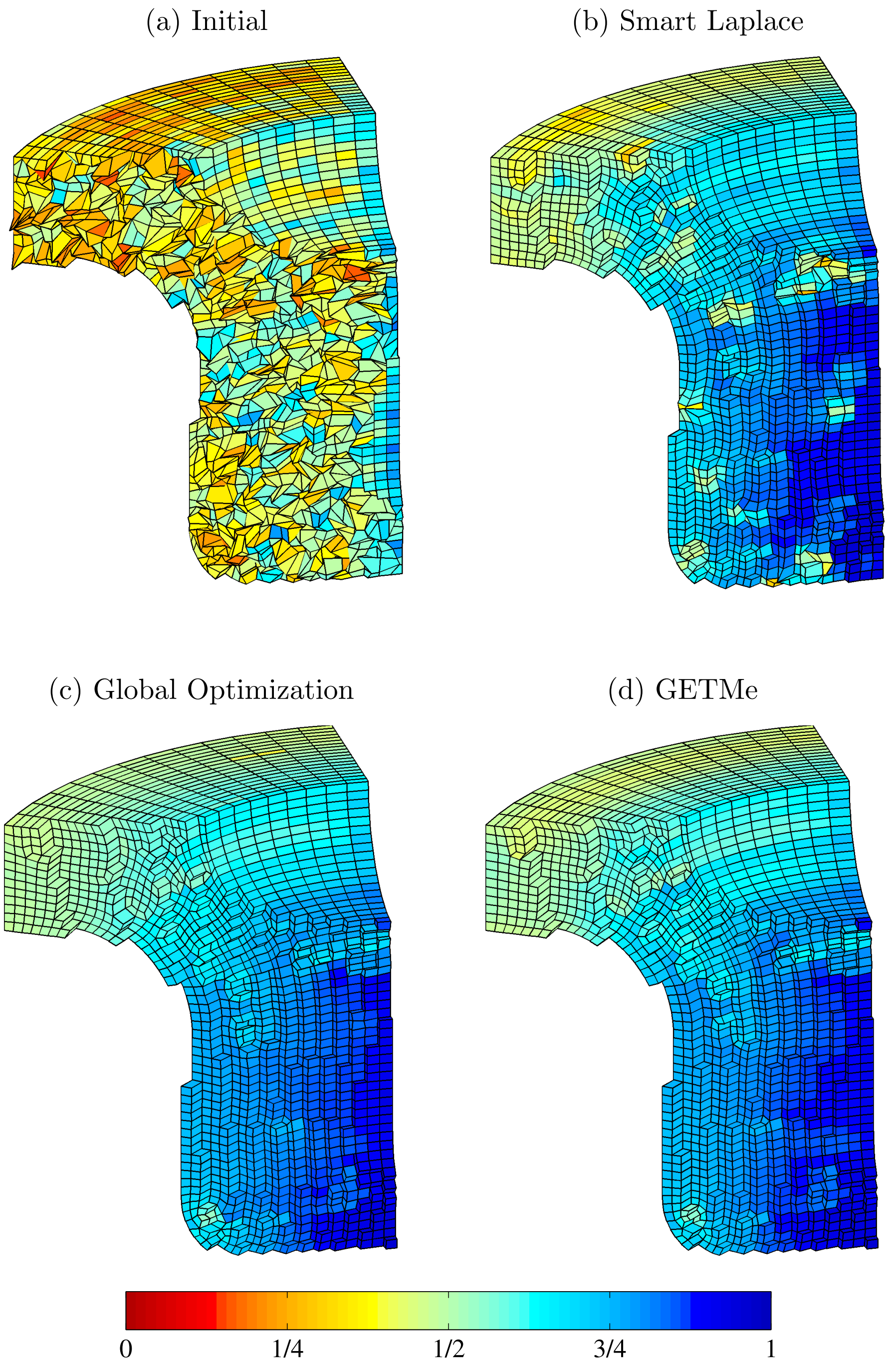


Figure 5.25: Cross sections of initial and smoothed sweep meshes

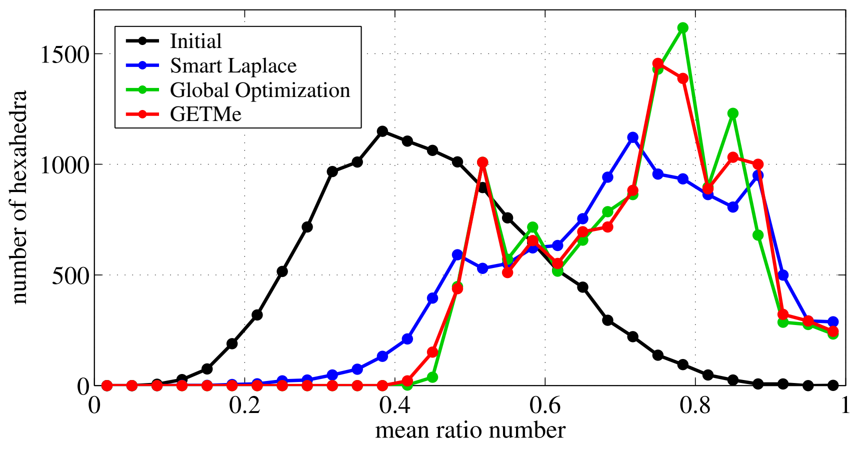


Figure 5.26: Sweep mesh element quality histogram

Due to the comparatively low number of sequential GETMe steps, there is no characteristic peak of elements with quality numbers near $q_{\min}$ in the element quality histogram depicted in Fig. 5.26. As can also be seen, the quality graph of GETMe smoothing visually resembles that of the global optimization approach, whereas smart Laplacian smoothing differs significantly. In particular, the number of elements with $q(H_j) < 0.4$ is larger.

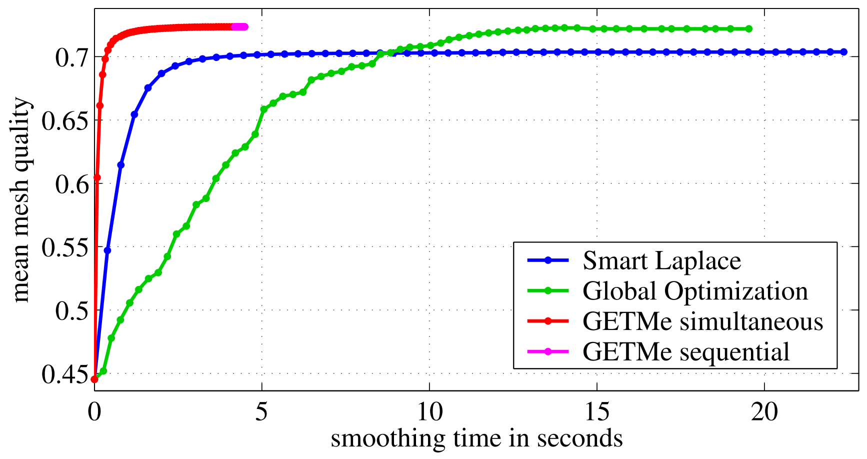


Figure 5.27: Sweep mesh mean quality with respect to smoothing runtime

Fig. 5.27 depicts the mean element quality number with respect to smoothing runtime. Again, GETMe smoothing is significantly faster. To be precise, smoothing took in total 22.4s, 19.5s, and 4.5s in the case of smart Laplacian, global optimization-based, and GETMe smoothing, where the simultaneous substep took 4.2s and the sequential 0.3s. All runtimes refer to non-parallelized implementations. Compared to this, a first simple parallelized version of GETMe simultaneous based on the OpenMP [160] directive "`#pragma omp parallel for`" applied to major element loops, like that of element transformation and node averaging, resulted in a runtime of 2.4s. This already yields an excellent speedup factor of about 1.8 on the test system named before with a theoretical speedup limit of 2.0. Hence, GETMe smoothing is well suited for parallelization techniques.

5.3.4 Aletis CFD-mesh

The last example considers the inner core of a real world computational fluid dynamics mesh of the car Aletis developed by TWT GmbH Science & Innovation, commissioned by the Hellenic Vehicle Industry EΛBO S.A. Thessaloniki [169]. A picture of the fully functional Aletis prototype is shown in the upper row of Fig. 5.28 and the feature reduced surface mesh of the model used in the CFD simulation representing one half of the car is depicted in the middle row.

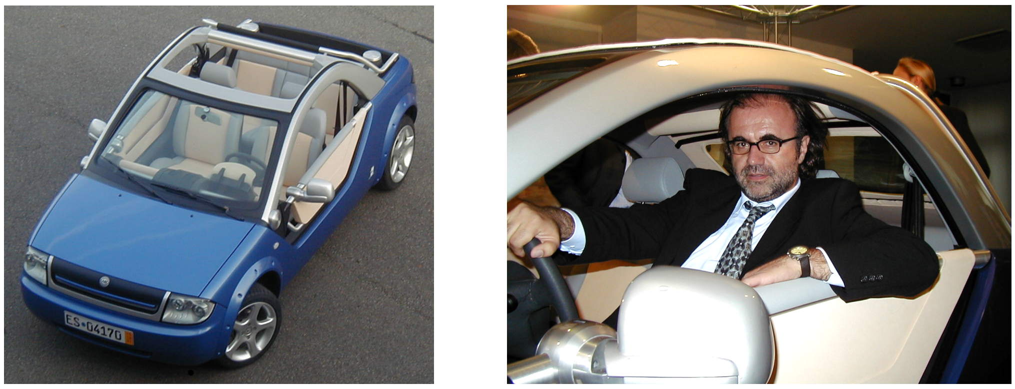
The initial volume mesh generated in 2001 by TWT GmbH Science & Innovation using ICEM-Hexa version 4.0 by ANSYS, Inc. covering the external of the car model is depicted in the lower row of Fig. 5.28. It comprises of 1,710,978 nodes and 1,610,234 hexahedral elements with quality numbers ranging from $q_{\text{min}} = 0.0043$ to 0.9998 resulting in a mean mesh quality of $q_{\text{mean}} = 0.6195$. On average 7.5 elements are adjacent to each node, where individual numbers range from 1 to 10. Each element has on average 24.9 adjacent elements, that is elements, which share at least one node, where numbers range from 7 to 30.

Since the CFD simulation approach uses a specific turbulence model, the initial mesh contains several layers of thin interface elements near the surface of the car body. In practice these interface mesh layers would have been kept fix. However, in order to judge smoothing results under unfavorable circumstances, all mesh elements including those of the interface layers have been smoothed imposing additional difficulties due to the rapid change of element size.

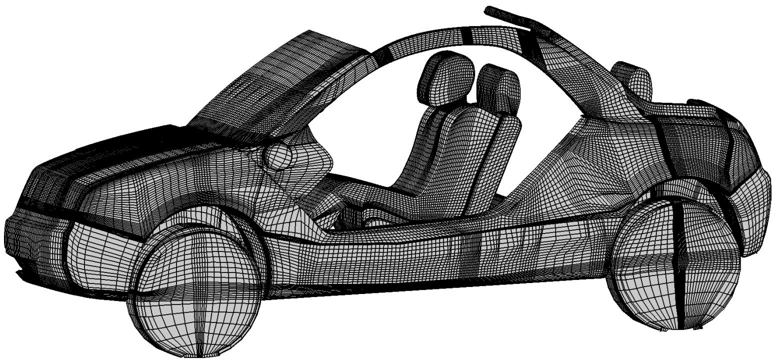
Although the initial mesh consists entirely of valid hexahedral elements the shape improvement wrapper of Mesquite generated eight inverted elements during optimization and could not proceed due to invalid gradient computations. Hence, only results of smart Laplacian smoothing can be given for comparison with the results of GETMe smoothing using the unified standard parameters as stated in the beginning of Section 5.3. The quality numbers achieved are given in Table 5.11.

As can bee seen, both smoothing approaches only led to a slight improvement of the worst element quality due to disadvantageous geometry and fixed boundary configurations as depicted in Fig. 5.29. In this, a small bundle of long elements

(a) Fully functional Aletis prototype



(b) Reduced complexity surface mesh



(c) CFD volume mesh

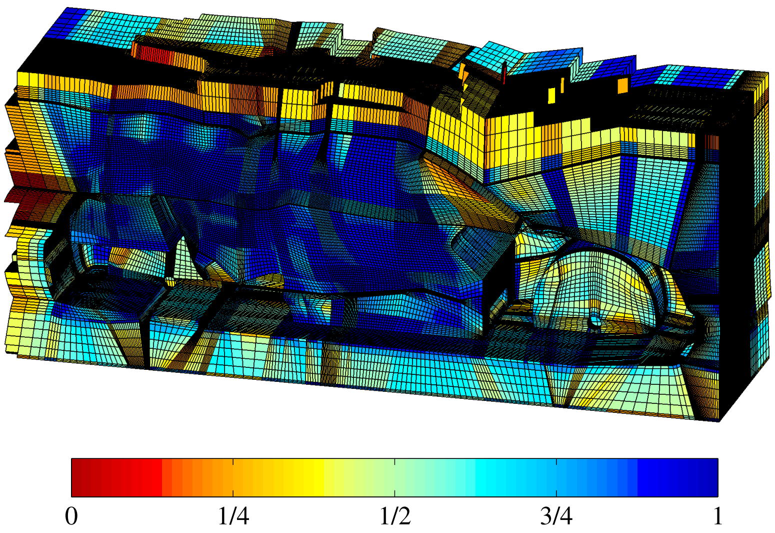


Figure 5.28: Aletis prototype and meshes

Table 5.11: Aletis mesh smoothing results

Method / Criterion	$q_{\min}$	q_{mean}
Initial	0.0043	0.6195
Smart Laplace	0.0046	0.7546
Global Optimization	aborted	aborted
GETMe	0.0052	0.7507

including the overall worst element fixed on one side has not the necessary degrees of freedom to be improved appropriately. In practice, topology modifying approaches, like subdivision techniques, have to be involved in order to resolve such problems.

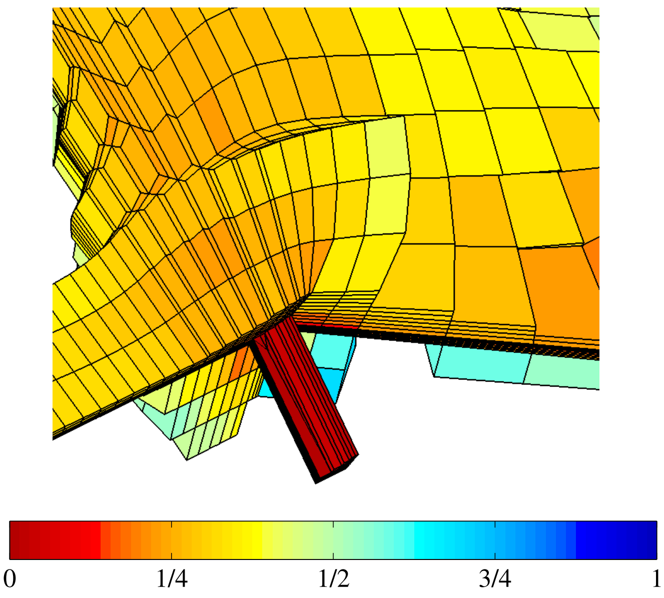


Figure 5.29: Worst elements of mesh

Although the worst element quality is only slightly improved, GETMe reduces the number of low quality elements considerably better than smart Laplacian smoothing as can be seen in the element quality histogram depicted in Fig. 5.30. For example, whereas the initial mesh contains 33,630 and 125,137 elements with quality numbers below 0.1 and 0.2 respectively, smart Laplacian smoothing reduces these numbers to 17,255 and 69,525, and GETMe smoothing to 1898 and 18,154 respectively. Hence, the number of problematic elements with quality numbers below 0.1 is therefore about 9 and 18 times smaller as in the case of smart Laplacian smoothing or the initial mesh respectively.

As can be seen by the quality histogram, smart Laplacian smoothing results in

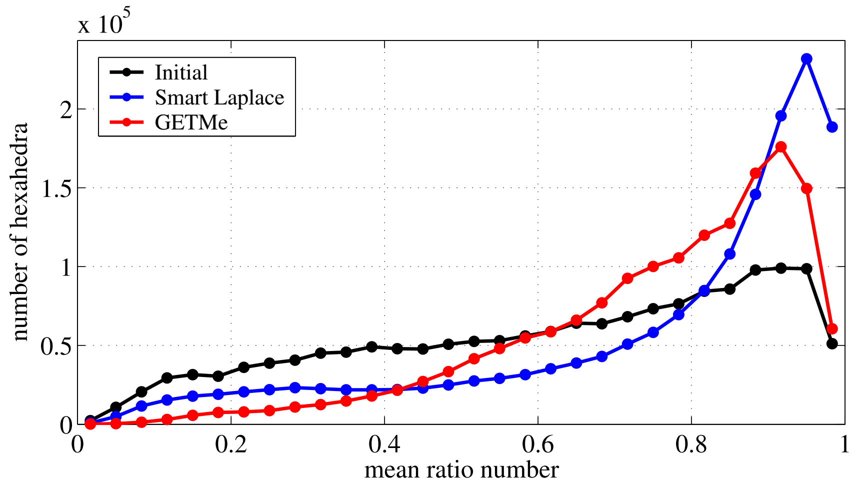


Figure 5.30: Aletis mesh element quality histogram

a larger number of elements with high quality numbers and thus yields a slightly larger mean mesh quality value. However, as is shown by the cross and longitudinal sections of the initial as well as the resulting smoothed meshes depicted in Fig. 5.31, results of smart Laplacian smoothing are not satisfying from an application point of view. Although there are regions of high quality, clusters of low quality elements and complete layers of non smoothed elements remain with a rapid change of element size reducing numerical stability and solution accuracy of the CFD computation. In contrast, GETMe smoothes the entire mesh and results in an gradual change of element size. As in the case of the other examples, further improvements can be achieved by using individual parameters for GETMe smoothing resulting e.g. in a mesh with $q_{\text{min}} = 0.0067$ and $q_{\text{mean}} = 0.7580$.

To study the influence of the initial node positions on the results of the smoothing process, hence the robustness of the smoothing approach, the initial mesh has been additionally distorted by validity preserving random inner node movements. That is, the undistorted and the distorted initial mesh differ only in inner node positions but not in boundary node positions, validity and mesh topology. Here, the minimal and maximal element quality deteriorated only slightly to $q_{\text{min}} = 0.0042$ and 0.9750 respectively. In contrast, the mean mesh quality deteriorated by nearly 33% to $q_{\text{mean}} = 0.4165$.

Ideally, smoothing results are mainly affected by mesh topology and fixed boundary node positions but not by initial inner node positions since the resulting configuration should maximize (at least locally) the overall mesh quality. This does not hold for smart Laplacian smoothing as can be seen by the results given in Table 5.12.

As in the case of the undistorted mesh, the shape improvement wrapper of

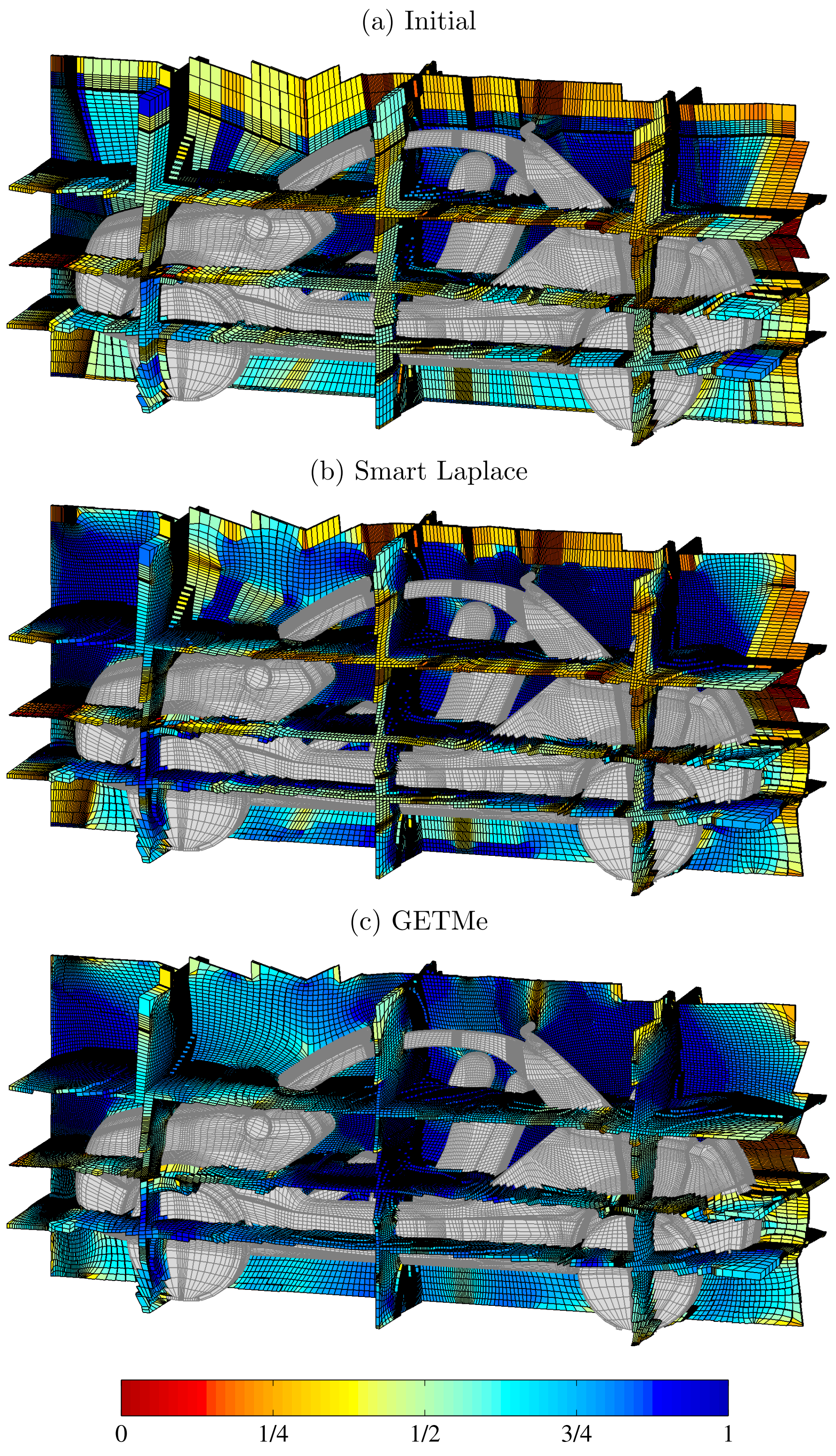


Figure 5.31: Cross and longitudinal sections of not randomly distorted initial and smoothed Aletis meshes

Table 5.12: Distorted Aletis mesh smoothing results

Method / Criterion	q_{min}	q_{mean}
Initial	0.0042	0.4165
Smart Laplace	0.0045	0.7059
Global Optimization	aborted	aborted
GETMe	0.0062	0.7492

mesquite aborted preliminary due to the generation of inverted elements and resulting failed gradient computations. Compared to the quality numbers of the undistorted initial mesh, the mean mesh quality achieved by smart Laplacian smoothing deteriorates by more than 6.5%, while in the case of GETMe smoothing q_{mean} deteriorates only by 0.2%. This can also be seen by comparing the smoothed meshes of the distorted Aletis model depicted in Fig. 5.32 with those of the undistorted initial model depicted in Fig. 5.31. While the cross sections of GETMe are graphically nearly the same, quality deterioration is clearly visible in the results of smart Laplacian smoothing.

This is also substantiated by the histogram difference graphs depicted in Fig. 5.33. Here, the ordinate of each marker indicates the difference of the number of elements within a given quality bin for the undistorted and the distorted Aletis mesh. For example, the positive values for quality numbers $q \in [0.0, 0.62)$ in the initial mesh marked black indicate that the number of low and medium quality elements in the distorted mesh is significantly larger than in the undistorted mesh. In turn, the number of good to high quality elements, i.e. elements of quality $q \in [0.62, 1.0]$, is decreased. The histogram difference graph of smart Laplacian smoothing marked blue shows a considerably large reduction of high quality elements with $q > 0.85$ and an increase of the number of elements with $q < 0.85$. In contrast, GETMe smoothing results are affected only little, since element numbers in the same bin of the histogram for the distorted and the undistorted mesh differ only up to 1493 for $q \approx 0.72$. In the case of smart Laplacian smoothing element numbers differ up to 72,542 for $q \approx 0.98$ and in the case of the initial mesh up to 98,963 for $q \approx 0.92$.

Again, GETMe smoothing is also more advantageous with respect to its quality and runtime behavior if compared to smart Laplacian smoothing as is depicted in Fig. 5.34. Due to the tight termination tolerance used in both cases iteration numbers and runtimes are comparably large. In the case of the undistorted initial mesh smart Laplacian smoothing performed 1460 iterations taking 21 hours, 33 minutes, and 12 seconds in total with an average iteration runtime of about 53 seconds. The simultaneous GETMe substep performed 1257 iterations within three hours, 53 minutes, and 49 seconds with an average iteration runtime of

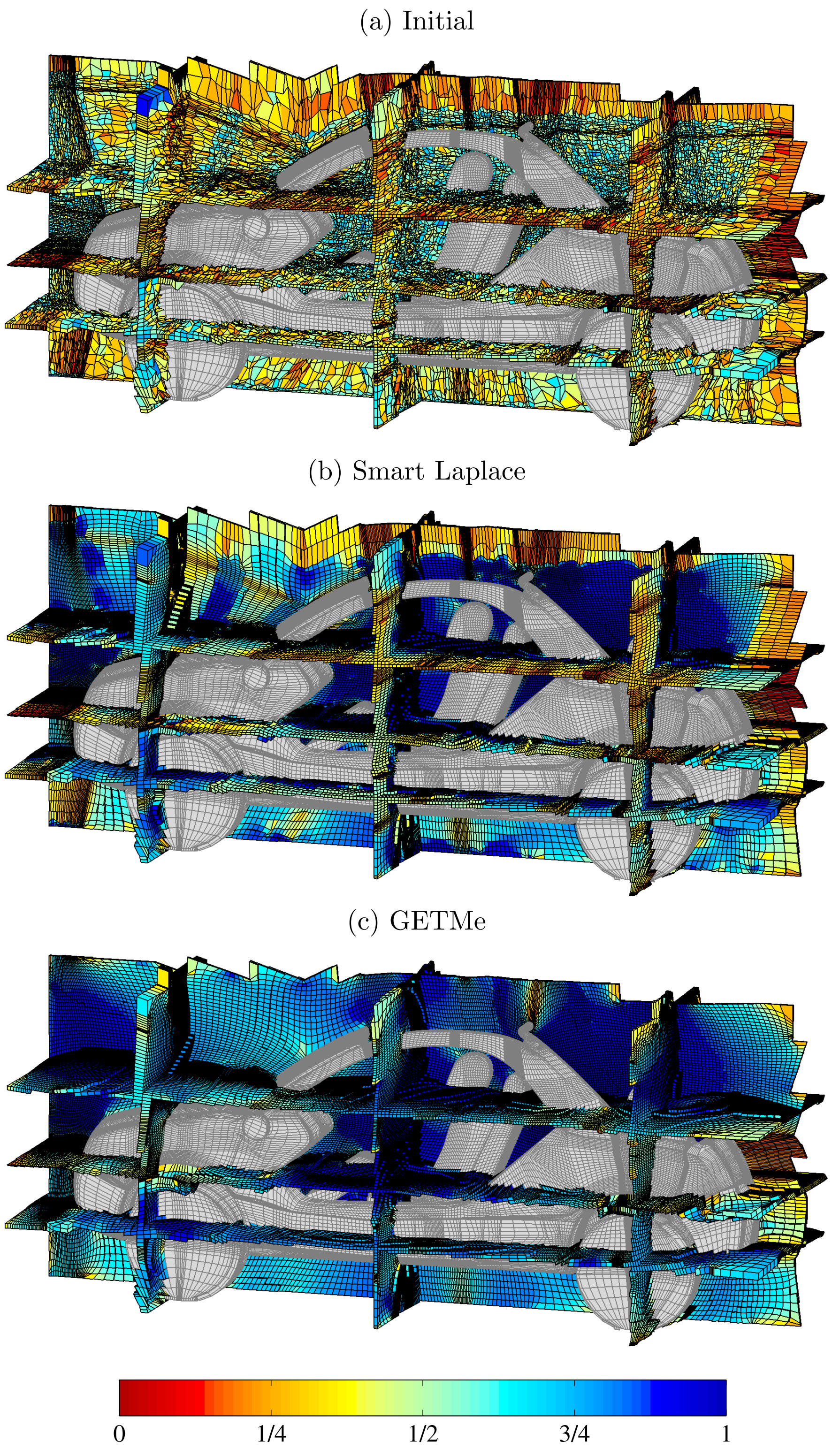


Figure 5.32: Cross and longitudinal sections of randomly distorted initial and smoothed Aletis meshes

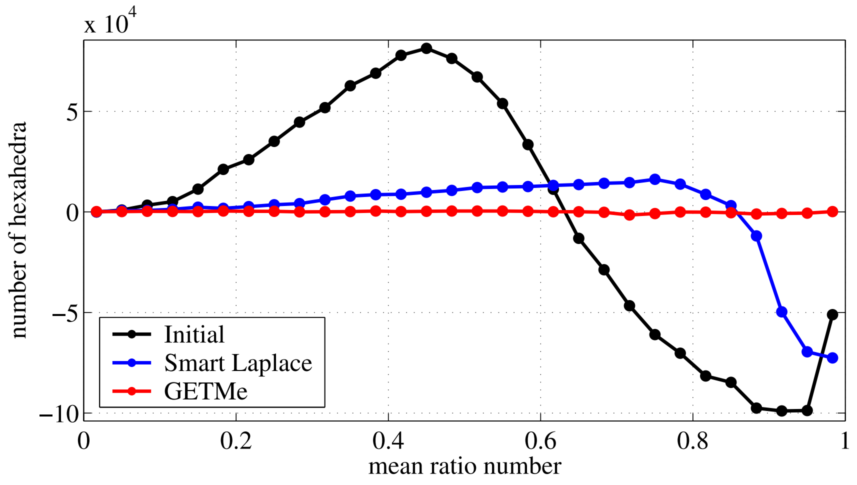


Figure 5.33: Distorted Aletis histogram difference graphs

about 11 seconds, hence being about 4.8 times faster than one iteration of smart Laplacian smoothing. The parallelized version of simultaneous GETMe smoothing achieved the same result in two hours, 11 minutes and 35 seconds resulting in a speedup factor of about 1.8. The subsequently applied sequential GETMe substep performed 16,400 iteration steps within 21 seconds.

Due to the comparatively large number of iterations, in Fig. 5.34 iteration markers would not be distinguishable and hence are omitted. Furthermore, since the sequential GETMe smoothing time is very short compared to the simultaneous GETMe smoothing time the GETMe quality and runtime graphs comprise both smoothing substeps.

Fig. 5.34 also depicts results for the distorted initial Aletis mesh. As can be seen, distorting the mesh not only has a significant impact on the solution accuracy but also on the runtime behavior of smart Laplacian smoothing. In contrast to the case of the undistorted mesh smoothing terminated already after 700 iterations taking 10 hours, 20 minutes and 12 seconds in total, whereas GETMe smoothing took three hours, 55 minutes and 19 seconds performing 1240 simultaneous and 2800 sequential smoothing steps. That is, GETMe runtimes and quality differ only slightly, which can also be seen by the overlapping quality and runtime graphs in Fig. 5.34. Hence GETMe smoothing is not only stable with respect to smoothing results, but also with respect to its runtime behavior. Furthermore, in the case of the distorted initial mesh, GETMe smoothing reaches a mean mesh quality number of 0.7 after 37 steps taking six minutes and 58 seconds, being about 16 times faster than smart Laplacian smoothing requiring 124 iterations taking one hour and 49 minutes and 54 seconds.

In practice, results can be obtained much faster. For example, using a work-

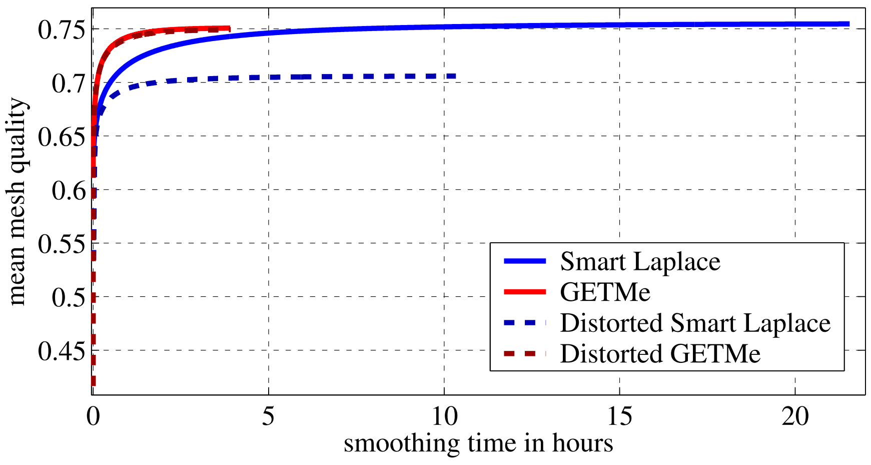


Figure 5.34: Undistorted and distorted Aletis mesh mean quality with respect to smoothing runtime (GETMe results are nearly congruent)

station with two Intel® Xeon™ X5550 processors, being more representative for a computational engineering environment, one iteration of the parallelized simultaneous GETMe smoothing of the undistorted Aletis model takes less than a second. Furthermore, it is reasonable to perform smoothing only as long as there is a steep ascent in mesh quality. That is, approximately one fourth of the iterations performed in the presented examples suffice to yield practically good meshes. On this basis, the total GETMe smoothing time of the undistorted Aletis model actually reduces to less than six minutes resulting in a mesh with $q_{\text{min}} = 0.0050$ and $q_{\text{mean}} = 0.7422$.

5.4 Mixed volume mesh smoothing examples

In this section, three meshes of different type will be considered to illustrate the effectiveness of GETMe smoothing of mixed volume meshes. Results will be compared to those obtained by the geometry-based smart Laplacian smoothing approach, which is terminated, if the q_{mean} values of two consecutive meshes deviate less than 10^{-6}. GETMe results are also compared to those obtained by the shape improvement wrapper of the mesh quality improvement toolkit Mesquite version 2.1.2 [137]. This wrapper has been applied repeatedly using its default settings until the mean mesh quality could not be further improved. As in the case of smart Laplacian smoothing, the mesh with the best overall mean quality is used for comparison.

In the case of GETMe smoothing a common parameter set was used for all examples. It has been determined numerically by assessing the results of various meshes and parameter sets. For the simultaneous substep this resulted in $\sigma^{\text{tet}} \in [0.77, 0.84]$, $\sigma^{\text{hex}} \in [2.57, 3.45]$, $\sigma^{\text{pyr}} = 1.86$, $\sigma^{\text{pri}} = 1.59$, mean element edge length preserving scaling, $\varrho = 2/3$, and $\eta = 1/4$. The same termination criterion was applied as in the case of smart Laplacian smoothing. In the case of the sequential GETMe substep $\sigma^{\text{tet}} = 0.81$, $\sigma^{\text{hex}} = 2.74$, $\sigma^{\text{pyr}} = 1.82$, $\sigma^{\text{pri}} = 0.85$, average element edge length preserving scaling, and the more conservative relaxation parameter $\varrho = 0.01$ was used. Penalty values have been set to $\Delta\pi_{\text{i}} = 0.01$, $\Delta\pi_{\text{r}} = 0.0005$, and $\Delta\pi_{\text{s}} = 0.01$.

5.4.1 Hexahedral in tetrahedral mesh embedding

The first example considers a hexahedral mesh of a plate with two drill holes depicted in Fig. 5.35a embedded into a tetrahedral mesh of a cuboid. The plate mesh consists of 5748 hexahedral elements and is refined towards the drill hole with the smaller diameter. This mesh has been wrapped by one layer of 2220 pyramidal elements in order to embed it into the tetrahedral mesh consisting of 26,082 elements. A clipped version of the resulting hybrid mesh is depicted in Fig. 5.35b. Here, different element types are marked by different shades of gray and the cuboid is marked by thick black lines.

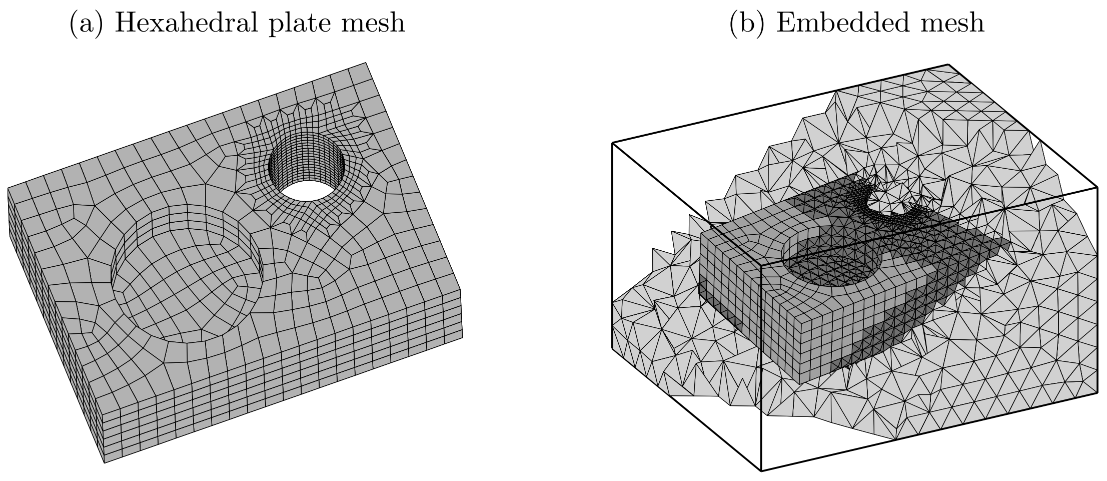


Figure 5.35: Hexahedral mesh of a plate and its embedding into a tetrahedral mesh using one layer of pyramidal elements. Different element types are marked by different shades of grey

In total, the mesh consists of 34,050 elements and 11,371 nodes of which 1022 are boundary nodes on the outer cuboid faces kept fixed during smoothing. The boundary nodes of the plate model surface have not been fixed and no constraints have been used in order to allow an unhampered smoothing of adjacent elements of different type. Hence, the shape of the embedded plate is not preserved. In practice, smoothing methods are usually combined with surface and feature line back-projection techniques or constrained node movement techniques in order to preserve external and internal boundaries.

Although the lowest element quality $q_{\mathrm{min}} = 0.1625$ is comparatively low, the mean mesh quality of $q_{\mathrm{mean}} = 0.7302$ indicates that the overall mesh quality is already reasonable. The initial mesh has been smoothed using the methods and parameters as described at the beginning of this section resulting in the runtime information and quality numbers given in Table 5.13. Cross sections of the initial, as well as the smoothed meshes, are depicted in Fig. 5.36.

Table 5.13: Embedded plate smoothing results

Method/Criterion	Iter	Time (s)	q_{min}	q_{mean}
Initial	–	–	0.1625	0.7302
Smart Laplace	21	5.17	0.0258	0.8195
Global Optimization	52	9.72	0.3400	0.8382
GETMe	26/11,500	2.70	0.4476	0.8300

Table 5.13 also provides the iteration numbers and smoothing time in seconds for all methods. In the case of GETMe smoothing the iteration numbers of both substeps are given. Smart Laplacian smoothing terminated after 21 iterations improving q_{mean} by 12.2% but also decreasing q_{min} significantly by 84.1%. This generation of low quality elements is also obvious from the cross section depicted in Fig. 5.36b if compared to the initial mesh depicted in Fig. 5.36a. Here, all elements are colored according to their mean ratio quality number, where reddish colors indicate low quality elements and bluish colors elements of high quality. Since meshes usually have to meet specific minimal requirements in subsequent applications, such a loss in minimal element quality is a drawback. However, no invalid elements have been generated due to the smart approach as they would have been generated by standard Laplacian smoothing.

The global optimization-based approach performed a total of 52 feasible Newton iterations within eight shape improvement wrapper cycles of Mesquite. Here, both quality numbers have been improved, namely q_{min} by 109.2% and q_{mean} by 14.8%, which can also be seen by the cross section depicted in Fig. 5.36c.

The GETMe simultaneous approach terminated after 26 iterations resulting in $q_{\mathrm{mean}} = 0.8351$, However, the mean mesh quality slightly decreased during the subsequently applied GETMe sequential approach, performing 11,500 iterations. This is caused by directly setting new node positions obtained by transforming the worst elements, which led to a slight decrease in neighboring elements quality.

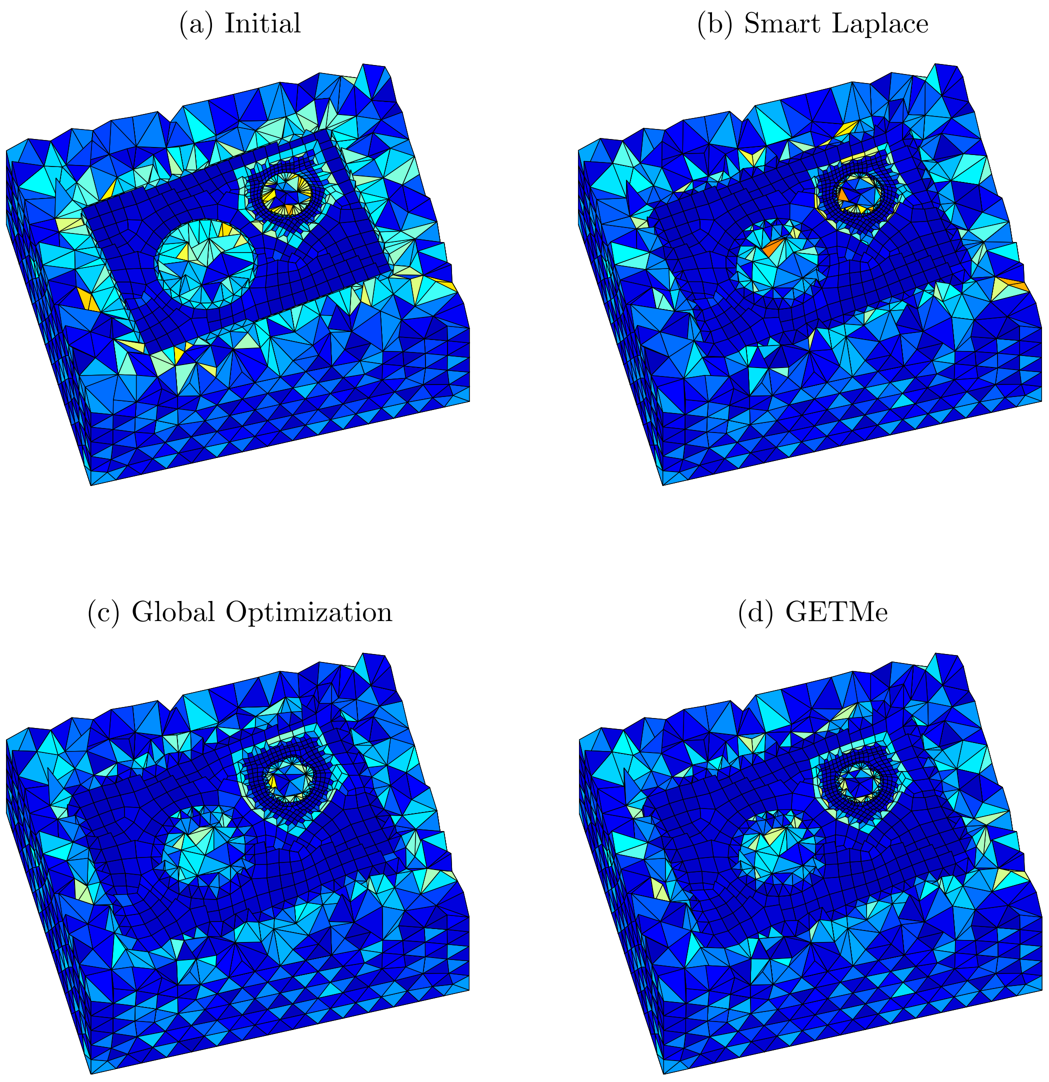


Figure 5.36: Initial and smoothed plate embedding meshes with elements colored according to their mean ratio quality number

Since only one element is transformed during one iteration step of GETMe sequential, the total number of element transformations corresponds to those performed in less than one GETMe simultaneous step, transforming all 34,050 mesh elements at once. Compared to the initial mesh the combined GETMe approach improved q_{min} by 175.4% and q_{mean} by 13.7%.

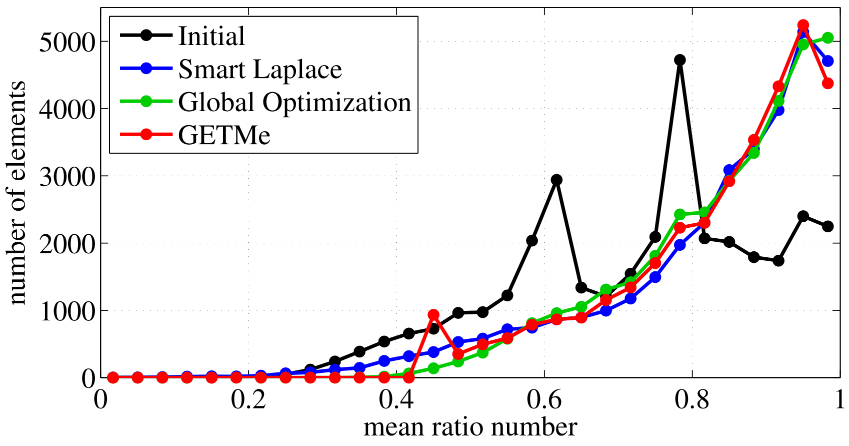


Figure 5.37: Embedded plate element quality histogram

As can be seen by the element quality histogram depicted in Fig. 5.37 and the minimal element quality results given in Table 5.13, GETMe smoothing significantly reduces the number of low quality elements. To be precise, the number of elements with a mean ratio value below 0.44 amounts to 2150, 1138, 106, and 0 in the case of the initial mesh and those obtained by smart Laplacian smoothing, global optimization-based smoothing, and GETMe smoothing, respectively. The peak in the histogram of GETMe smoothing near q_{min} is caused by the sequential substep, which successively improves low quality elements, leading to an accumulation of elements near this threshold.

GETMe smoothing also has a favorable runtime behavior, as can be seen in Fig. 5.38 depicting the mean mesh quality q_{mean} with respect to smoothing runtime in seconds. Here, each circular marker indicates the results of one iteration. Results have been obtained by a straightforward C++ implementation of GETMe smoothing and smart Laplacian smoothing. They are compared to the results of the feasible Newton-based global optimization approach of Mesquite version 2.1.2 [137], which is also implemented in C++. Runtimes have been measured on a notebook with an Intel® Core$^{\text{TM}}$ i5-540M CPU (dual core, 3MB cache, 2.53GHz), 4GB RAM, 64bit Linux operating system with kernel 2.6.34.4, and the GNU C++ compiler version 4.5.1.

The complete smoothing time amounts to 5.17s, 9.72s, and 2.70s in the case

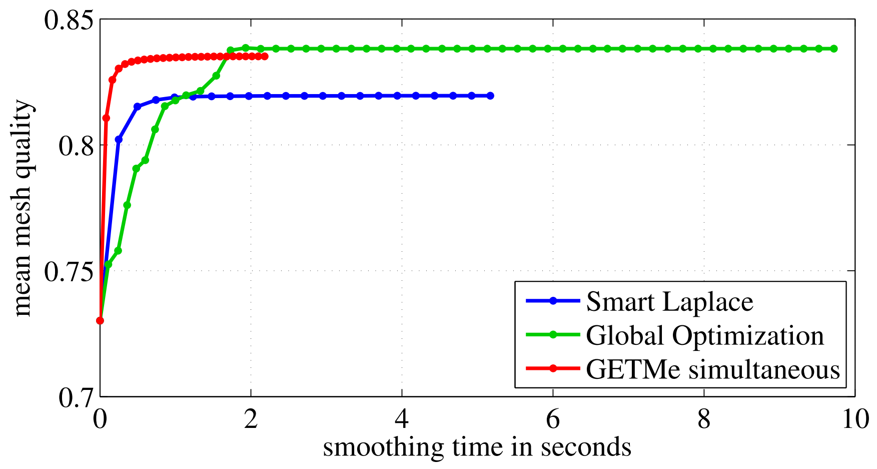


Figure 5.38: Embedded plate mean mesh quality with respect to smoothing runtime

of smart Laplace, global optimization and GETMe smoothing respectively. In the case of GETMe smoothing, the simultaneous substep took 2.18s and the sequential 0.52s. In all cases, due to the comparatively tight termination criterion, the majority of iterations only led to little improvements. In practice, smoothing would have been stopped at an earlier stage. For example, the mean mesh quality of 0.83 was reached by global optimization after 12 iterations taking 1.7s and by GETMe simultaneous after three iterations taking 0.2s. That is, even a not specifically optimized version of GETMe smoothing can achieve quality meshes comparable to those obtained by the global optimization approach within significantly shorter runtimes. In contrast, smart Laplacian smoothing was not able to reach this level of mesh quality.

5.4.2 Tetrahedral in hexahedral mesh embedding

The second example considers the embedding of a ring into a hexahedral mesh. The ring surface mesh depicted in Fig. 5.39a is provided by the Gamma project mesh database [170] and consists of 5226 triangular elements. It has been embedded into a hexahedral mesh of $24 \times 25 \times 16$ regular elements. Three layers of hexahedral elements have been removed around the ring model and the resulting space was filled with tetrahedral elements. One layer of pyramidal elements connects the tetrahedral mesh with the hexahedral mesh. A clipped version of the resulting mesh is depicted in Fig. 5.39b. Again, different element types are marked by different shades of gray and the outline of the regular hexahedral mesh is marked by thick black lines.

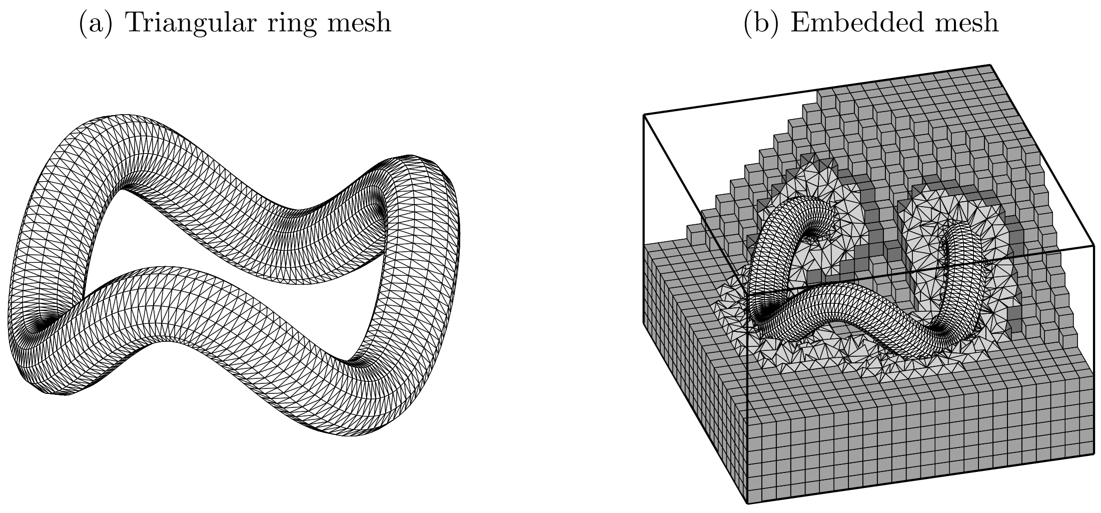


Figure 5.39: Triangular mesh of a ring and its embedding into a hexahedral mesh using a tetrahedral wrapper mesh and one layer of pyramidal elements. Different element types are marked by different shades of grey

The mesh consists of 74,106 tetrahedra, 7486 hexahedra, and 1618 pyramids resulting in a total of 83,210 volume elements. All nodes of the outer cuboid surface as well as all nodes of the ring surface have been kept fixed resulting in 14,246 movable nodes out of total of 25,531 nodes. Since the resulting initial mesh again was of already good quality, it has been additionally distorted by element validity preserving random node movements to complicate smoothing. The resulting initial mesh quality, runtime information, as well as the results obtained by the smoothing approaches described at the beginning of this section, are given in Table 5.14.

Table 5.14: Embedded ring smoothing results

Method/Criterion	Iter	Time (s)	q_{min}	q_{mean}
Initial	–	–	0.0005	0.4499
Smart Laplace	24	9.80	0.0005	0.6542
Global Optimization	72	14.00	0.1984	0.7534
GETMe	30/6700	7.78	0.2324	0.7437

As in the case of the first example, smart Laplacian smoothing was not able to improve the minimal element quality. However, the mean mesh quality was improved by 45.4%. In contrast, global optimization not only improved q_{mean} by 67.5% but also improved the minimal element quality significantly, although the method is mainly geared towards improving q_{mean}. Finally, GETMe smoothing even further improved $q_{\min}$ by 17.1% if compared to the value obtained by global optimization, and the mean mesh quality by 65.3% if compared to the initial mesh.

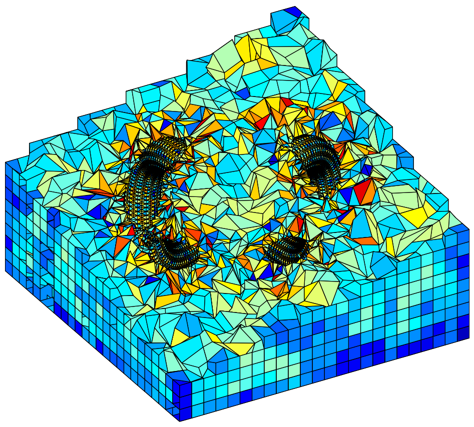
The achieved mesh quality is also reflected by the cross sections of the initial and smoothed versions of the mesh depicted in Fig. 5.40. In the case of smart Laplacian smoothing several clusters of low quality elements remain, since node updates are only performed if quality is improved. Standard Laplacian smoothing would have been able to resolve some of these clusters, but on the other hand would have led to a lower mean mesh quality and the generation of invalid elements. In contrast, global optimization and GETMe smoothing yielded valid meshes of high quality.

Compared to the first example, the number of low quality elements is significantly higher in the case of smart Laplacian smoothing, as can be seen in the element quality histogram depicted in Fig. 5.41. For example, the number of elements with a mean ratio quality number below 0.23 amounts to 12,634, 3269, 3, and 0 in the case of the initial mesh and those obtained by smart Laplacian smoothing, global optimization and GETMe smoothing, respectively. Although GETMe smoothing is a geometry-based method not using optimization techniques, its histogram resembles that of the global optimization approach.

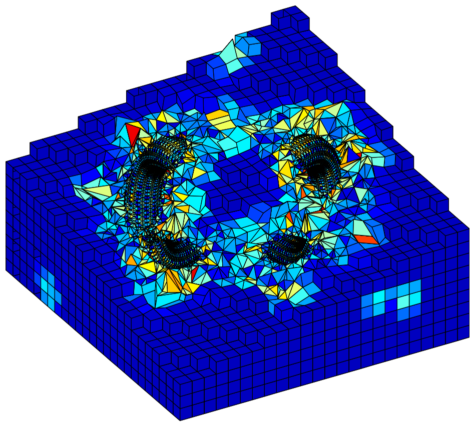
Finally, Fig. 5.42 depicts the mean mesh quality with respect to smoothing runtime. For the example under consideration, smart Laplacian smoothing performed 24 iterations within 9.8s. Four cycles of the shape improvement wrapper performing 72 feasible Newton iterations in the case of the global optimization approach took 14.0s in total. The simultaneous GETMe substep accomplished 30 iterations within 7.6s, followed by the sequential substep performing 6700 iterations within 0.2s, resulting in a total GETMe smoothing time of 7.8s. Again, a slightly lower mean mesh quality could be obtained much faster in the case of GETMe smoothing. For example, the threshold $q_{\text{mean}} = 0.73$ was obtained after 8 iterations of the simultaneous substep taking 2.0s in total. In contrast, global optimization required 44 iterations taking 8.6s in total to achieve this quality threshold.

In addition, the simultaneous GETMe approach is well suited for parallelization, since steps like the single element transformation and new node computation can be accomplished in parallel. For example, using the OpenMP [160] directive "`#pragma omp parallel for`" in order to parallelize these two steps resulted in an already good speedup factor of 1.7 for the named test system with a theoretical speedup limit of 2.0.

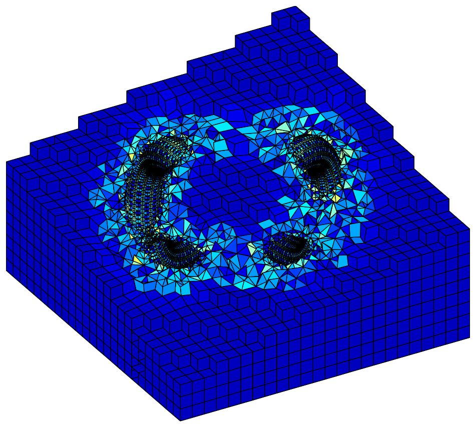
(a) Initial



(b) Smart Laplace



(c) Global Optimization



(d) GETMe

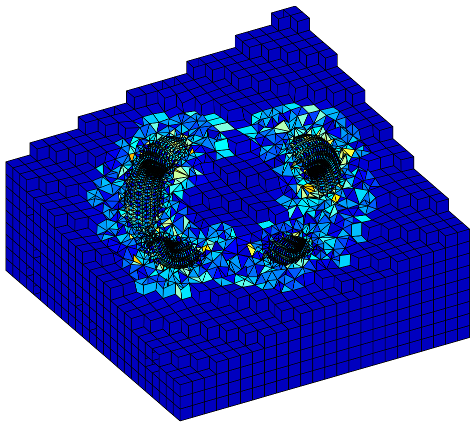


Figure 5.40: Initial and smoothed ring embedding meshes with elements colored according to their mean ratio quality number

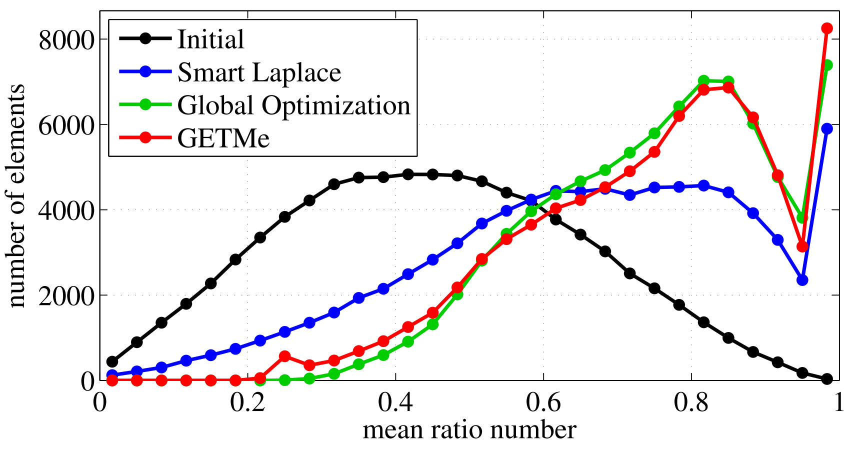


Figure 5.41: Embedded ring element quality histogram

5.4.3 Tetrahedral mesh wrapped with prismatic boundary layer

Tetrahedral meshes wrapped with layers of prisms are used for example in computational biofluid dynamic applications [10]. As an example, a part of an aorta will be considered in this section. The model, as well as its initial tetrahedral mesh, is provided courtesy of MB-AWG by the AIM@SHAPE Shape Repository [165]. It consists of 35,551 tetrahedral elements and has been wrapped with six layers of prisms with a constant height of 0.012. Here, the height was chosen small to avoid the intersection of layer elements as well as not to change the model surface too much. The wrapped model is depicted in Fig. 5.43b. In this, part of the prismatic boundary layer has been removed for illustration purposes. The part of the aorta represented by the model is marked by a blue rectangle in Fig. 5.43a. The scheme itself was created by J. Heuser and is provided by the Wikimedia Commons media archive [171].

Due to the minimal and average element quality numbers given by 0.2677 and 0.8098 respectively, the initial tetrahedral mesh is of already good quality. However, adding six layers of thin prisms reduces these numbers significantly as can be seen by the q_{min} and q_{mean} values of the initial mesh given in Table 5.15. The number of prisms amounts to 80,724 resulting in an initial mesh with 49,903 nodes and 116,275 elements in total.

The low quality of the prismatic boundary layers can also bee seen in the cross section of the initial mesh depicted in Fig. 5.44a. Although smart Laplacian smoothing is able to widen up the prismatic layers resulting in a mean mesh quality improvement of 86.4%, the minimal element quality halves if compared to the initial mesh. In particular, results obtained by this approach are not satis-

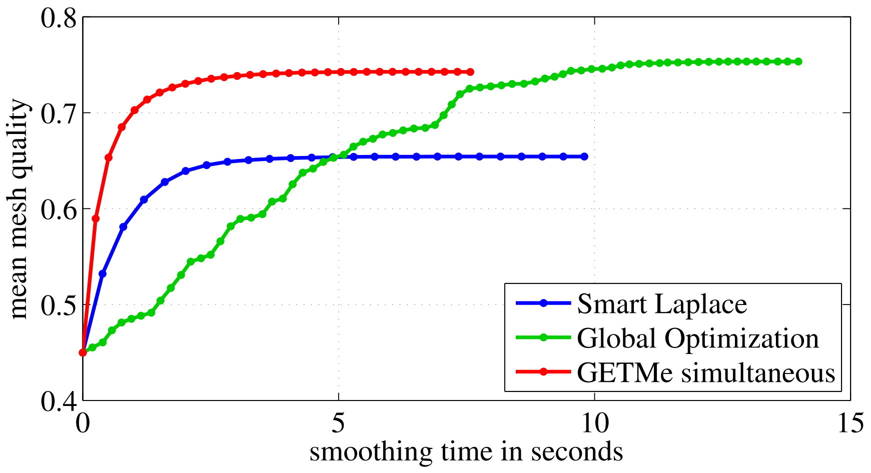


Figure 5.42: Embedded ring mean mesh quality with respect to smoothing runtime

Table 5.15: Aorta smoothing results

Method/Criterion	Iter	Time (s)	q_{min}	q_{mean}
Initial	–	–	0.0413	0.3077
Smart Laplace	98	120.84	0.0211	0.5737
Global Optimization	140	105.04	0.3895	0.8392
GETMe	251/66,300	95.65	0.5522	0.8231

factory if compared to the other two smoothing methods resulting in significantly better quality numbers. This is because smart Laplacian smoothing is based on a simple node averaging scheme being not quality driven. Here, quality only comes into account in the decision if node movements are applied depending on the mean quality improvement of adjacent elements. Due to the uniform prism heights, widening the prismatic boundary layers is mainly induced by the movement of the interface nodes shared by both element types. In addition, overall mesh improvement is hampered by the interior quality tetrahedral mesh, as can also be seen in Fig. 5.44. This does not hold for standard Laplacian smoothing leading to a significantly better mean mesh quality of 0.7542 but also to the generation of 372 invalid elements, thus resulting in an unusable mesh for most applications.

In contrast, since GETMe smoothing is based on regularizing element transformations and quality weighted node averaging, the prismatic layers are widened up by the transformation induced node movements. Hence, applying GETMe simultaneous resulted in a significantly better mean mesh quality of $q_{\text{mean}} = 0.8253$

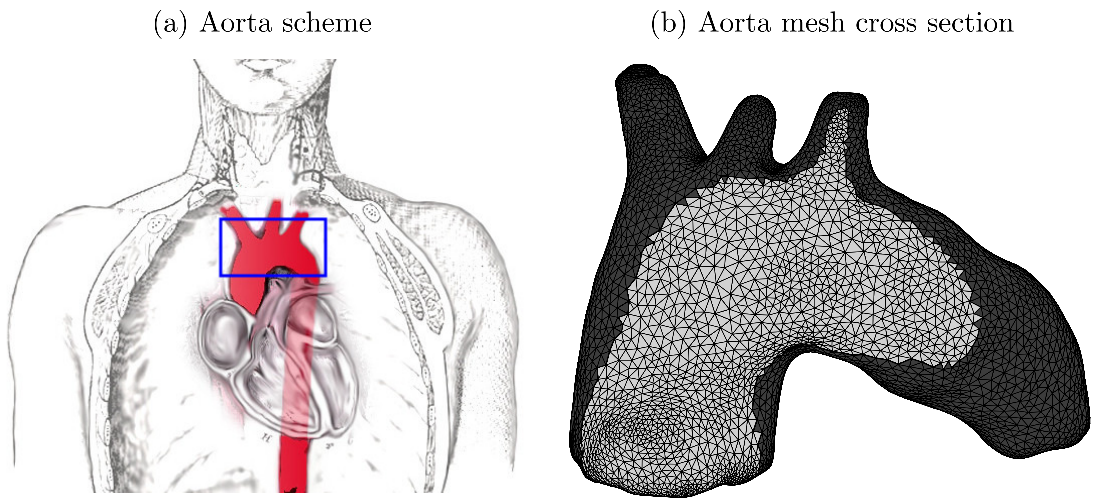


Figure 5.43: Schematic representation of an aorta, and its tetrahedral mesh with six prismatic boundary layers. Different element types are marked by different shades of grey

representing an improvement by 168.2% if compared to the initial mesh quality. The subsequently applied sequential substep of GETMe smoothing slightly reduced this mean mesh quality number to $q_{\text{mean}} = 0.8231$ while improving the minimal element quality by factor 13.4, if compared to the initial mesh.

Due to its global optimization-based approach, the shape improvement wrapper of Mesquite also widened the prismatic boundary layer significantly, resulting in a minimal quality improvement by factor 9.4 and a mean mesh quality improvement by 172.7% if compared to the initial mesh.

The big difference in mesh quality between smart Laplacian smoothing on the one hand, and global optimization-based and GETMe smoothing on the other hand is also obvious from the quality histogram depicted in Fig. 5.45. Here, the mean initial prisms quality of 0.0866 is reflected by a peak of the initial mesh histogram. For sake of clarity, this peak representing 49,604 elements in the quality bin $[1/15, 1/10)$ is not visible. The mean quality of all prisms is improved by smart Laplacian smoothing to 0.4796. However, the prisms mean quality numbers 0.8440 and 0.8586 achieved by GETMe smoothing and global optimization are nearly twice as large. Furthermore, there are a large number of low quality elements in the case of smart Laplacian smoothing. For example, the number of elements with a mean ratio number below 0.5522 amounts to 81,381, 58,385, 587 and 0 in the case of the initial mesh, and those obtained by smart Laplacian smoothing, global optimization, and GETMe smoothing respectively.

For the given example, smart Laplacian smoothing terminated after 98 iterations taking 121s in total. Due to the local oriented approach of GETMe

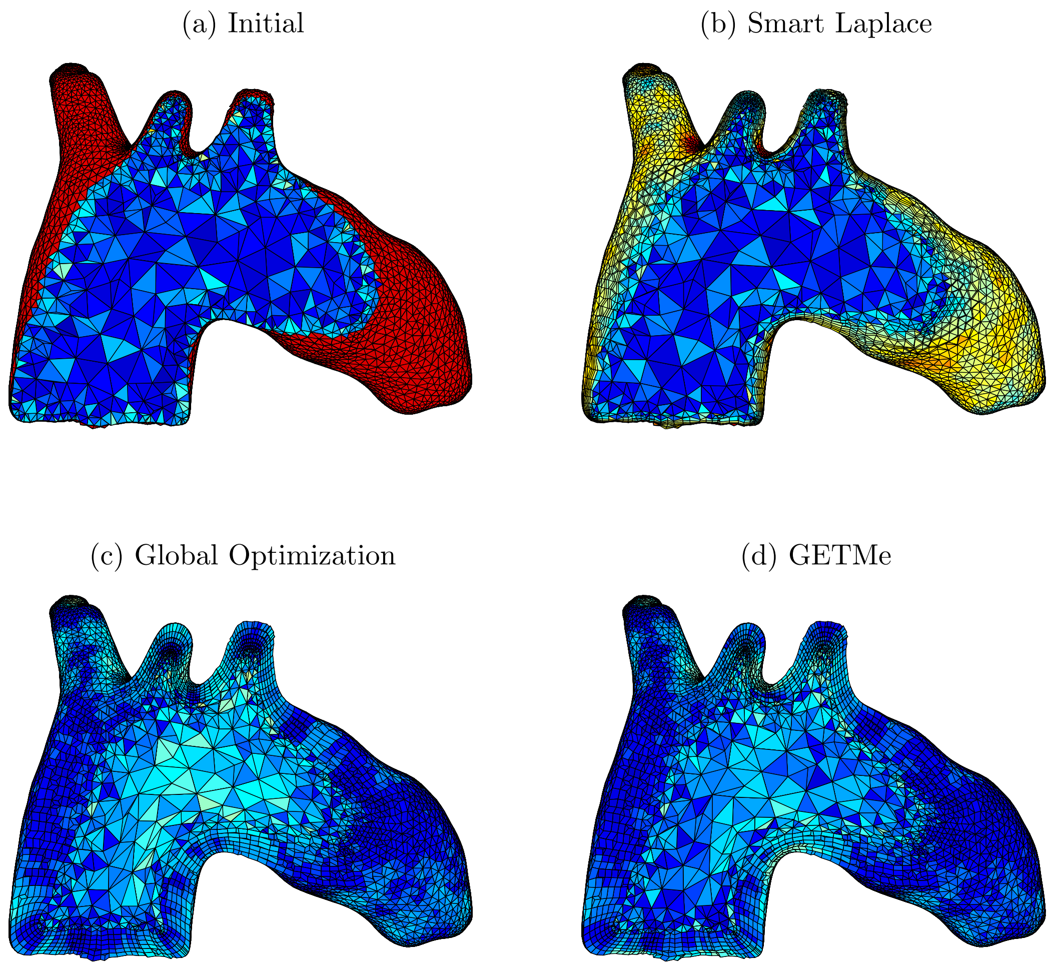


Figure 5.44: Initial and smoothed aorta meshes with elements colored according to their mean ratio quality number

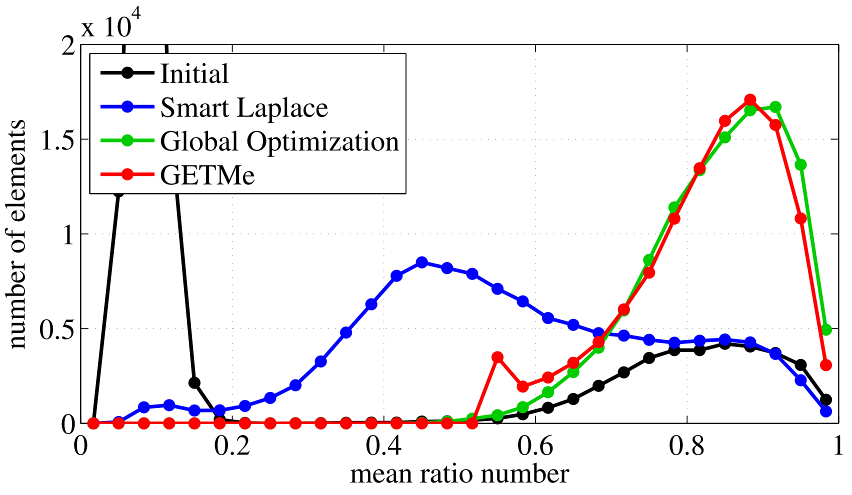


Figure 5.45: Aorta element quality histogram

smoothing and the thin initial prismatic boundary-layers, GETMe simultaneous performed a comparatively large number of 251 iterations taking 93s in total. That is, one iteration of GETMe simultaneous smoothing is about 3.3 times faster if compared to one step of smart Laplacian smoothing. This is caused by the lower number of element quality evaluations, which, in the case of the mean ratio quality criterion, are comparatively expensive. The subsequently applied sequential GETMe approach performed 66,300 iterations within 3s resulting in a total GETMe smoothing time of about 96s. Mesquite performed 140 feasible Newton iterations within three cycles of the shape improvement wrapper, taking 105 seconds in total. Since one iteration of GETMe simultaneous smoothing is about two times faster compared to one feasible Newton iteration, the overall smoothing time of GETMe smoothing is shorter, although a larger number of iterations was performed.

As in the previous examples, the first iterations of the simultaneous GETMe substep led to a sharp rise in mesh quality. Hence, practically good meshes can be obtained in significantly shorter runtimes. For example, GETMe simultaneous achieved $q_{\text{mean}} = 0.6$ within 14 iterations taking 6s, hence being about 15.7 times faster than the global optimization approach, which required 128 iterations taking 94s. This is also caused by a slow mean quality improvement of the first shape improvement wrapper cycle of Mesquite, which achieved $q_{\text{mean}} = 0.5956$ after 127 iterations. The second cycle led to a more rapid improvement, resulting in $q_{\text{mean}} = 0.8392$ after ten additional iterations. However, even the quality threshold $q_{\text{mean}} = 0.8$ was achieved about 2.4 times faster by GETMe smoothing.

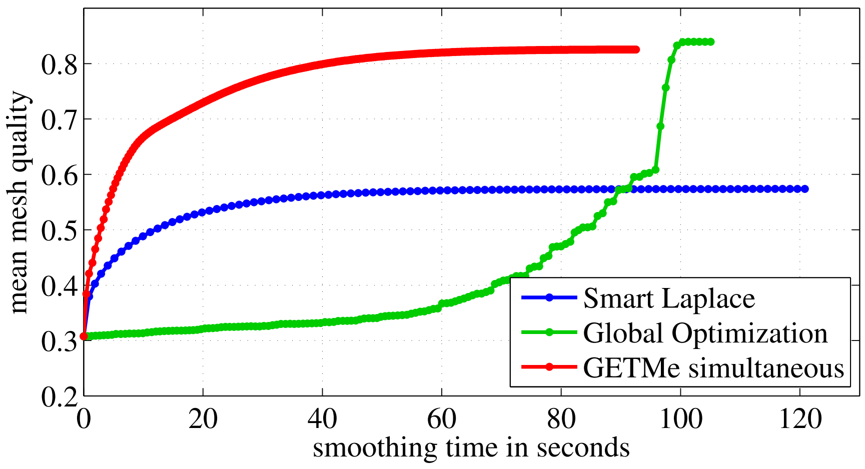


Figure 5.46: Aorta mean mesh quality with respect to smoothing runtime

5.4.4 Mean ratio and warpage-based GETMe smoothing

In the previous sections, mesh quality was assessed using the mean ratio quality criterion only. However, other quality criteria can be incorporated as is described in Section 4.7 introducing a combined GETMe approach being based on the mean ratio element quality criterion as well as the warpage criterion for measuring the quality of quadrilateral element faces. For this approach, named GETMe q_γ, and the mixed volume meshes introduced in the preceeding three sections, smoothing results are given in Table 5.16.

In this, the first four rows of each example contain the mean ratio and warpage values for the meshes obtained in the previous examples. As can be seen, global optimization as well as the mean ratio-based GETMe smoothing led to comparable mean warpage values. Using the example of the embedded ring model, Fig. 5.47 depicts a considerable reduction of w_{mean} within the first few iterations of GETMe simultaneous. As a result, GETMe smoothing reached the threshold $w_{\text{mean}} = 0.01$ more than eight times faster than the global optimization-based approach, whereas smart Laplacian smoothing did not reach this threshold. In addition, smart Laplace generated strongly warped faces in the case of the ring and aorta model. However, compared to the results of the global optimization-based approach, GETMe smoothing led to increased $w_{\max}$ values for the embedded plate and ring model.

Such low quality elements can effectively be improved by a mean ratio and warpage-based GETMe sequential substep as it is described in Section 4.7. Results obtained for such a q_γ-based GETMe sequential approach applied to the GETMe smoothed meshes of the previous examples are given in the rows of Ta-

Table 5.16: Mesh quality with respect to the mean ratio (q) and warpage (w) quality criterion for previous examples and results after applying the combined GETMe approach (denoted as GETMe q_γ). Larger mean ratio values indicate elements of better quality, whereas lower warpage values indicate quadrilateral faces of better quality

Mesh	Method	q_{min}	q_{mean}	w_{max}	w_{mean}
Plate	Initial	0.1625	0.7302	0.7368	0.0100
	Smart Laplace	0.0258	0.8195	0.7752	0.0139
	Global Optimization	0.3400	0.8382	0.6529	0.0154
	GETMe	0.4476	0.8300	0.9192	0.0194
	GETMe q_γ	0.2346	0.8298	0.2972	0.0179
Ring	Initial	0.0005	0.4499	1.8157	0.4498
	Smart Laplace	0.0005	0.6542	1.5461	0.0566
	Global Optimization	0.1984	0.7534	0.8013	0.0095
	GETMe	0.2324	0.7437	1.0856	0.0071
	GETMe q_γ	0.2324	0.7435	0.1173	0.0047
Aorta	Initial	0.0413	0.3077	0.6028	0.0092
	Smart Laplace	0.0211	0.5737	1.9245	0.1203
	Global Optimization	0.3895	0.8392	0.9915	0.0247
	GETMe	0.5522	0.8231	1.0117	0.0355
	GETMe q_γ	0.4359	0.8231	1.0000	0.0355

ble 5.16 denoted by GETMe q_γ. As can be seen, this method led to improved results with respect to q_{min} as well as w_{max} in 4 of 6 cases if compared to global optimization. In addition, results with respect to the mean quality numbers q_{mean} and w_{mean} are comparable. Here, for all three examples, the transformation and penalty parameters of the q_γ-based GETMe sequential smoothing substep have been set to those of the mean ratio-based sequential substep given in the beginning of section 5.4. In addition, the quality weight $\gamma = 0.25$ has been used.

Due to the incorporation of two quality criteria, the q_γ-based sequential substep has an increased computational complexity. Furthermore, mesh quality was assessed after each iteration, since quality changes more rapidly if compared to the mean ratio-based GETMe sequential approach. This resulted in the smoothing runtimes 0.44s, 2.10s, and 0.01s for the plate, ring and aorta example requiring 2555, 7068, and 218 iterations, respectively. Nevertheless, the overall runtimes of the GETMe q_γ approach amounting to 3.14s, 9.88s, and 95.66s are significantly lower than those of the global optimization-based approach requiring 9.72s, 14.00s, and 105.04s, respectively.

Differences between the two quality criteria are evident for the aorta example.

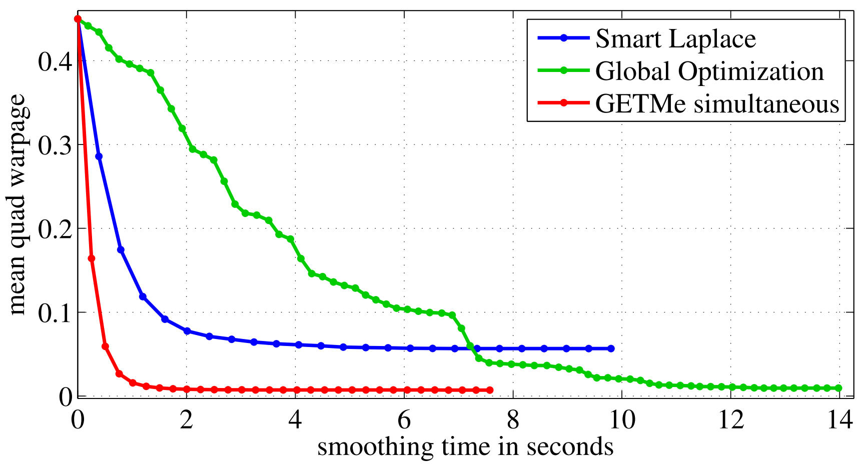


Figure 5.47: Mean quadrilateral warpage w_{mean} with respect to smoothing time for embedded ring example. Lower values indicate better quality quadrilateral faces

Here, the initial mesh has been constructed by wrapping a tetrahedral mesh with six prismatic layers of constant height. Due to this, the nodes of the quadrilateral faces are almost coplanar, which results in an excellent initial mean warpage of $w_{\text{mean}} = 0.0092$. In contrast, due to the low heights, these prisms are strongly non-regular yielding a very low average mean ratio value of $q_{\text{mean}} = 0.3077$. This is is also visible by the initial mesh cross section depicted in Fig. 5.44a. Applying global optimization and GETMe smoothing widened up these layers of prisms resulting in a considerable improvement of its mean ratio quality. At the same time, the warpage values increased, since the prismatic layer heights and with this the quadrilateral element faces became less uniform. This shows that both quality criteria can result in opposing smoothing objectives. Nevertheless, in the case of q_γ-based GETMe smoothing the quality weight γ allows a good compromise to be found.

For the given examples, numerical tests have shown that applying the sequential q_γ-based GETMe approach to the results of GETMe smoothing leads to better results than applying it to the results of the mean ratio-based simultaneous substep. Since the same transformation and penalty parameters have been applied during the two sequential substeps, this can be interpreted as one sequential substep incorporating a change of the quality criterion.

Further improvements are expected by an adaptive choice of transformation parameters. Here, numerical tests based on parameter variations have shown that similar GETMe q_γ results as given in Table 5.16 can be achieved within significantly shorter runtimes. Furthermore, $w_{\max}$ values have been further reduced by

12.1 %, 63.3 %, and 2.7 % for the plate, ring and aorta model by using individual smoothing parameters. It is noted however, that these results should not be compared with those of smart Laplacian smoothing and the global optimization-based approach, since the latter are tuned to improve mesh quality with respect to the mean ratio criterion only.

This first test demonstrates the flexibility of the GETMe sequential smoothing approach with respect to the quality criterion used for smoothing control. A similar potential for improvement is expected for an accordingly adjusted GETMe simultaneous approach. However, since in this case element quality is also incorporated in the computation of element transformation parameters and the weighted node averaging scheme, adjustments are more closely bound to the characteristics of the quality criterion. Further research regarding these topics has to be accomplished within the context of specific applications and their individual requirements for mesh quality.

5.5 Adaptive GETMe smoothing examples

For two generic meshes this section provides a comparison of results obtained by GETMe adaptive smoothing as well as smart Laplace, a global optimization-based approach, and basic GETMe smoothing. It will also discuss implementational aspects by comparing results for the straightforward C++ implementation of GETMe smoothing with those of a more efficient C implementation of GETMe adaptive smoothing.

5.5.1 Test description

In the case of GETMe smoothing, the default parameters as given in Section 5.4 and Chapter 6 have been used. For GETMe adaptive smoothing, the maximum number of iterations has been set to 1000 and the q_{mean} improvement tolerance to 10^{-4}. The tetrahedral mesh has been smoothed by using the opposite face normal transformation. For the hexahedral mesh, the dual element-based transformation has been applied. In both cases, the fixed element transformation parameter $\sigma = 3/2$ was used. The tables of relaxation values have been set to $R = (1, 1/4, 1/16, 0)$ and $R = (1/2, 1/4, 1/10, 1/100, 0)$ in the case of the q_{mean} and q^*_{min} oriented smoothing stage, respectively. Each q^*_{min} oriented smoothing cycle has been terminated after the NoMinImproveCounter reached five iterations.

Results of both GETMe variants are compared to those of smart Laplacian smoothing, which was implemented using the same data structures as the GETMe adaptive approach. Thus it also benefits from the lean data management of the C implementation. In addition, results for an OpenMP-based parallelized smart

Laplacian smoothing approach are provided. Here, testing if a node update would increase local mesh quality is applied in parallel for all nodes. In this case, nodes are not updated directly in order to assure reproducibility of results and the independence of the node numbering scheme. Instead, similar to GETMe adaptive, new node positions are stored in p'_i, which is interchanged with p_i at the end of the iteration. In the case of invalid element generation, nodes are iteratively reset to their original coordinates until the mesh is valid again. Smoothing is terminated, if the q_{mean} values of two consecutive iterations differ by less than 10^{-4}.

The quality of the results is compared to the state of the art global optimization-based approach provided by the shape improvement wrapper of the mesh quality improvement toolkit Mesquite version 2.2.0 [21, 137]. This algorithm has been applied iteratively until the q_{mean} improvements of two consecutive calls dropped below 10^{-4}.

Mesquite also provides parallel mesh smoothing based on smoothing submeshes and synchronizing submesh interfaces using MPI [172]. However, for the given examples, the shape improvement wrapper failed due to the detection of termination criteria issues, which would have led to infinite loops. Therefore, as a substitute, optimal results have been assumed for the parallel global optimization-based smoothing approach. That is, the sequential runtime has been divided by the number of processor cores of the test system to provide an ideal case estimate of the parallel runtime. In practice, runtimes are expected to be significantly higher due to the usage of shared resources.

The GETMe approach and the Mesquite toolkit are implemented in C++. In contrast, GETMe adaptive and smart Laplacian smoothing are implemented in C. All programs have been compiled using the GNU Compiler Collection version 4.7.1 [162]. Computations have been accomplished on a personal computer with an Intel® Core™ i7-870 CPU (quad core, 8 MB cache, 2.93 GHz), 16 GB RAM, and a 64 bit Linux operating system. Here, hyper-threading has been deactivated and all four cores of the processor have been used for parallel computations. Thus, the ideal speedup factor for the parallel implementation is 4.

5.5.2 Tetrahedral mesh example

The first example considers the piston model depicted in Fig. 5.48. It was constructed by completing a partial model provided by the Drexel University Geometric & Intelligent Computing Laboratory model repository [173].

A tetrahedral mesh consisting of 729,923 nodes and 4,129,608 elements has been generated by Delaunay tetrahedralization resulting in a highly unstructured mesh with the number of tetrahedra attached to individual nodes ranging from 2 to 48. This mesh was distorted by random element validity preserving node

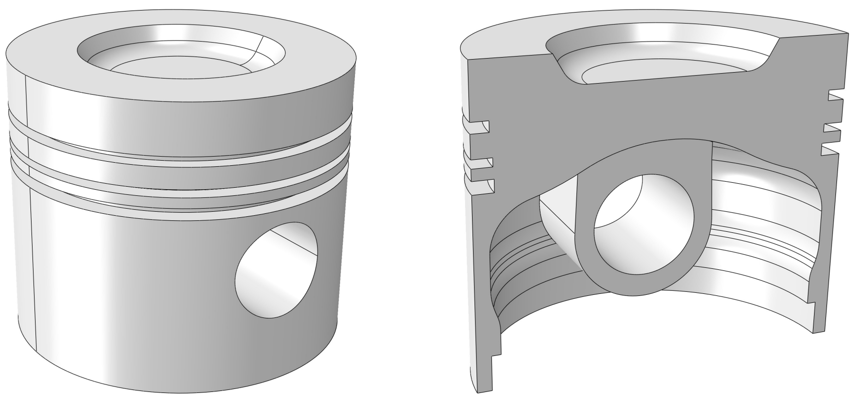


Figure 5.48: Full piston model (left) and cross section (right)

movements in order to increase the smoothing potential of the mesh. The resulting mesh has been smoothed with the C implementation of the smart Laplacian smoothing approach, the C++ based shape improvement wrapper of Mesquite, the C++ implementation of GETMe, and the new C implementation of GETMe adaptive, as described in the previous sections. Cross sections of the resulting meshes are depicted in Fig. 5.49. Here, each element is colored according to its mean ratio quality number, where reddish colors indicate low quality elements and bluish color high quality elements.

As can be seen in Fig. 5.49a, elements of the initial mesh are severely distorted to challenge all smoothing methods. At first sight it seems that the mesh quality obtained by smart Laplacian smoothing is comparable to those of the other smoothing methods. However, a closer look reveals elements of very low quality, which can also be verified by the worst element quality number $q^*_{\min} = 0.0001$ given in Table 5.17. In contrast, the corresponding quality numbers of the global optimization and GETMe-based methods are significantly better. For example, the numbers of elements E_j with $q^*_{\min} \leq q(E_j) \leq 0.4$ amount to 1,798,777 for the initial mesh and 4463, 164, 514, 32 for its variants smoothed by smart Laplace, global optimization, GETMe, and GETMe adaptive, respectively. In the case of smart Laplace, the mean ratio quality of 465 elements is even below 0.1, which may lead to numerical instabilities in subsequent finite element computations. Nevertheless, the mesh is valid, which is not the case if classical Laplacian smoothing is used instead.

The increased number of elements with $q(E_j) \leq 0.4$ in GETMe smoothing is due to its approach of consecutively improving the worst elements, which leads to an accumulation of elements with a quality number slightly above $q^*_{\min}$. This effect is ameliorated by the adaptive approach, which also uses the weighted node averaging scheme of Equation (4.3) during the $q^*_{\min}$-oriented smoothing stage instead of setting transformed nodes directly. Furthermore, both quality numbers

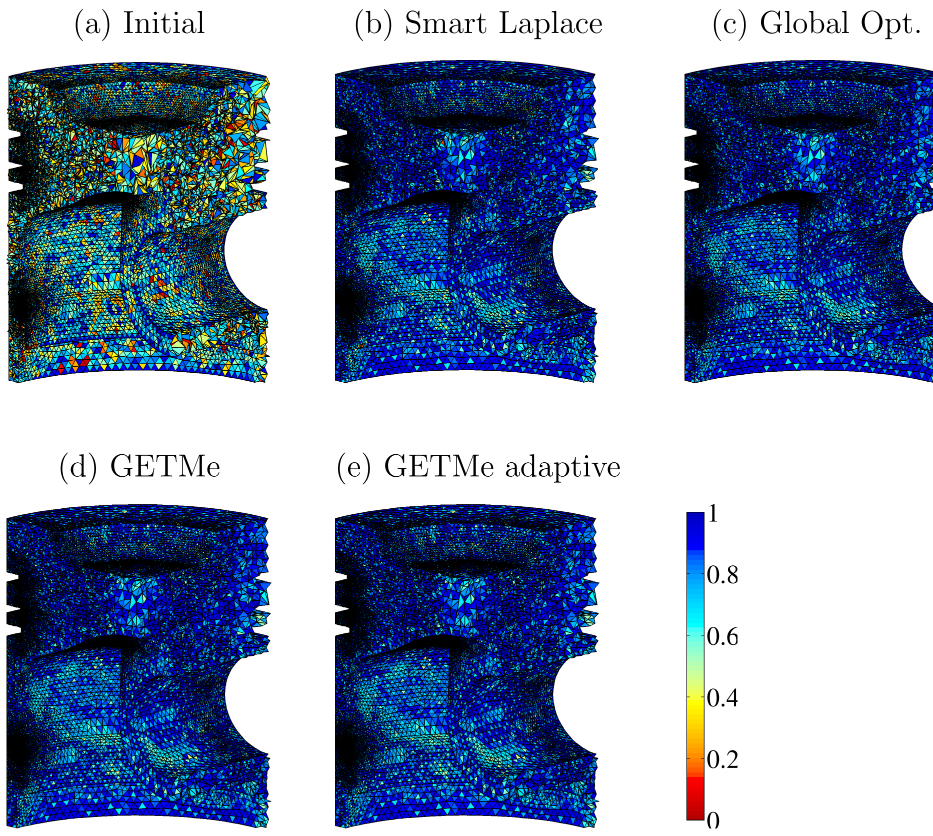


Figure 5.49: Piston mesh cross sections with elements colored according to their mean ratio quality number

obtained by GETMe adaptive smoothing are slightly better than those of GETMe smoothing.

Table 5.17 also provides the maximum memory size of the application measured in gibibytes (2^{30} bytes), the smoothing time in seconds, as well as the number of iterations. In the case of GETMe smoothing, the iteration number of the simultaneous and sequential substeps are given. Similarly, for GETMe adaptive iteration numbers are given for the q_{mean} and for the q^*_{min}-oriented stage of the smoothing process.

Due to its simple approach, memory requirements of smart Laplacian smoothing are low. The large amount of memory used in the case of the C++ implementation of GETMe smoothing is not caused by the requirements of the algorithm, but by the use of general data structures storing redundant topology and statistical informations. This can also be seen by the comparably low memory requirements of the GETMe adaptive approach implemented in C. Thus, an implementation of GETMe smoothing using similar lean data structures would result in a

Table 5.17: Piston mesh smoothing results with (P) indicating parallel versions, (OP) denoting an estimate for optimal parallelization

Name	Mem [GiB]	Time [s]	Iter	$q_{\min}^*$	q_{mean}
Initial	–	–	–	0.0000	0.4506
Smart Laplace	0.3	28.90	12	0.0001	0.8584
Global Opt.	2.0	791.42	149	0.2963	0.8654
GETMe	5.7	74.41	12/5500	0.3336	0.8636
GETMe adaptive	1.6	40.05	15/47	0.3421	0.8637
Smart Laplace (P)	0.4	9.18	12	0.0001	0.8584
Global Opt. (OP)	2.0	197.85	149	0.2963	0.8654
GETMe adaptive (P)	1.6	17.29	15/47	0.3421	0.8637

significantly lower memory profile comparable to that of GETMe adaptive. However, the results for the unmodified C++ implementation of GETMe smoothing are included for consistency with previous publications. Thus improvements with respect to memory requirements and smoothing time demonstrated in this work could also be obtained for the examples given in previous publications.

On average, one iteration of smart Laplacian smoothing requires 2.41s. It can be seen that smart Laplacian smoothing results in the lowest overall smoothing time. In contrast, the global optimization-based approach, due to the high average smoothing time of 5.31s per iteration and the large number of iterations, takes 27.4 and 19.8 times longer if compared to smart Laplacian smoothing and GETMe adaptive smoothing, respectively. For the latter, the average time of 2.57s per iteration during the first stage is only slightly larger than that of smart Laplacian smoothing with 2.41s per iteration, despite the fact that the elements are transformed. This is due to the lower number of mean ratio quality evaluations. Furthermore, the average runtime per iteration of 0.03s during the second smoothing stage of GETMe adaptive is low.

Mesh quality with respect to smoothing time is depicted in Fig. 5.50. As can be seen by the results for the minimal element quality number $q_{\min}^*$ given on the left, smart Laplacian smoothing leads to almost no improvement. In contrast, GETMe and GETMe adaptive lead to a sharp rise at the beginning of the q_{mean}-oriented stage, as well as during the complete $q_{\min}^*$-oriented stage. Improvements of the global optimization-based approach are achieved within the last quarter of its runtime, which prohibits a preliminary termination from an application point of view.

The second part of Table 5.17 also provides results for the parallel versions. Again, it is pointed out that in the case of global optimization a lower bound for

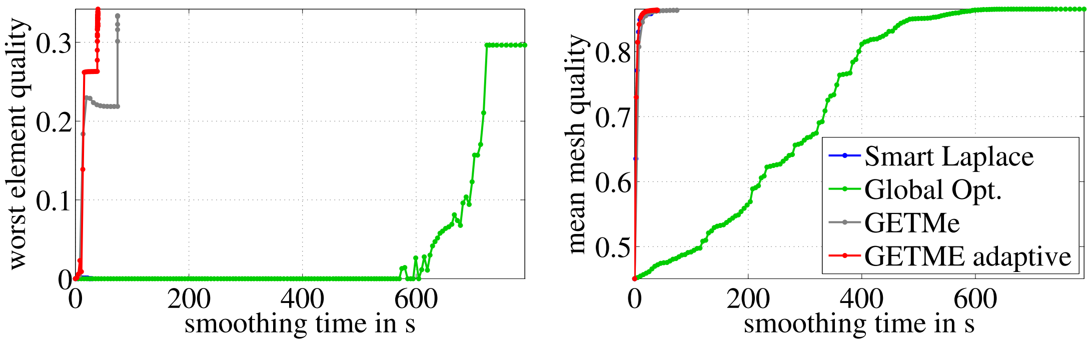


Figure 5.50: Piston mesh $q^*_{\min}$ (left) and q_{mean} (right) with respect to smoothing time for sequential implementations

the runtime is given, by assuming that the method achieves the optimal speedup factor 4. The actual speedup factors of smart Laplacian smoothing and GETMe adaptive smoothing reach 3.1 and 2.3, respectively. As can be seen, the speedup of GETMe adaptive is inferior due to to the incorporation of atomic memory operations and a larger amount of data to be transferred.

5.5.3 Hexahedral mesh example

The second example considers the pump carter model depicted in Fig. 5.51, of which a STEP-file was provided courtesy of Rosalinda Ferrandes, Grenoble INP, by the AIM@SHAPE Shape Repository [165].

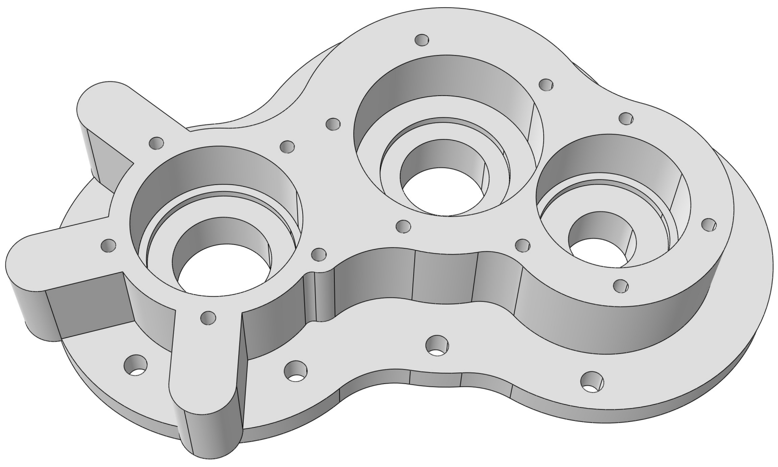


Figure 5.51: Pump carter model

An all-hexahedral mesh consisting of 2,779,096 nodes and 2,646,976 elements has been generated by sweeping a quadrilateral surface mesh along the z-axis resulting in an almost regular hexahedral mesh, where 90% of the nodes are shared by eight hexahedra each. The number of attached elements of the remaining 10% of nodes ranges from two to ten. As in the case of the first example, this mesh was distorted by element validity preserving random node movements resulting in the initial mesh. The latter has been subsequently improved by all smoothing methods under consideration. Cross sections of the resulting meshes are depicted in Fig. 5.52. Compared to the case of the tetrahedral mesh, the ratio of low quality elements is further increased in the case of smart Laplacian smoothing. For example, the numbers of elements E_j with $q^*_{\min} \leq q(E_j) \leq 0.5$ amounts to 1,586,570, 18,509, 23, 4, and 0 in the case of the initial mesh and those smoothed by smart Laplace, global optimization, GETMe, and GETMe adaptive, respectively. Here, the large number of low quality elements in the case of smart Laplacian smoothing is also reflected by the decreased mean mesh quality given in Table 5.18. As in the case of the previous example, applying classical Laplacian smoothing invalidates the mesh.

Compared to global optimization and GETMe smoothing, GETMe adaptive smoothing leads to a further improvement of $q^*_{\min}$. In all three cases, mean mesh quality numbers are near the optimal value one, which is also reflected by the bluish element colors in the cross sections given in Fig. 5.52.

Table 5.18: Pump carter mesh smoothing results with (P) indicating parallel versions and (OP) denoting an estimate for optimal parallelization

Name	Mem [GiB]	Time [s]	Iter	$q^*_{\min}$	q_{mean}
Initial	–	–	–	0.0385	0.4709
Smart Laplace	0.7	344.62	30	0.0604	0.9393
Global Opt.	4.4	5241.92	269	0.4501	0.9766
GETMe	2.9	227.25	24/31800	0.4667	0.9720
GETMe adaptive	0.9	88.97	25/43	0.5442	0.9719
Smart Laplace (P)	0.9	103.84	30	0.0604	0.9393
Global Opt. (OP)	4.4	1310.48	269	0.4501	0.9766
GETMe adaptive (P)	0.9	33.81	25/43	0.5442	0.9719

Again, the maximum memory size of smart Laplacian smoothing and GETMe adaptive is low, if compared to the C++ implementations of GETMe and global optimization. For example, global optimization requires five times more memory than GETMe adaptive. Furthermore, GETMe adaptive smoothing is 3.9, 58.9, and 2.6 times faster, compared to smart Laplacian smoothing, global optimization, and GETMe smoothing, respectively. Here, smart Laplacian smoothing is slower than both GETMe variants, due to the larger number of element quality

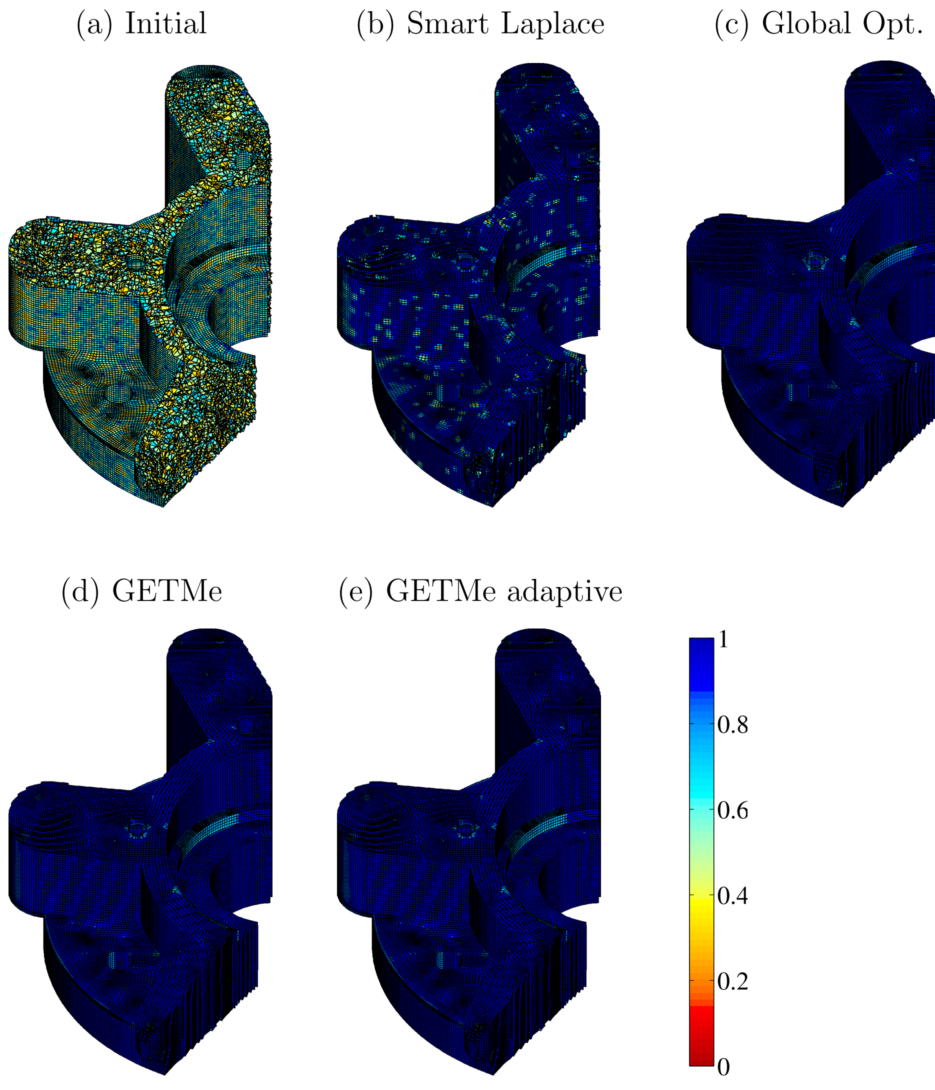


Figure 5.52: Pump carter mesh cross sections with elements colored according to their mean ratio quality number

evaluations and the fact that determining the mean ratio of hexahedra requires the computation of eight 3×3 determinants and Frobenius norms each.

This increased speedup obtained by GETMe adaptive is also apparent in the quality with respect to smoothing time graphs depicted in Fig. 5.53. As can be seen, global optimization leads to a decrease of the worst element quality during the first iteration, which cannot be resolved during almost 4000 smoothing seconds. In contrast, the same method leads to a slow but steady improvement of the mean mesh quality within that time. The convergence behavior of smart Laplacian smoothing and both geometry-based approaches differs significantly. Here, the first few iterations lead to a sharp rise of mesh quality with respect to both, q^*_{min} and q_{mean}.

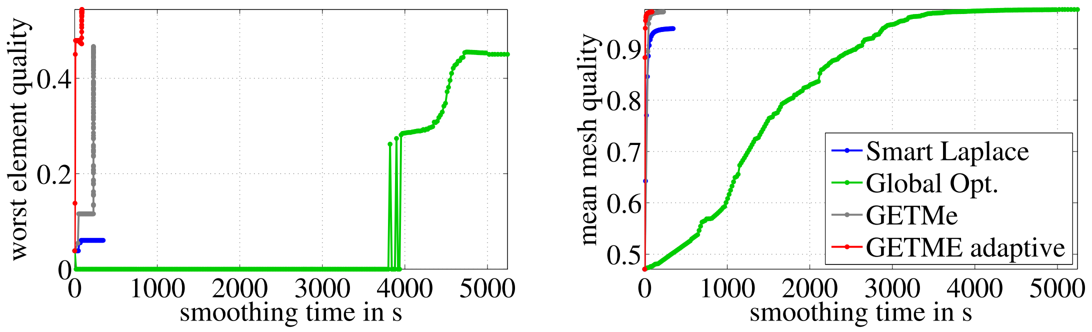


Figure 5.53: Pump carter mesh q^*_{min} (left) and q_{mean} (right) with respect to smoothing time for sequential implementations

Table 5.18 also provides results for the parallel versions of smart Laplacian smoothing and GETMe adaptive based on an OpenMP-approach. Here, the speed up by applying smart Laplacian smoothing in parallel is 3.3 compared to the sequential version. The speed up of parallel GETMe adaptive smoothing is 2.6. Furthermore, the parallel version of GETMe adaptive smoothing is 3.1 and 38.8 times faster compared to the parallel version of smart Laplacian smoothing and the ideal parallelization of the global optimization-based approach, respectively.

Chapter 6

Improving finite element simulation by GETMe smoothing

In this section, the influence of the improvement of mesh quality with the GETMe smoothing technique on finite element solution accuracy and computational efficiency will be demonstrated [37]. Fig. 6.1 depicts the main steps in the finite element-based simulation process, where the mesh generation and the assembly of the system of linear equations will not be addressed in this study. Hence, these steps are marked by dotted lines.

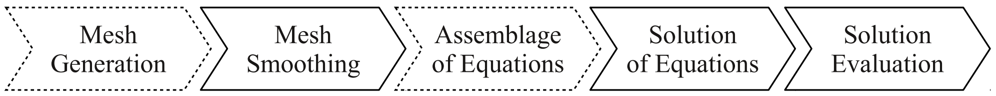


Figure 6.1: Steps of the finite element simulation process

Due to the universal approach of the geometric element transformation method, smoothing of surface and volume meshes differ only by the involved element transformations, which have to be chosen according to the type of the mesh elements. To substantiate this, in the following examples a common set of GETMe control parameters was used for the 2D as well as the 3D meshes. With regard to the subsequently performed finite element computations, these parameters increase the smoothing speed while maintaining high mesh quality.

In the case of the GETMe simultaneous substep, mean element edge length preserving scaling, and no relaxation, i.e. $\varrho = 1$, have been applied. Furthermore, by using the exponent $\eta = 0$, the weighted node averaging scheme simplifies to the arithmetic mean, which further reduces the numerical complexity. In the case of the GETMe sequential substep likewise mean element edge length preserving scaling was used. Furthermore, the relaxation parameter has been set to $\varrho = 1/100$, and the penalty parameters $\Delta\pi_i = 10^{-4}$, $\Delta\pi_r = 10^{-5}$, and

$\Delta\pi_s = -10^{-3}$ were applied in all cases. These parameters and the element type specific transformation parameters stated in the subsequent sections have been obtained on the basis of the examples of the previous chapter and the results of a series of parameter variation test for a collection of meshes of different type.

The parameter variation tests also indicate that GETMe simultaneous, due to its averaging approach, is particularly stable within its parameters. That is, similar parameter sets lead to similar results. In comparison, due to its single element transformation approach, GETMe sequential is more sensitive with respect to its transformation and relaxation parameters. For example, using parameters leading to a rapid change of shape of the lowest quality element might invalidate neighboring elements.

All smoothing methods considered were implemented in C++ and have been compiled with the GNU C++ compiler version 4.5.1. Runtimes were measured on a personal computer with an Intel® Core™ i7-870 CPU (quad core, 8 MB cache, 2.93 GHz), 16 GB RAM, and a 64 bit Linux operating system.

6.1 Model Problem

As a model problem, Poisson's equation with Dirichlet boundary conditions is considered. This can be interpreted as a steady state heat conduction problem on a domain $\Omega \subset \mathbb{R}^n$, $n \in \{2, 3\}$, with homogeneous material and prescribed temperature on the domain's boundary $\partial\Omega$. Setting the thermal diffusivity to one results in the elliptic boundary value problem

$$-\nabla^2 u = f \quad \text{on } \Omega\,, \tag{6.1}$$

$$u = g \quad \text{on } \partial\Omega\,, \tag{6.2}$$

where ∇^2 denotes the Laplace operator [174]. Here, the solution $u : \overline{\Omega} \to \mathbb{R}$ describes the temperature defined over the closure $\overline{\Omega}$ of Ω, $f : \Omega \to \mathbb{R}$ represents the heat source, and $g : \partial\Omega \to \mathbb{R}$ provides the prescribed boundary temperature.

In order to preserve in the following examples the initial boundary conditions of the finite element computation, boundary nodes will be kept fixed by all smoothing methods applied. This also avoids additional influences caused by the usage of different shape preservation techniques.

Since inner node positions of meshes improved by different smoothing methods do not match, the corresponding nodal finite element solutions cannot be compared directly. Comparing interpolated values instead is also problematic because of the additional interpolation error. However, in the ideal case, the finite element solution can be compared with the analytic solution. Therefore, in the following examples an arbitrarily chosen analytic solution u was prescribed for a given domain. From this, the boundary value problem was derived and solved by a finite element approach. This allows the determination of exact solution errors for arbitrarily complex domains Ω.

In order to compute the numerical solution, the heat source function f given by the left hand side of equation (6.1), as well as the boundary conditions according to equation (6.2) are required as input data. For this purpose, f has been determined by applying the Laplace operator to the given analytic solution u and interchanging signs. Dirichlet boundary conditions have been set to $g := u|_{\partial\Omega}$, i.e. by evaluating the analytic solution for all boundary nodes with the associated index set $I_{\partial\Omega}$. Indices of interior mesh nodes are given by the set $I_\Omega := I_p \setminus I_{\partial\Omega}$.

Let $\tilde{u} : \overline{\Omega} \to \mathbb{R}$ denote the finite element approximation of the prescribed analytic solution u. Solution accuracy is assessed by means of the nodal error numbers

$$\begin{aligned} e_{\max} &:= \max_{i \in I_\Omega} |u(p_i) - \tilde{u}(p_i)| \,, \\ e_{\text{mean}} &:= \frac{1}{|I_\Omega|} \sum_{i \in I_\Omega} |u(p_i) - \tilde{u}(p_i)| \,, \end{aligned}$$

which represent the maximal and mean temperature deviation for all inner mesh nodes. Here, $|I_\Omega|$ denotes the number of these nodes. In addition,

$$\begin{aligned} e_{L^2} &:= \sqrt{\int_\Omega |u - \tilde{u}|^2 \, dx} \,, \\ e_{H^1} &:= \sqrt{\sum_{|\alpha| \le 1} \int_\Omega |\partial^\alpha (u - \tilde{u})|^2 \, dx} \end{aligned}$$

give the errors with respect to the associated norms of the underlying Sobolev space in which the weak solution is derived. Thereby $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ represents a partial differentiation multi-index with $|\alpha| = \sum_{i=1}^n \alpha_i$.

6.2 Planar meshes example

The first example considered is the planar involute gear model with 17 teeth and a radius of 9.5 units depicted in Fig. 6.2a. In order to demonstrate the sensitivity of the finite element discretization accuracy, the strongly varying analytic solution

$$u(x,y) := 40 \sin\left(\frac{1}{2}(x+1)(y-1)\right) + 60$$

has been chosen, which is depicted in Fig. 6.2b. Applying the Laplace operator to u and interchanging signs results in the right hand side f of the Poisson's equation used for the finite element computation.

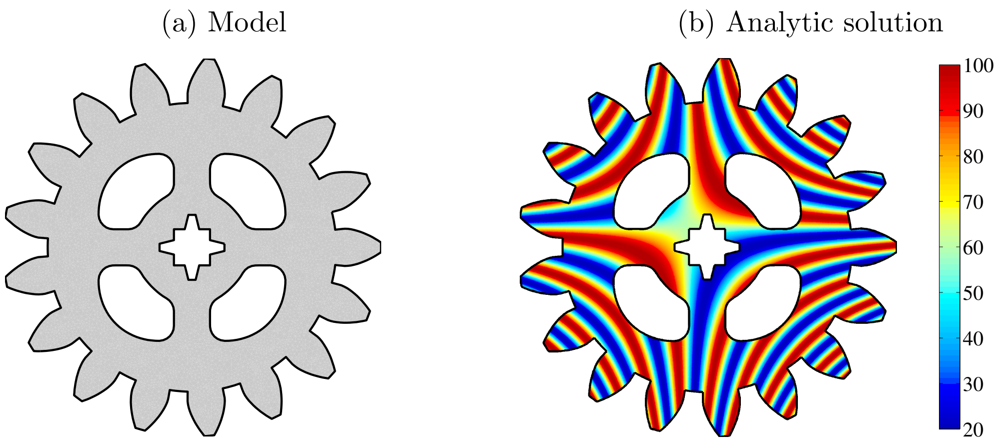


Figure 6.2: Involute gear problem: Model and model colored according to the analytic solution u of the Poisson problem

Two different initial meshes for the same domain have been generated. One is a triangular mesh with 14,346 elements and 7660 nodes, the other a quadrilateral mesh with 6229 elements and 6716 nodes. Here, the number of elements has been chosen such, to yield similar average element edge lengths in both cases. These meshes have been subsequently distorted by element validity preserving random node movements in order to increase their smoothing potential and to demonstrate the influence of low quality meshes on finite element solution accuracy.

In the case of GETMe smoothing, the polygonal element transformation scheme, as described in Section 2.3 and depicted in Fig. 2.9, has been applied. Here, the simultaneous substep transformation parameters have been set to $\lambda = 7/20$ and $\theta = \pi/12$. In the case of the sequential substep $\lambda = 9/20$ and $\theta = 5\pi/36$ have been applied. Since one iteration of GETMe sequential smoothing consists of transforming a single element, the comparatively expensive evaluation of mesh quality, which is used for termination control, was only assessed after a cycle of 100 consecutive single element transformations each. For comparative purposes, the meshes have also been improved by area-weighted Laplacian smoothing, smart Laplacian smoothing, and the global optimization-based approach provided by the shape improvement wrapper of Mesquite version 2.1.2.

Details of the initial, as well as the resulting smoothed meshes, are depicted in Fig. 6.3. Here, each mesh element is colored according to its individual mean ratio quality number. As can be seen by the quality color bar shown in the lower part of Fig. 6.3, low quality elements are marked by reddish colors, whereas high quality elements are marked by bluish colors.

The resulting smoothing runtime information is provided by Table 6.1. In the case of GETMe, the iteration numbers of the simultaneous and the sequential

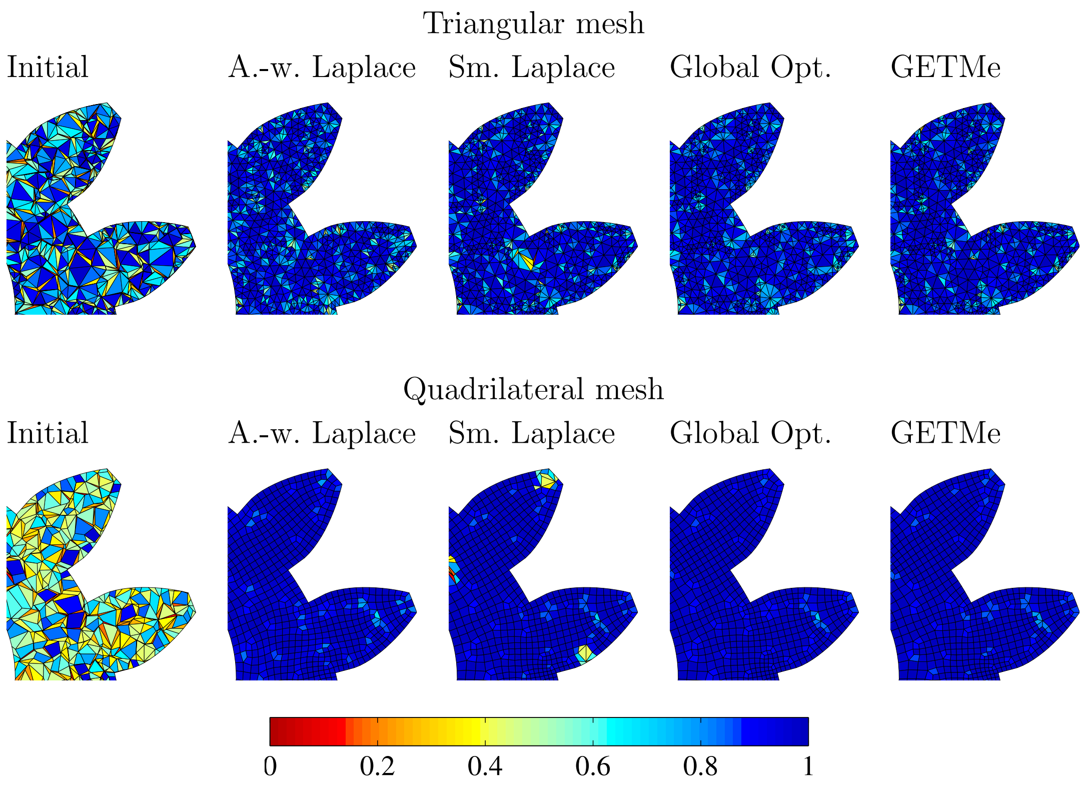
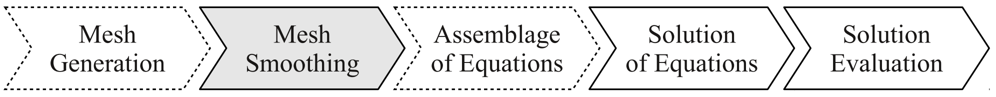


Figure 6.3: Involute Gear problem: Details of initial and smoothed meshes with elements colored according to their mean ratio quality number

substep as well as the total smoothing time are given. Table 6.1 also provides the quality numbers of the best mesh achieved during smoothing, which was used for the subsequent finite element computation. Since the best mesh is not necessarily generated by the last iteration, this requires to backup the node coordinates in case of quality improvements and reverting to these after smoothing has been terminated according to the termination criteria described before. Area-weighted Laplacian smoothing does not involve any quality computation. Therefore, results are given for the final mesh.

As can be seen from Table 6.1, all methods improved the low initial mean mesh quality number q_{mean} significantly. However, in contrast to area-weighted Laplacian smoothing, global optimization, and GETMe smoothing, the minimal element quality number q_{min} was marginally improved by smart Laplacian smoothing. As can be seen in Fig. 6.3, this is caused by clusters of distorted elements. Here, node movements have been repeatedly rejected by the smart variant in order to avoid the generation of invalid elements or a temporary decrease of local mesh quality, leaving such clusters unchanged. Since area-weighted Laplacian smoothing does not involve any of such quality constraints, all problematic regions could be improved although invalid elements occurred during the first three smoothing steps. In contrast, GETMe smoothing avoided at any time the generation of invalid elements and led to particularly good $q_{\min}$ values due to the incorporated sequential substep focusing explicitly on low quality elements.

Table 6.1: Involute gear problem: Performance of mesh smoothing techniques

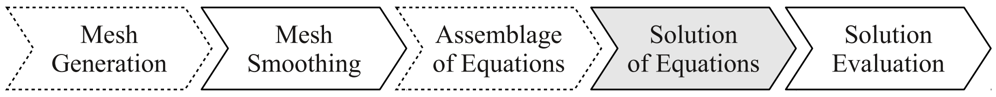


Mesh	Method	Iter	Time (s)	$q_{\min}$	q_{mean}
Tri	Initial	–	–	0.0000	0.3699
	Area-weighted Laplace	24	0.13	0.2045	0.8749
	Smart Laplace	18	0.20	0.0000	0.8790
	Global Optimization	194	2.27	0.2901	0.8941
	GETMe	40/2800	0.31	0.3367	0.8896
Quad	Initial	–	–	0.0004	0.4180
	Area-weighted Laplace	39	0.10	0.6484	0.9625
	Smart Laplace	27	0.16	0.0050	0.9394
	Global Optimization	137	1.83	0.6624	0.9718
	GETMe	33/7200	0.13	0.7907	0.9698

The iteration numbers of the global optimization-based approach are significantly larger than those required by both Laplacian smoothing variants and GETMe simultaneous smoothing. In combination with the increased time required per iteration this led to considerable longer overall smoothing runtimes for the global optimization approach. For example, in the case of the triangular and quadrilateral mesh, global optimization takes 7.3 and 14.1 times longer compared to the total GETMe smoothing time. It can also be seen that in both cases area-weighted Laplacian smoothing is faster than smart Laplacian smoothing despite the fact that the iteration numbers are increased. This is due to the fact that area-weighted Laplacian smoothing does not involve costly evaluations of the mean ratio quality criterion.

The finite element solution $\tilde{u}$, approximating the analytic solution u, as well as the associated error norms have been computed by a solver being based on the GetFEM++ library version 4.1.1 [175]. Here, the first step consisted of assembling the stiffness matrix A and the right hand side b of the system of linear equations $Ac = b$. Subsequently, this system was solved by the conjugated gradient (CG) solver using the termination tolerance 10^{-8} in order to determine the coefficient vector c of the trial functions modeling the approximate solution

Table 6.2: Involute gear problem: Performance of CG solver



Mesh	Basis	Method	cond(A)	Iter	Time (s)
Tri	$d = 1$	Initial	1.09e+08	10,618	1.78
		Area-weighted Laplace	7.39e+02	152	0.02
		Smart Laplace	4.28e+06	1204	0.20
		Global Optimization	2.88e+02	140	0.02
		GETMe	4.26e+02	160	0.03
	$d = 2$	Initial	1.50e+09	43,128	71.04
		Area-weighted Laplace	7.22e+03	435	0.72
		Smart Laplace	3.45e+07	8421	13.83
		Global Optimization	2.99e+03	367	0.61
		GETMe	3.05e+03	426	0.70
Quad	$d = 1$	Initial	1.64e+05	1443	0.24
		Area-weighted Laplace	3.07e+02	99	0.02
		Smart Laplace	1.33e+04	393	0.06
		Global Optimization	1.34e+02	94	0.01
		GETMe	1.35e+02	94	0.02
	$d = 2$	Initial	2.74e+06	5589	8.87
		Area-weighted Laplace	2.67e+03	278	0.43
		Smart Laplace	2.19e+05	1580	2.51
		Global Optimization	1.14e+03	263	0.42
		GETMe	1.16e+03	263	0.42

$\tilde{u}$.

As is well known, mesh quality affects the condition number cond(A) of the stiffness matrix and hence the computational effort in the case of iterative solvers. This is reflected by the results given in Table 6.2, where the 2-norm condition numbers as well as the iteration numbers and solution times are given.

The finite element analysis has been accomplished for piecewise linear ($d = 1$) as well as piecewise quadratic ($d = 2$) Lagrangian elements. In doing so, the resulting number of degrees of freedom for the triangular and quadrilateral mesh became 7660 and 6716, respectively for the linear case. For the quadratic basis, using six node triangular Lagrangian elements and nine node quadrilateral Lagrangian elements the corresponding number of degrees of freedom increased to 29,670 and 25,894, respectively. As described in Section 6.1, Dirichlet type boundary conditions have been applied, i.e. the analytic solution $u(p_i)$ has been prescribed for all boundary nodes $p_i \in I_{\partial\Omega}$. Since these have been kept fixed during smoothing, boundary conditions are identical for all meshes.

As can be seen in Table 6.2, large condition numbers occur for the very low quality initial meshes resulting in significantly increased solution times. Applying area-weighed Laplacian smoothing, global optimization and GETMe smoothing led to a reduction of these condition numbers by six and three orders of magnitude, if compared to the initial triangular and quadrilateral meshes. In consequence, solving the system of linear equations was sped up at least by the factors 59.3 and 12.0, respectively. Compared to this, the results of smart Laplacian smoothing are inferior.

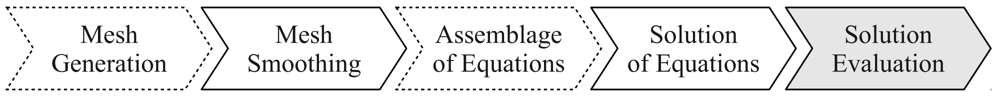
In practice, CG iteration numbers can be further decreased by applying preconditioning techniques. For example, using diagonal preconditioning in the case of the quadrilateral elements and $d = 2$, the number of CG iterations for the initial mesh, area-weighted Laplace, smart Laplace, global optimization and GETMe are reduced to 1293, 234, 424, 223, and 226, respectively. As can be seen by comparing these numbers with those given in Table 6.2, CG benefits most from preconditioning in the case of the problematic initial and smart Laplacian smoothing mesh since condition numbers are particularly large in these cases. However, the number of iterations are still inferior if compared to the other smoothing methods.

The favorable runtime behavior of GETMe smoothing also becomes apparent, if smoothing and CG solution time are jointly considered. For example, in the case of the quadrilateral mesh and piecewise quadratic basis functions, GETMe smoothing results in a speedup factor of 4.9 and 4.1 compared to smart Laplace and global optimization, respectively. However, by avoiding expensive mean ratio quality evaluations and not addressing mesh validity, the area-weighted Laplacian smoothing is about four percent faster than GETMe smoothing.

Table 6.3 gives results for the finite element error numbers as defined in Section 6.1. In accordance to the trend of the condition numbers, low quality initial meshes led to large solution errors with respect to all criteria. They were significantly reduced by all smoothing methods, as is particularly visible in the case of quadratic basis functions i.e. $d = 2$. For example, in the case of the triangular initial mesh the finite element solution error numbers $e_{\max}$, e_{mean}, e_{L^2}, and e_{H^1} are 16.4, 13.3, 12.7, and 22.7 times larger compared to those obtained for the GETMe mesh. For the quadrilateral mesh, the corresponding factors become 29.0, 26.3, 27.3, and 65.0, respectively.

The results in Table 6.3 also clearly show the benefit of higher order basis functions. For example, in the case of GETMe smoothing the error numbers $e_{\max}$ and e_{L^2} of linear and quadratic basis functions differ by the factors 11.0 and 41.6 for the triangular mesh and by the factors 53.8 and 125.5 for the quadrilateral mesh. However, this benefit is accompanied by an increased computational effort

Table 6.3: Involute gear problem: Finite element solution errors



Mesh	Basis	Method	$e_{\max}$	e_{mean}	e_{L^2}	e_{H^1}
Tri	$d = 1$	Initial	31.2934	3.0871	54.0214	1097.9148
		Area-weighted Laplace	5.0354	0.7406	11.0910	84.0124
		Smart Laplace	9.4938	0.8919	15.0708	101.5688
		Global Optimization	7.5949	0.8139	13.0744	83.2316
		GETMe	6.0528	0.8129	13.1754	79.1765
	$d = 2$	Initial	9.0137	0.1749	4.0055	179.9044
		Area-weighted Laplace	0.5407	0.0118	0.2437	8.0906
		Smart Laplace	0.9056	0.0163	0.4125	14.1967
		Global Optimization	0.6512	0.0146	0.3661	9.1231
		GETMe	0.5487	0.0132	0.3166	7.9157
Quad	$d = 1$	Initial	19.3538	2.3280	38.5825	309.5934
		Area-weighted Laplace	4.1516	0.5539	8.5329	43.4628
		Smart Laplace	9.7240	0.6560	11.4813	81.8776
		Global Optimization	4.5228	0.5795	9.4838	43.6133
		GETMe	4.3557	0.5788	9.5786	43.3649
	$d = 2$	Initial	2.3459	0.1025	2.0842	85.1985
		Area-weighted Laplace	0.1043	0.0040	0.0748	1.5818
		Smart Laplace	2.2336	0.0128	0.6341	17.0958
		Global Optimization	0.0933	0.0040	0.0773	1.3811
		GETMe	0.0810	0.0039	0.0763	1.3109

and higher memory requirements for generating and solving the system of linear equations.

The numerical tests given in the previous Chapter provided evidence that GETMe smoothing leads to significant improvements of mesh quality within the first few iterations, which is particularly advantageous in industrial applications. To substantiate this, and to provide a deeper insight into the effect of mesh smoothing on finite element solution accuracy, the solution errors will be subsequently analyzed with respect to smoothing time. For this purpose, a complete finite element analysis was performed for the resulting mesh of each smoothing iteration. This gives a better insight into the connection between single smoothing iterations and finite element solution errors, and hence allows a more detailed analysis with respect to smoothing runtime.

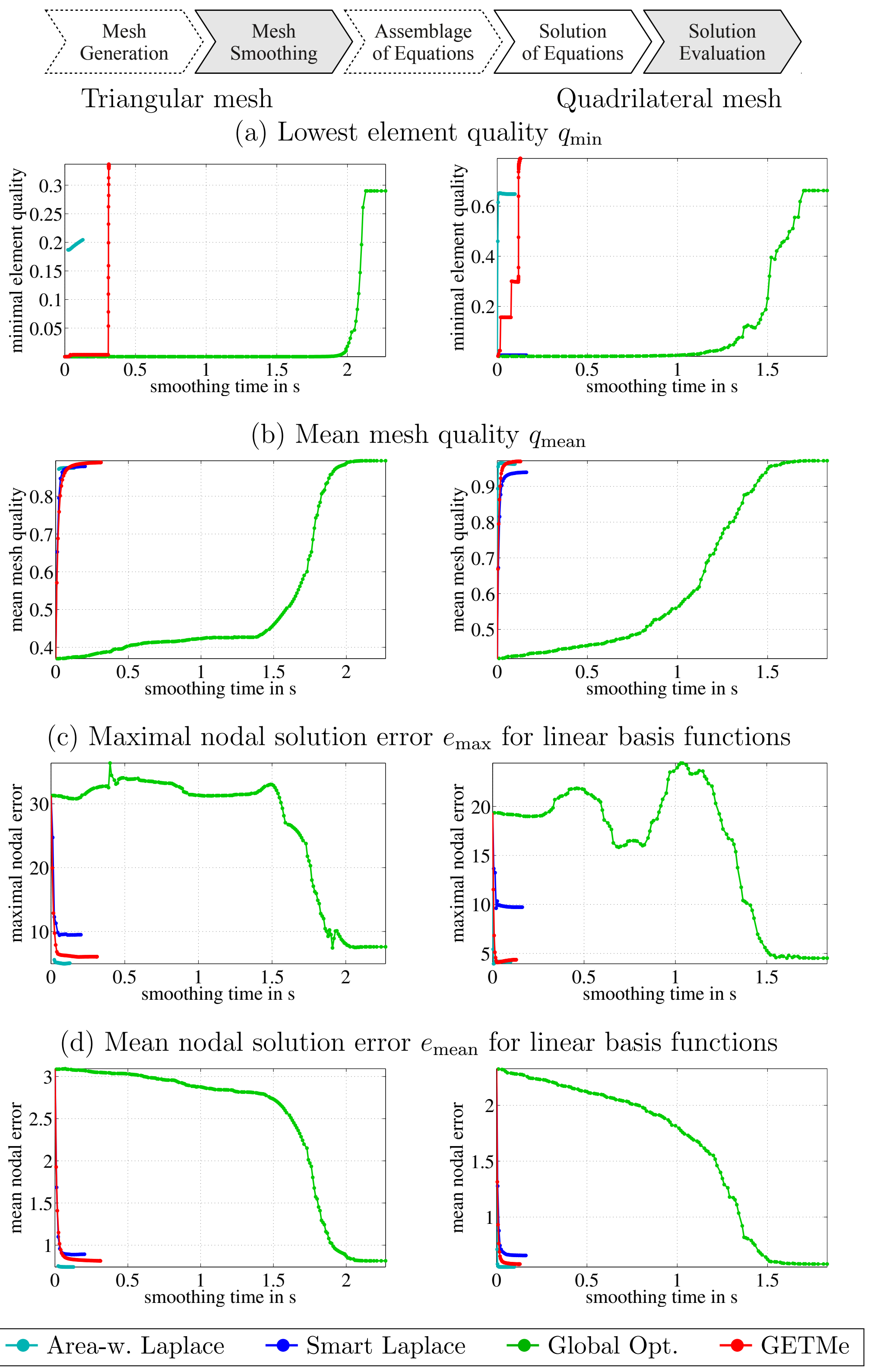


Figure 6.4: Involute gear problem: Mesh quality with respect to smoothing time and corresponding finite element discretization errors for the triangular (left) and quadrilateral meshes (right)

Results for linear basis functions are depicted in Fig. 6.4. Here, Fig. 6.4a shows the minimal element quality $q_{\min}$ with respect to smoothing time for the triangular mesh (left) and the quadrilateral mesh (right). In the given graphs each point indicates the results after one smoothing iteration or smoothing cycle in the case of GETMe sequential. Quality improvements of the latter are particularly visible by the sharp rise of $q_{\min}$ starting after 0.31s for the triangular mesh and after 0.12s for the quadrilateral mesh. In contrast, the graph of smart Laplacian smoothing is hardly visible in both cases, since $q_{\min}$ improvements are marginal. In the case of the triangular mesh and area-weighted Laplacian smoothing, meshes obtained by the first three iterations are invalid. Hence the finite element solution could not be computed leading to an according gap in the runtime graphs depicted on the left side of Fig. 6.4.

Similar to both Laplacian smoothing variants, GETMe simultaneous led to a sharp rise of the mean mesh quality q_{mean} within the first few steps as is shown in Fig. 6.4b. In contrast, improvements by the global optimization-based approach were rather moderate during the first half of its iterations.

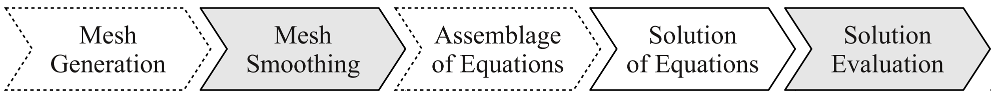
Although mesh quality gives an indication, making a priori statements on the resulting finite element solution quality is not fully reliable. This holds particularly for the worst element quality and maximal nodal errors. This is shown in the lower part of Fig. 6.4 by the finite element solution error diagrams with respect to smoothing time. In practice, such information is not available since the finite element analysis is accomplished only once, after mesh smoothing has been completed and not after each smoothing iteration or cycle.

As can be seen in Fig. 6.4c, depicting the maximal nodal error $e_{\max}$ with respect to smoothing time, the connection to $q_{\min}$ is less obvious. Although smart Laplacian smoothing failed to improve $q_{\min}$ significantly, $e_{\max}$ was reduced up to a certain extent. However, in the case of the quadrilateral mesh particularly, the results are inferior if compared to those of global optimization and GETMe. Area-weighted Laplacian smoothing resulted in the lowest finite element errors in various cases although the achieved $q_{\min}$ values do not reach those of global optimization and GETMe smoothing. In the case of the global optimization-based approach, $e_{\max}$ was increased up to 26.2% during smoothing the quadrilateral mesh compared to the initial error. Therefore, a preliminary termination of the global optimization approach is not sensible due to its runtime behavior.

Due to the incorporation of all nodes and elements, the mean nodal error e_{mean} shown in Fig. 6.4d is more closely related to the mean mesh quality q_{mean}. Here, the strong reduction of the finite element error reflects the sharp rise of mesh quality in the case of the Laplace variants and GETMe-based smoothing. In the case of global optimization, significant improvements were mainly achieved within the second half of its runtime.

As can be seen by these results, an effective reduction of finite element solu-

Table 6.4: Involute gear problem: Mesh smoothing time required to reach prescribed finite element error bounds in the subsequent finite element approximation



Mesh	Basis	Error	Threshold	Global Opt.	GETMe	Speedup
Tri	$d = 1$	e_{max}	7.8015	1.91 s	0.04 s	49.0
		e_{mean}	0.8545	2.01 s	0.08 s	26.1
		e_{L^2}	13.7279	2.03 s	0.11 s	17.7
		e_{H^1}	87.3932	2.01 s	0.08 s	26.1
	$d = 2$	e_{max}	0.6759	2.10 s	0.11 s	18.3
		e_{mean}	0.0152	2.01 s	0.07 s	29.1
		e_{L^2}	0.3807	2.06 s	0.08 s	26.8
		e_{H^1}	9.5254	2.03 s	0.11 s	18.8
Quad	$d = 1$	e_{max}	4.7485	1.55 s	0.02 s	96.9
		e_{mean}	0.6083	1.51 s	0.03 s	52.1
		e_{L^2}	9.9550	1.52 s	0.04 s	35.4
		e_{H^1}	45.7818	1.57 s	0.03 s	49.1
	$d = 2$	e_{max}	0.0935	1.62 s	0.12 s	13.3
		e_{mean}	0.0042	1.62 s	0.05 s	31.8
		e_{L^2}	0.0810	1.62 s	0.05 s	31.8
		e_{H^1}	1.4486	1.65 s	0.08 s	20.9

tion errors is achieved particularly during the first few GETMe iterations. Hence, in practice, runtime benefits of GETMe smoothing are even better than those indicated by comparing the total smoothing times given in Table 6.1. To demonstrate this, Table 6.4 gives the runtimes required to achieve prescribed finite element error thresholds, which are given in the fourth column. These thresholds represent the minimal error numbers achieved during global optimization based smoothing increased by an arbitrarily chosen quantity of 5%. The fifth and sixth columns give the smoothing runtimes required by global optimization and GETMe smoothing in order to reach these thresholds in the subsequent finite element computations. The last column contains the speedup factors obtained by dividing the runtimes of global optimization by the runtimes of GETMe. This test reflects the fact that achieving minimal errors has to be payed by a disproportional computational effort. Therefore, from a practical point of view, looking for almost optimal results is more advantageous since these are accomplished with significantly shorter runtimes.

Error bounds have been set on the basis of the global optimization-based approach. Due to these demanding bounds, smart Laplacian smoothing failed to achieve the prescribed bounds in all of the 16 cases. Thus, results of smart Laplacian smoothing are inferior and hence omitted in Table 6.4. Area-weighted Laplacian smoothing has also been omitted, since it is not guaranteed to result in valid meshes and thus is not generally applicable. In contrast, GETMe smoothing achieves the prescribed thresholds for all meshes, basis functions, and error types within short runtimes. This provides evidence for the practicability and effectiveness of this element transformation-based smoothing approach. Furthermore, comparing the resulting GETMe runtimes of Table 6.4 with the total smoothing runtimes of 0.31s and 0.13s for the triangular and quadrilateral mesh, suggests that in practice weaker smoothing termination criteria may be used.

6.3 Volumetric meshes example

Due to the increased approximation power of the associated finite element basis functions, hexahedral elements are known to result in more accurate finite element solutions with a lower number of elements compared to tetrahedral meshes. However, depending on the geometrical complexity of the model, the generation of quality hexahedral meshes is usually far more complex than the generation of quality tetrahedral meshes. Thus, depending on the application, the usage of hexahedral elements only pays off in case of reasonably meshable geometries.

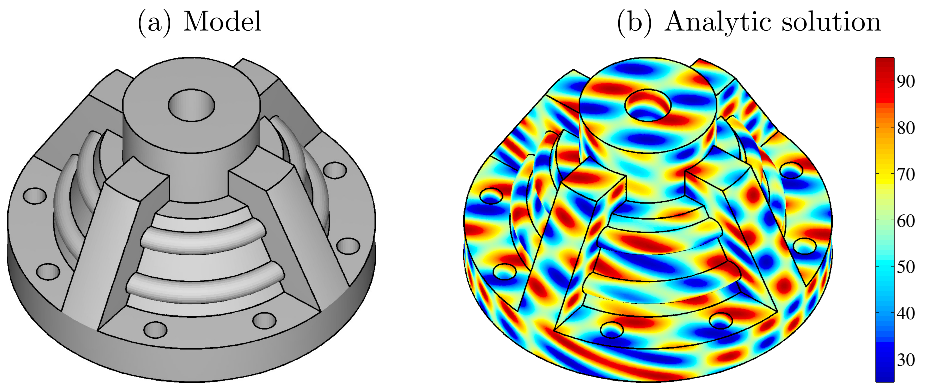


Figure 6.5: Volumetric problem: Model and model colored with respect to the analytic solution

The model depicted in Fig. 6.5a is meshable by a sweeping approach, i.e. by decomposing it into simple sub-parts, which are meshed by sweeping a quadrilateral mesh from a source surface onto an adequate target surface [83]. This example will be used to demonstrate the influence of mesh quality, different element types and the polynomial degree of the basis functions on finite element solution accuracy. Fig. 6.5b depicts the model colored with respect to the analytic solution, which has been set to

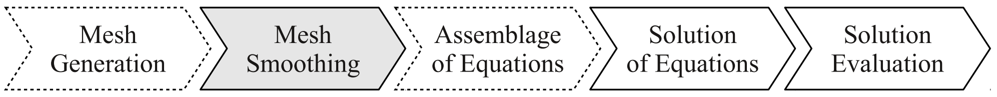
$$u(x, y, z) := 35 \sin(2x + y) \cos(y - 3z) + 60 \, .$$

The bounding box of the model is given by $(x, y, z) \in [-8, 8] \times [-8, 8] \times [0, 9]$.

The hexahedral mesh consists of 74,730 nodes and 67,055 elements. For comparison reasons, a tetrahedral mesh of the same model consisting of 103,230 nodes and 566,004 elements is also considered. Since the volume of a regular hexahedron is $6\sqrt{2} \approx 8.5$ times larger than the volume of a regular tetrahedron with the same edge length, the number of tetrahedral elements has been increased accordingly. Again, both initial meshes have been distorted by element validity preserving random node movements. Cross sections of the resulting meshes as well as their smoothed counterparts are depicted in Fig. 6.6.

In the case of GETMe smoothing, the default parameters have been used. In addition, the element transformation specific parameter of GETMe simultaneous has been set to the element quality dependant choice $\sigma = 0.7 + 50(1 - q(E))$ for both mesh types. In the case of the GETMe sequential substep, the fixed choice $\sigma = 17$ was applied in both cases.

Table 6.5: Volumetric problem: Performance of mesh smoothing techniques



Mesh	Method	Iter	Time (s)	$q_{\min}$	q_{mean}
Tet	Initial	–	–	0.0001	0.4178
	Volume-weighted Laplace	20	10.49	invalid	invalid
	Smart Laplace	23	15.99	0.0018	0.8748
	Global Optimization	74	51.58	0.4935	0.8887
	GETMe	26/45000	11.24	0.6591	0.8871
Hex	Initial	–	–	0.0606	0.4631
	Volume-weighted Laplace	42	12.42	0.5613	0.9322
	Smart Laplace	28	17.71	0.0690	0.8777
	Global Optimization	110	53.97	0.6118	0.9410
	GETMe	30/600	4.15	0.6125	0.9363

Quality numbers, as well as runtime information for all meshes and smoothing methods, are summarized in Table 6.5. As can be seen, GETMe is particularly

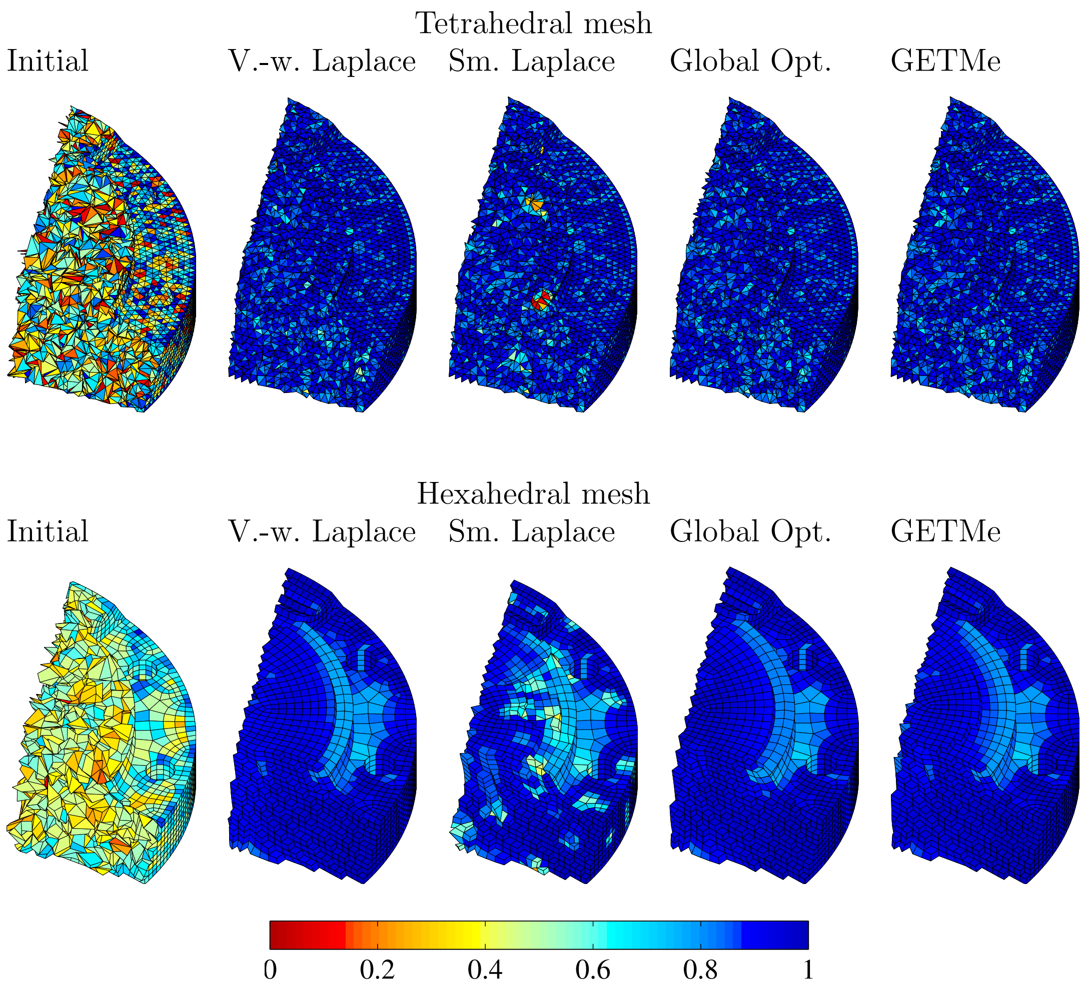


Figure 6.6: Volumetric problem: Cross sections of initial and smoothed meshes with elements colored according to their mean ratio quality number

fast in the case of the hexahedral mesh, for which smoothing is accomplished 3.0, 4.3, and 13.0 times faster compared to volume-weighted Laplacian smoothing, smart Laplacian smoothing, and global optimization, respectively. The volume-weighted Laplacian smoothing is relatively slow in this test, since it requires a comparably expensive computation of the element volume, for each node and each attached element, per iteration. The effort of the smart Laplacian smoothing approach is even higher, since for each node and each attached element the quality has to be computed twice per iteration. In contrast, GETMe smoothing only requires one quality evaluation per element per iteration step. Again, the number of iterations required by the global optimization-based approach is comparably high.

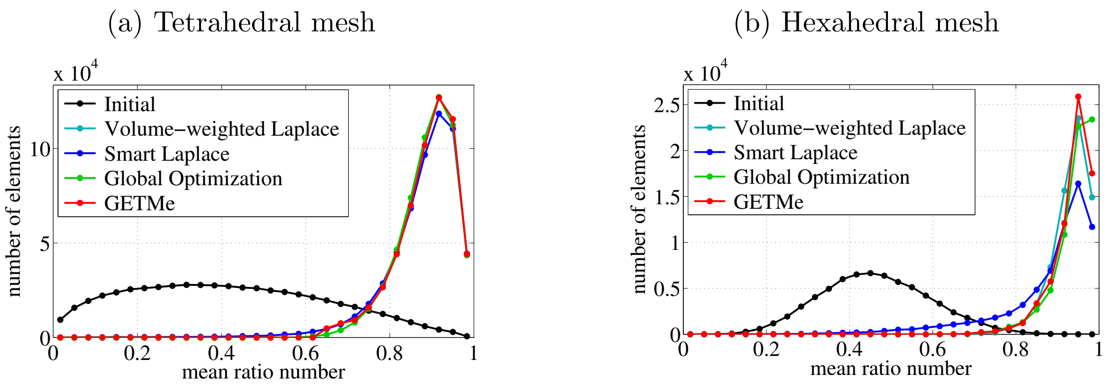


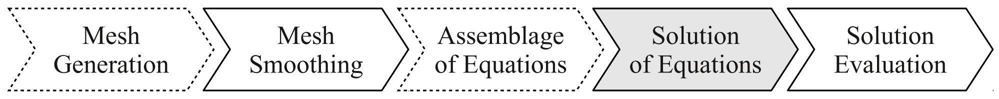
Figure 6.7: Volumetric problem: Mesh element quality histograms

In the case of the tetrahedral mesh, volume-weighted Laplacian smoothing led in none of its 20 iterations to a valid mesh. Consequently, finite element solutions could not be computed. Similar to the case of the planar meshes, smart Laplacian smoothing failed to improve the lowest element quality considerably, as indicated in Table 6.5. For example, the tetrahedral mesh smoothed by smart Laplacian smoothing still contains 2365 elements with a mean ratio number below 0.3. In contrast, the tetrahedral meshes improved by global optimization and GETMe smoothing do not contain any elements with quality numbers below 0.4935 and 0.6591, respectively. However, the associated element quality histograms of all successfully smoothed meshes depicted in Fig. 6.7a differ only slightly. As a consequence, the resulting mean mesh quality numbers q_{mean} are comparable as can be seen in Table 6.5.

In the case of the hexahedral elements mesh quality differences are also obvious with respect to q_{mean} due to a lower number of high quality elements. For example, the number of mesh elements with a mean ratio quality number larger than 0.8 amounts to 0.6%, 98.4%, 82.3%, 97.9%, and 98.1% in the case of the initial mesh and those smoothed by volume-weighted Laplace, smart Laplace, global optimization and GETMe, respectively.

In the case of the valid tetrahedral meshes, the finite element analysis has been accomplished with 4 and 10 node Lagrangian elements resulting in 103,230, and 791,449 degrees of freedom in the associated systems of linear equations. In the case of the hexahedral mesh, the linear and quadratic basis have been generated using 8 and 27 node Lagrangian elements resulting in 74,730 and 566,968 degrees of freedom. Dirichlet type boundary conditions have been applied for all 18,961 linear basis and 77,043 quadratic basis tetrahedral mesh boundary nodes and 15,154 linear basis and 60,664 quadratic basis hexahedral mesh boundary nodes, respectively. Table 6.6 summarizes the stiffness matrix condition numbers and the results of the CG solver.

Table 6.6: Volumetric problem: Performance of CG solver

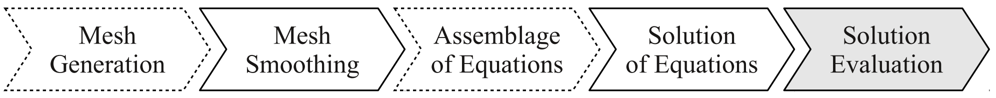


Mesh	Basis	Method	cond(A)	Iter	Time (s)
Tet	$d = 1$	Initial	2.34e+07	7530	66.06
		Volume-weighted Laplace	–	–	–
		Smart Laplace	1.30e+05	1267	11.13
		Global Optimization	3.66e+02	139	1.23
		GETMe	3.57e+02	139	1.23
	$d = 2$	Initial	4.90e+08	42,301	4946.16
		Volume-weighted Laplace	–	–	–
		Smart Laplace	2.12e+06	6020	705.22
		Global Optimization	2.25e+03	323	37.89
		GETMe	2.20e+03	326	38.25
Hex	$d = 1$	Initial	1.51e+04	507	3.61
		Volume-weighted Laplace	2.89e+02	124	0.89
		Smart Laplace	1.94e+03	227	1.62
		Global Optimization	4.45e+02	147	1.06
		GETMe	3.67e+02	140	1.01
	$d = 2$	Initial	7.35e+05	3809	464.02
		Volume-weighted Laplace	2.85e+03	361	43.96
		Smart Laplace	2.16e+05	1188	144.72
		Global Optimization	4.54e+03	441	53.82
		GETMe	3.60e+03	405	49.46

The impact of mesh smoothing on the convergence of the iterative solution process of the finite element equations is evident in the case of tetrahedral meshes. Although element quality histograms did not differ much, low quality elements in the mesh obtained by smart Laplacian smoothing resulted in an increased condition number and subsequently in a larger number of CG iterations. In consequence, solving the system of linear equations for the linear and quadratic basis associated to the tetrahedral smart Laplace mesh took 9.0 and 18.4 times longer compared to those associated to the results of global optimization or GETMe smoothing. Therefore, in terms of the overall simulation time and resulting mesh quality, smart Laplacian smoothing is not competitive. Volume-weighted Laplacian smoothing did not lead to any results at all. In contrast, since GETMe smoothing resulted in a similar CG solution time as global optimization, smoothing runtime advantages directly pay off.

Due to the sweeping approach, the topological structure of the hexahedral mesh is almost regular. To be precise, 92.8% of all inner nodes are shared by exactly eight hexahedral elements. This is also reflected by the resulting high quality meshes and the comparably low condition number of the associated stiffness matrices for volume-weighted Laplacian smoothing, global optimization and GETMe smoothing.

Table 6.7: Volumetric problem: Finite element solution errors



Mesh	Basis	Method	$e_{\max}$	e_{mean}	e_{L^2}	e_{H^1}
Tet	$d = 1$	Initial	26.5052	3.7173	129.4961	2332.6902
		Vol.-weight. Laplace	–	–	–	–
		Smart Laplace	14.2714	0.8742	31.3861	270.9987
		Global Optimization	5.1902	0.8052	28.0895	150.7587
		GETMe	4.6681	0.7981	27.7852	147.7948
	$d = 2$	Initial	4.3067	0.2119	8.8008	554.4773
		Vol.-weight. Laplace	–	–	–	–
		Smart Laplace	3.3903	0.0195	1.0386	52.3217
		Global Optimization	0.3456	0.0145	0.5396	13.7242
		GETMe	0.3050	0.0142	0.5216	13.1961
Hex	$d = 1$	Initial	16.1917	2.0114	66.1737	466.4109
		Vol.-weight. Laplace	2.1788	0.4495	14.8826	66.3884
		Smart Laplace	8.9167	0.6372	23.2996	126.0352
		Global Optimization	2.6647	0.4512	15.6927	68.1431
		GETMe	2.2435	0.4479	15.1690	66.7886
	$d = 2$	Initial	3.7028	0.1058	4.8893	105.2790
		Vol.-weight. Laplace	0.2760	0.0043	0.2227	2.9266
		Smart Laplace	2.4930	0.0133	0.9654	18.6139
		Global Optimization	0.1997	0.0045	0.2827	3.2438
		GETMe	0.2203	0.0042	0.2369	2.9515

By comparing the finite element solution errors provided in Table 6.7, one can observe that for the Poisson problem under consideration hexahedral elements lead to particularly good results. Similarly, finite element solution accuracy is improved significantly by the higher order basis functions. For example, in the case of the tetrahedral mesh smoothed by GETMe, quadratic basis functions decrease $e_{\max}$, e_{mean}, e_{L^2}, and e_{H^1} by the factors 15.3, 56.2, 53.3, and 11.2, if compared to linear basis functions. The corresponding improvement factors for the hexahedral mesh are 10.2, 106.6, 64.0, and 22.6, respectively.

As in the case of the planar meshes, the reduction of finite element solution errors obtained by smart Laplacian smoothing is inferior compared to the results of global optimization and GETMe smoothing. This is particularly visible for quadratic basis function, where the maximal nodal errors differ by the factor 11.1 and 11.3 for the tetrahedral and hexahedral mesh. Similarly, large errors occur in the first derivatives as can bee seen by the results for the H^1-norm. Here, the error numbers of smart Laplacian smoothing are 4.0 and 6.3 times larger, if compared to the results of GETMe smoothing.

In the case of the hexahedral mesh, results of volume-weighted Laplacian smoothing are comparable to those of GETMe smoothing. However, to guarantee the preservation of mesh validity in any case, this approach has to be modified in order to incorporate mesh quality aspects. This will further increase smoothing time. Furthermore, simple adjustments may not suffice, as can be seen on the example of the smart Laplacian smoothing approach. This is explained by the simple node movement techniques of Laplacian smoothing, which do not incorporate dedicated quality improvement mechanisms like quality based optimization or quality improving regularizing element transformation as used by global optimization or GETMe smoothing.

As can be seen in Table 6.7, GETMe smoothing resulted in slightly lower finite element approximation errors in 15 of 16 cases if compared to global optimization. This is also an indication that the substantial runtime advantages obtained by adapted GETMe parameters, at the expense of a slight decrease of the mean mesh quality, do not necessarily have to result in increased finite element approximation errors.

As in the case of the first example, the finite element analysis has been performed for each iteration or cycle of the mesh smoothing process to provide a deeper insight into the effect of mesh smoothing on finite element solution accuracy. Results with respect to the minimal element quality numbers $q_{\min}$ are depicted in Fig. 6.8a. For both the tetrahedral mesh, as well as the hexahedral mesh, the effect of the GETMe sequential substep is visible by the sharp rise of $q_{\min}$ after 9.98s and 4.13s, respectively. In the case of the hexahedral mesh, the GETMe simultaneous step already achieved an excellent minimal element quality number of 0.5818. With respect to the mean mesh quality q_{mean}, GETMe simultaneous smoothing led in both cases to remarkably fast and substantial improvements as is demonstrated in Fig. 6.8b.

The effect of mesh smoothing on the maximal nodal error for quadratic basis functions is shown in Fig. 6.8c. As can be seen, during GETMe simultaneous smoothing, $e_{\max}$ decreased significantly although $q_{\min}$ remained comparatively

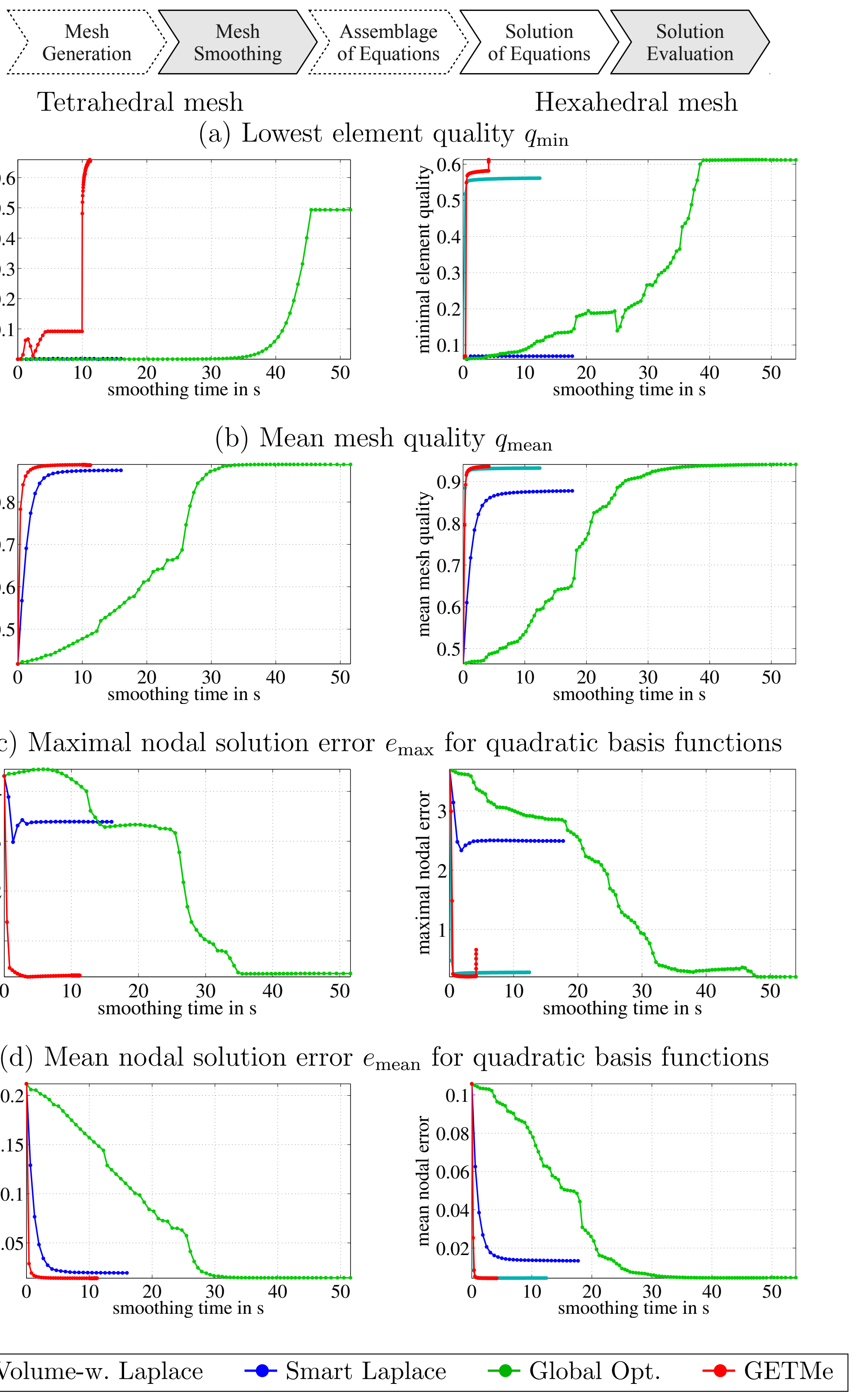
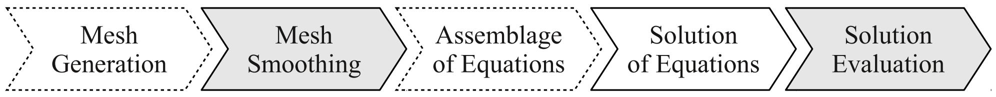


Figure 6.8: Volumetric problem: Mesh quality with respect to smoothing time and corresponding finite element discretization errors for tetrahedral (left) and hexahedral meshes (right)

Table 6.8: Volumetric problem: Smoothing time required to reach prescribed finite element error bounds



Mesh	Basis	Error	Threshold	Global Opt.	GETMe	Speedup
Tet	$d = 1$	e_{max}	5.0605	31.79 s	1.97 s	16.1
		e_{mean}	0.8454	29.84 s	1.58 s	18.9
		e_{L^2}	29.4432	30.56 s	1.97 s	15.5
		e_{H^1}	158.2460	31.79 s	1.97 s	16.1
	$d = 2$	e_{max}	0.3557	35.47 s	1.97 s	18.0
		e_{mean}	0.0153	31.79 s	1.97 s	16.1
		e_{L^2}	0.5649	32.40 s	1.97 s	16.4
		e_{H^1}	14.4060	32.40 s	1.97 s	16.4
Hex	$d = 1$	e_{max}	2.5883	42.45 s	0.67 s	63.4
		e_{mean}	0.4718	31.69 s	0.80 s	39.6
		e_{L^2}	16.2106	32.19 s	0.80 s	40.2
		e_{H^1}	70.9716	33.18 s	0.80 s	41.5
	$d = 2$	e_{max}	0.2097	47.99 s	1.73 s	27.7
		e_{mean}	0.0046	35.55 s	0.80 s	44.4
		e_{L^2}	0.2752	33.66 s	0.67 s	50.2
		e_{H^1}	3.2797	37.47 s	0.80 s	46.8

low in the case of the tetrahedral mesh. Furthermore, e_{max} increased during the GETMe sequential substep for the hexahedral mesh while q_{min} was further improved. Hence, largest q_{min} values do not guarantee lowest e_{max} values. However, according to Table 6.7 the maximal nodal error $e_{\mathrm{max}} = 0.2203$ obtained by GETMe smoothing is low, since the best q_{min} value was not obtained in the last cycle of GETMe sequential smoothing. For the tetrahedral as well as the hexahedral mesh, the reduction of e_{max} is insufficient in the case of smart Laplacian smoothing, or requires significantly longer time as can be observed in the case of global optimization. The same holds for the mean nodal errors depicted in Fig. 6.8d.

To confirm the favorable runtime behavior of GETMe smoothing also for the case of volumetric meshes, Table 6.8 gives in the fourth column the times required to reach prescribed quality thresholds. As in the case of the planar meshes, these thresholds represent the minimal error numbers achieved during global optimization based smoothing increased by an arbitrarily chosen quantity of 5% to reflect the ability of GETMe to achieve almost optimal results within short times. As in the case of the planar involute gear example, GETMe reaches all these thresholds significantly faster than the global optimization-based approach, whereas smart Laplacian smoothing does not reach these thresholds in all 16 cases. Furthermore, the results given in Table 6.8 indicate that the speedup factors for both tetrahedral and hexahedral meshes are remarkably high.

Conclusions

The geometric element transformation method has been introduced as a novel approach for effective and efficient finite element mesh smoothing. In doing so the stability, effectivity and accuracy of subsequent finite element computations are improved. In contrast to other smoothing algorithms, GETMe is based on geometric element transformations, which lead to more regular, thus better quality elements if applied iteratively. In the case of polygonal elements, such transformations can be based on classical geometric constructions using similar triangles as was shown in Chapter 2.

In this thesis triangular elements were transformed by erecting isosceles triangles on each of the sides of the initial triangle as described in Section 2.1. The apices of the erected triangles are the nodes of the resulting triangle. Repeating this transformation leads to a sequence of triangles. By means of analysis it was shown that the shapes of these triangles converge to a characteristic shape that does not depend on the choice of the initial triangle but on the shape of the erected isosceles triangles. It was shown explicitly that this characteristic shape is approximated in every transformation step. Moreover, there is an upper bound for the speed of convergence.

Using the same transformation scheme in the case of polygons with an arbitrary number of nodes leads to a circulant matrix representation, which was analyzed by means of linear algebra in Section 2.2. This was done with respect to the base angle of the isosceles triangles erected on the sides of the polygon. All characteristic base angles leading to a change in the geometry of the limit polygon obtained by iteratively applying the same transformation have been determined. Additionally, it has been shown that these limit polygons are linear combinations of eigenpolygons of the initial polygon. Since all transformation schemes, which can be represented by circulant matrices, lead to the same eigenpolygons, the results obtained are not only applicable to the analyzed transformation, but to a broad variety of geometric polygon transformations. Furthermore, the classic theorems of Napoleon as well as of Petr-Douglas-Neumann were naturally deduced as special cases in the choice of parameters and number of transformation steps. In addition, it followed that these results are unique with respect to their construction and the choice of characteristic parameters.

In Section 2.3 a combined transformation for polygonal elements has been proposed and analyzed. It eliminates the rotational effect of the original similar triangles based transformation, which is adverse in the context of finite element mesh smoothing. This transformation combines two steps of the same transformation scheme using flipped similar triangles in the second substep, resulting in a circulant Hermitian transformation matrix with positive eigenvalues. Furthermore, by introducing an additional parameter, in order to define the similar triangles, this transformation is more general. Nevertheless, all possible resulting limit polygons have been derived with respect to the underlying parameter domain. This facilitates a directed choice of transformation parameters in the mesh smoothing context.

In preparation of the smoothing of volumetric finite element meshes, regularizing transformations for volumetric elements were considered in Chapter 3. First, the mean ratio quality criterion was introduced in Section 3.1 as a measure for element regularity. It is applicable not only to the most relevant volumetric elements, but also to polygonal elements thus providing a proper basis for mesh quality evaluation and smoothing control.

Regularizing transformations for tetrahedral, hexahedral, pyramidal and prismatic finite elements were introduced in the Sections 3.2 and 3.3. Whereas the efficient opposite normals based transformation is only applicable to tetrahedral elements due to their specific topological configuration, the more general dual element transformation is suitable for all element types under consideration. Numerical tests have shown that both transformations reliably and efficiently regularize even invalid elements of all types within a low number of steps. Additional mechanisms, like a normal scaling factor and relaxation, allow the speed of regularization to be controlled, which is reasonable with respect to mesh smoothing. In addition, basic properties of the transformation regarding the preservation of the centroid as well as the invariance with respect to scaling, rotation, and translation have been discussed.

Two different approaches of mesh smoothing based on geometric element transformations and a combination of them were presented in Chapter 4. The first is the simultaneous GETMe approach combining two driving forces. One is the regularizing element transformation, the other is a Laplacian smoothing like node averaging scheme. However, unlike Laplacian smoothing, which is based on computing arithmetic means of neighboring nodes, GETMe simultaneous smoothing is based on weighted means of temporary nodes obtained by transforming adjacent elements. Hence, quality improvement is mainly caused by the element quality controlled regularizing transformation and the quality weighted node averaging scheme. The second approach, named GETMe sequential, is based on successively improving the lowest quality element of the mesh by applying the transformation directly. Whereas the strong point of the simultaneous approach is to improve the overall mesh quality, the sequential approach efficiently improves the minimal element quality. Therefore, GETMe smoothing, as introduced in Section 4.4, consists of applying both methods consecutively, leading to high quality results with respect to both quality numbers.

An advanced version of the geometric element transformation method for finite element mesh smoothing was introduced in Section 4.5. In contrast to the standard GETMe approach, GETMe adaptive involves concepts of adaptivity, implementing a quality controlled two-stage smoothing technique integrated into one main smoothing loop, an adaptive node relaxation scheme, in order to avoid the generation of invalid mesh elements and to accelerate the rate of mesh improvement, and a quality ratio-based weighting scheme for updating the nodes, which is now consistently applied in both smoothing stages. Furthermore, by the usage of fixed transformation parameters and by eliminating the element quality penalty parameters, GETMe adaptive further reduces the number of parameters if compared to the standard GETMe approach.

In addition, an improved code implementation and parallelization of GETMe adaptive was discussed in Section 4.6. This results in a finite element mesh smoothing method which has both, a comparably low memory profile and short smoothing iteration runtimes. In combination with the powerful regularizing effect of the incorporated element transformations, this leads to an effective smoothing approach for finite element meshes of various types.

The effectivity and efficiency of the geometric element transformation method was demonstrated by the extensive collection of numerical results demonstrated in Chapter 5. In this Chapter results are given for a broad variety of mesh types covering planar polygonal meshes, polygonal surface meshes, all-tetrahedral, all-hexahedral and mixed volume meshes. The obtained results were compared to variants of smart Laplacian smoothing and a state of the art global optimization-based approach. It has been shown that GETMe smoothing leads to convincing results with respect to both quality and runtime. That is to say, it results in mesh qualities comparable to those obtained by the global optimization approach, while being significantly faster than the other methods. Furthermore, any Laplacian smoothing variant failed to improve the minimal element quality significantly and thus usually did fulfill the mesh quality requirements of finite element computations.

This was demonstrated in more detail in Chapter 6. Here the results of GETMe smoothing have not only been assessed with respect to mesh quality but also with respect to finite element solution accuracy for a number of Poisson problems with analytic solutions. Different meshes with regard to element types, mesh resolution, and the order of the associated finite element basis functions have been considered, and the beneficial effect of GETMe smoothing with respect to finite element system of equations solution efficiency and stability as well as finite element solution accuracy has been shown.

From an application point of view, additional potential for improvements of GETMe smoothing lies within the incorporation of adjusted element transformations, parameter sets and density function-based element scaling schemes, which might for example be required in order to smooth anisotropic meshes. Furthermore, it should be investigated, to which extent the incorporation of a priori error estimates or the combination with a posteriori error estimates can facilitate the smoothing process to achieve a better quality finite element approximation. However, since smoothing is limited by the initial mesh topology, a practice-oriented approach should also be combined with mesh topology modification techniques allowing local refinement or coarsening. From a theoretical point of view, additional work has to be done in order to provide a convergence proof for the volumetric element transformations. Although the transformation schemes are geometrically simple, involving cross products and normalization in a recursive approach leads to complex nonlinear expressions. The same holds for the pending proof of the mesh improving effect and convergence of the entire GETMe smoothing process.

References

[1] Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors. *Handbook of grid generation.* CRC Press, 1999.

[2] Pascal J. Frey and Paul-Louis George. *Mesh Generation.* Wiley-ISTE, 2nd edition, 2008.

[3] Vladimir D. Liseikin. *Grid Generation Methods.* Springer, 2nd edition, 2010.

[4] Steven J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of the 7th International Meshing Roundtable*, pages 239–267, 1998.

[5] A.O. Cifuentes and A. Kalbag. A performance study of tetrahedral and hexahedral elements in 3-d finite element structural analysis. *Finite Elements in Analysis and Design*, 12(3–4):313–318, 1992.

[6] Steven E. Benzley, Ernest Perry, Karl Merkley, Brett Clark, and Greg Sjaardama. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *Proceedings of the 4th International Meshing Roundtable*, pages 179–191, 1995.

[7] Dan Wake, Klas Lilja, and Victor Moroz. A hybrid mesh generation method for two and three dimensional simulation of semiconductor processes and devices. In *Proceedings of the 7th International Meshing Roundtable*, pages 159–166, 1998.

[8] Nicolas Flandrin, Houman Borouchaki, and Chakib Bennis. 3D hybrid mesh generation for reservoir simulation. *International Journal for Numerical Methods in Engineering*, 65(10):1639–1672, 2006.

[9] Yasushi Ito and Kazuhiro Nakahashi. Improvements in the reliability and quality of unstructured hybrid mesh generation. *International Journal for Numerical Methods in Fluids*, 45(1):79–108, 2004.

[10] Volodymyr Dyedov, Daniel R. Einstein, Xiangmin Jiao, Andrew P. Kuprat, James P. Carson, and Facundo del Pin. Variational generation of prismatic boundary-layer meshes for biomedical computing. *International Journal for Numerical Methods in Engineering*, 79(8):907–945, 2009.

[11] Yongjie Zhang, Thomas J.R. Hughes, and Chandrajit L. Bajaj. An automatic 3d mesh generation method for domains with multiple materials. *Computer Methods in Applied Mechanics and Engineering*, 199(5–8):405–415, 2010.

[12] Bryan Matthew Klingner and Jonathan Richard Shewchuk. Aggressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, pages 3–23, 2007.

[13] María-Cecilia Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40(18):3313–3324, 1997.

[14] J.M. Escobar, R. Montenegro, E. Rodríguez, and J.M. González-Yuste. Smoothing and local refinement techniques for improving tetrahedral mesh quality. *Computers & Structures*, 83(28–30):2423–2430, 2005.

[15] S.H. Lo. A new mesh generation scheme for arbitrary planar domains. *International Journal for Numerical Methods in Engineering*, 21(8):1403–1426, 1985.

[16] David A. Field. Laplacian smoothing and Delaunay triangulations. *Communications in Applied Numerical Methods*, 4(6):709–712, 1988.

[17] J. Vollmer, R. Mencl, and H. Müller. Improved Laplacian smoothing of noisy surface meshes. In *Computer Graphics Forum*, volume 18, pages 131–138, 1999.

[18] Lori A. Freitag. On combining Laplacian and optimization-based mesh smoothing techniques. *AMD Trends in Unstructured Mesh Generation*, 220:37–44, 1997.

[19] Scott A. Canann, Joseph R. Tristano, and Matthew L. Staten. An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes. In *Proceedings of the 7th International Meshing Roundtable*, pages 479–494, 1998.

[20] Lori A. Freitag Diachin, Patrick M. Knupp, Todd Munson, and Suzanne M. Shontz. A comparison of two optimization methods for mesh quality improvement. *Engineering with Computers*, 22(2):61–74, 2006.

[21] Michael Brewer, Lori A. Freitag Diachin, Patrick M. Knupp, Thomas Leurent, and Darryl Melander. The Mesquite mesh quality improvement toolkit. In *Proceedings of the 12th International Meshing Roundtable*, pages 239–250, 2003.

[22] Patrick M. Knupp. Algebraic mesh quality metrics. *SIAM Journal on Scientific Computing*, 23(1):193–218, 2001.

[23] Patrick M. Knupp. Remarks on mesh quality. In *Proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit*, 2007.

[24] Patrick M. Knupp. Introducing the target-matrix paradigm for mesh optimization via node-movement. In *Proceedings of the 19th International Meshing Roundtable*, pages 68–83. Springer, 2010.

[25] Dimitris Vartziotis. TWT Newsletter: Special issue on regularizing transformations for polygons and polyhedra and their applications. TWT GmbH Science & Innovation, 2007.

[26] Manolis Papadrakakis. Personal communication about a first presentation by Dimitris Vartziotis on regularizing transformations and their application in finite element mesh smoothing, 2008.

[27] Dimitris Vartziotis, Theodoros Athanasiadis, Iraklis Goudas, and Joachim Wipper. Mesh smoothing using the geometric element transformation method. *Computer Methods in Applied Mechanics and Engineering*, 197 (45–48):3760–3767, 2008.

[28] Dimitris Vartziotis and Joachim Wipper. The geometric element transformation method for mixed mesh smoothing. *Engineering with Computers*, 25(3):287–301, 2009.

[29] Dimitris Vartziotis, Joachim Wipper, and Bernd Schwald. The geometric element transformation method for tetrahedral mesh smoothing. *Computer Methods in Applied Mechanics and Engineering*, 199(1–4):169–182, 2009.

[30] Dimitris Vartziotis and Joachim Wipper. A dual element based geometric element transformation method for all-hexahedral mesh smoothing. *Computer Methods in Applied Mechanics and Engineering*, 200(9–12):1186–1203, 2011.

[31] Dimitris Vartziotis and Joachim Wipper. Fast smoothing of mixed volume meshes based on the effective geometric element transformation method. *Computer Methods in Applied Mechanics and Engineering*, 201–204:65–81, 2012.

[32] Dimitris Vartziotis and Manolis Papadrakakis. Adaptive GETMe-based mesh smoothing. to appear in Computer Assisted Methods in Engineering and Science, 2013.

[33] Dimitris Vartziotis. TWT Newsletter: Special issue on the relation between polygon transformations and prime numbers. TWT GmbH Science & Innovation, 2007.

[34] Dimitris Vartziotis and Joachim Wipper. Classification of symmetry generating polygon-transformations and geometric prime algorithms. *Mathematica Pannonica*, 20(2):167–187, 2009.

[35] Dimitris Vartziotis and Joachim Wipper. Characteristic parameter sets and limits of circulant Hermitian polygon transformations. *Linear Algebra and its Applications*, 433(5):945–955, 2010.

[36] Dimitris Vartziotis and Simon Huggenberger. Iterative geometric triangle transformations. *Elemente der Mathematik*, 67(2):68—-83, 2012.

[37] D. Vartziotis, J. Wipper, and M. Papadrakakis. Improving mesh quality and finite element solution accuracy by GETMe smoothing in solving the Poisson equation. *Finite Elements in Analysis and Design*, 66(1–4):36–52, 2013.

[38] Thomas James Wilson. Simultaneous untangling and smoothing of hexahedral meshes. Master's thesis, Universitat Politeècnica de Catalunya, 2011.

[39] David Merriell. Further remarks on concentric polygons. *The American Mathematical Monthly*, 72(9):960–965, 1965.

[40] John E. Wetzel. Converses of Napoleon's theorem. *The American mathematical monthly*, 99(4):339–351, 1992.

[41] Michael Fox. Napoleon triangles and adventitious angles. *The Mathematical Gazette*, 82(495):413–422, 1998.

[42] Wolfgang Schuster. Regularisierung von Polygonen. *Mathematische Semesterberichte*, 45(1):77–94, 1998.

[43] Dimitris Vartziotis. Problem 11328. *American Mathematical Monthly*, 114: 925, 2007.

[44] Barukh Ziv. Napoleon-like configurations and sequences of triangles. *Forum Geometricorum*, 2:115–128, 2002.

[45] Stephen B. Gray. Generalizing the Petr-Douglas-Neumann Theorem on n-gons. *American Mathematical Monthly*, 110(3):210–227, 2003.

[46] Daniel B. Shapiro. A periodicity problem in plane geometry. *The American Mathematical Monthly*, 91(2):97–108, 1984.

[47] G.C. Shephard. Sequences of smoothed polygons. In András Bezdek, editor, *Discrete Geometry*, Pure and Applied Mathematics, pages 407–430. Marcel Dekker, 2003.

[48] O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals.* Elsevier Butterworth-Heinemann, 6th edition, 2005.

[49] K.K. Gupta and J.L. Meek. A brief history of the beginning of the finite element method. *International Journal for Numerical Methods in Engineering*, 39:3761–3774, 1996.

[50] R.W. Clough. Early history of the finite element method from the view point of a pioneer. *International Journal for Numerical Methods in Engineering*, 60(1):283–287, 2004.

[51] K.H. Huebner, D.L. Dewhirst, D.E. Smith, and T.G. Byrom. *The Finite Element Method for Engineers.* Wiley, 4th edition, 2001.

[52] A. Samuelsson and O.C. Zienkiewicz. History of the stiffness method. *International Journal for Numerical Methods in Engineering*, 67(2):149–157, 2006.

[53] Lazarus Teneketzis Tenek and John H. Argyris. *Finite element analysis for composite structures.* Kluwer Academic Publishers, 1998.

[54] John H. Argyris. Personal communication, 1995.

[55] John H. Argyris. *Energy theorems and structural analysis.* Butterworth, 1960.

[56] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23, 1943.

[57] John William Strutt Rayleigh. *The Theory of Sound.* Dover Publications, 2nd edition, 1896.

[58] Walter Ritz. Über eine neue Methode zur Lösung gewisser Variationsprobleme der mathematischen Physik. *Journal für die reine und angewandte Mathematik*, 135:1–61, 1909.

[59] M.J. Turner, R.W. Clough, H.C. Martin, and L. Topp. Stiffness and deflection analysis of complex structures. *Journal of the Aeronautical Sciences*, 25:805–823, 1956.

[60] Ray W. Clough. The finite element method in plane stress analysis. In *Proceedings of the 2nd Conference on Electronic Computation*. A.S.C.E. Structural Division. Pittsburgh, Pennsylvania, 1969.

[61] Gilbert Strang and George Fix. *An Analysis of the Finite Element Method*. Wellesley-Cambridge Press, 2nd edition, 2008.

[62] I. Babuška and A.K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.

[63] Jonathan Richard Shewchuk. What is a good linear element? Interpolation, conditioning, and quality measures. In *Proceedings of the 11th International Meshing Roundtable*, pages 115–126, 2002.

[64] M. Berzins. Mesh quality: a function of geometry, error estimates or both? *Engineering with Computers*, 15(3):236–247, 1999.

[65] Mark A. Yerry and Mark S. Shephard. A modified quadtree approach to finite element mesh generation. *IEEE Computer Graphics and Applications*, 3(1):39–46, 1983.

[66] Mark A. Yerry and Mark S. Shephard. Automatic mesh generation for three-dimensional solids. *Computers & Structures*, 20(1–3):31–39, 1985.

[67] Mark S. Shephard and Marcel K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 32(4):709–749, 1991.

[68] Marshall Bern, David Eppstein, and John Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, 1994.

[69] Pascal J. Frey and Loïc Marechal. Fast adaptive quadtree mesh generation. In *Proceedings of the 7th International Meshing Roundtable*, pages 211–224, 1998.

[70] Rainald Löhner and Paresh Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, 8(10):1135–1149, 1988.

[71] Yasushi Ito, Alan M. Shih, and Bharat K. Soni. Reliable isotropic tetrahedral mesh generation based on an advancing front method. In *Proceedings of the 13th International Meshing Roundtable*, pages 95–106, 2004.

[72] Jonathan Richard Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, pages 86–95, 1998.

[73] Qiang Du and Desheng Wang. Tetrahedral mesh generation and optimization based on centroidal Voronoi tessellations. *International Journal for Numerical Methods in Engineering*, 56(9):1355–1373, 2003.

[74] B. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[75] Jonathan Richard Shewchuk. Lecture notes on Delaunay mesh generation. Technical report, Department of Electrical Engineering and Computer Science, University of California at Berkeley, 1999. CA 94720.

[76] D.T. Lee and B.J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Parallel Programming*, 9(3):219–242, 1980.

[77] R.A. Dwyer. A faster divide-and-conquer algorithm for constructing Delaunay triangulations. *Algorithmica*, 2(1):137–151, 1987.

[78] Frank T. Reusch. Entwicklung einer dynamischen Programmstruktur für einen 3D-Volumengenerator. Master's thesis, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, University of Stuttgart, Germany, 1988.

[79] Manuel F. F. Delgado. Entwicklung und Erprobung eines Generators für Volumenelemente. Master's thesis, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, University of Stuttgart, Germany, 1988.

[80] T.S. Li, R.M. McKeag, and C.G. Armstrong. Hexahedral meshing using midpoint subdivision and integer programming. *Computer Methods in Applied Mechanics and Engineering*, 124(1–2):171–193, 1995.

[81] M.A. Price and C.G. Armstrong. Hexahedral mesh generation by medial surface subdivision: Part ii. solids with flat and concave edges. *International Journal for Numerical Methods in Engineering*, 40(1):111–136, 1997.

[82] Timothy J. Tautges. The generation of hexahedral meshes for assembly geometry: survey and progress. *International Journal for Numerical Methods in Engineering*, 50(12):2617–2642, 2001.

[83] Patrick M. Knupp. Next-generation sweep tool: A method for generating all-hex meshes on two-and-one-half dimensional geometries. In *Proceedings of the 7th International Meshing Roundtable*, pages 505–513, 1998.

[84] Jason Shepherd, Scott A. Mitchell, Patrick M. Knupp, and David White. Methods for multisweep automation. In *Proceedings of the 9th International Meshing Roundtable*, pages 77–87, 2000.

[85] Xevi Roca, Josep Sarrate, and Antonio Huerta. Surface mesh projection for hexahedral mesh generation by sweeping. In *Proceedings of the 13th International Meshing Roundtable*, pages 169–180, 2004.

[86] Yongjie Zhang and Chandrajit Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. In *Proceedings of the 13th International Meshing Roundtable*, pages 365–376, 2004.

[87] Yasushi Ito, Alan M. Shih, and Bharat K. Soni. Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates. *International Journal for Numerical Methods in Engineering*, 77(13):1809–1833, 2009.

[88] Matthew L. Staten, Robert A. Kerr, Steven J. Owen, and Ted D. Blacker. Unconstrained paving and plastering: Progress update. In *Proceedings of the 15th International Meshing Roundtable*, pages 469–486, 2006.

[89] Timothy J. Tautges, Ted Blacker, and Scott A. Mitchell. The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 39(19):3327–3349, 1996.

[90] N.T. Folwell and S.A. Mitchell. Reliable whisker weaving via curve contraction. *Engineering with Computers*, 15(3):292–302, 1999.

[91] Steve J. Owen and Mark S. Shephard (eds.). International Journal for Numerical Methods in Engineering, Special Issue: Trends in Unstructured Mesh Generation, 2003.

[92] Manolis Papadrakakis, editor. *Solving large-scale problems in mechanics. The development and application of computational solution methods.* John Wiley & Sons, 1993.

[93] Lori A. Freitag and Carl Ollivier-Gooch. A cost/benefit analysis of simplicial mesh improvement techniques as measured by solution efficiency. *International Journal of Computational Geometry & Applications*, 10(4): 361–382, 2000.

[94] Nestor Calvo, Sergio R. Idelsohn, and Eugenio Oñate. The extended Delaunay tessellation. *Engineering Computations: International Journal for Computer-Aided Engineering*, 20(5/6), 2003.

[95] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge University Press, 2001.

[96] A. El-Hamalawi. A 2d combined advancing front-Delaunay mesh generation scheme. *Finite Elements in Analysis and Design*, 40(9–10):967–989, 2004.

[97] Igor Sazonov, Desheng Wang, Oubay Hassan, Kenneth Morgan, and Nigel Weatherill. A stitching method for the generation of unstructured meshes for use with co-volume solution techniques. *Computer Methods in Applied Mechanics and Engineering*, 195(13–16):1826–1845, 2006.

[98] R.B. Simpson. Geometry independence for a meshing engine for 2d manifolds. *International Journal for Numerical Methods in Engineering*, 60(3): 675–694, 2004.

[99] Lori A. Freitag and Carl Ollivier-Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering*, 40(21):3979–4002, 1997.

[100] Tian Zhou and Kenji Shimada. An angle-based approach to two-dimensional mesh smoothing. In *Proceedings of the 9th International Meshing Roundtable*, pages 373–384, 2000.

[101] J.Z. Zhu, O.C. Zienkiewicz, E. Hinton, and J. Wu. A new approach to the development of automatic quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):849–866, 1991.

[102] Mao Zhihong, Ma Lizhuang, Zhao Mingxi, and Li Zhong. A modified Laplacian smoothing approach with mesh saliency. In Andreas Butz, Brian Fisher, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics*, volume 4073 of *Lecture Notes in Computer Science*, pages 105–113. Springer, 2006.

[103] D. Rypl. Approaches to discretization of 3D surfaces. In *CTU Reports*, volume 7 (2). CTU Publishing House, 2003.

[104] Nina Amenta, Marshall Bern, and David Eppstein. Optimal point placement for mesh smoothing. *Journal of Algorithms*, 30:302–322, 1999.

[105] Hongtao Xu and Timothy S. Newman. An angle-based optimization approach for 2D finite element mesh smoothing. *Finite Elements in Analysis and Design*, 42(13):1150–1164, 2006.

[106] Yongjie Zhang, Chandrajit Bajaj, and Guoliang Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Communications in Numerical Methods in Engineering*, 25(1):1–18, 2009.

[107] Long Chen. Mesh smoothing schemes based on optimal Delaunay triangulations. In *Proceedings of the 13th International Meshing Roundtable*, pages 109–120, 2004.

[108] Lori A. Freitag, Mark Jones, and Paul Plassmann. A parallel algorithm for mesh smoothing. *SIAM Journal on Scientific Computing*, 20(6):2023–2040, 1999.

[109] N.A. Calvo and S.R. Idelsohn. All-hexahedral mesh smoothing with a node-based measure of quality. *International Journal for Numerical Methods in Engineering*, 50(8):1957–1967, 2001.

[110] Xiang-Yang Li and Lori A. Freitag. Optimization-based quadrilateral and hexahedral mesh untangling and smoothing techniques. Technical report, Argonne National Laboratory, 1999.

[111] A. Egemen Yilmaz and Mustafa Kuzuoglu. A particle swarm optimization approach for hexahedral mesh smoothing. *International Journal for Numerical Methods in Fluids*, 60(1):55–78, 2009.

[112] Simon Kulovec, Leon Kos, and Jožef Duhovnik. Mesh smoothing with global optimization under constraints. *Strojniški vestnik - Journal of Mechanical Engineering*, 57(7–8):555–567, 2011.

[113] Yannick Sirois, Julien Dompierre, Marie-Gabrielle Vallet, and François Guibault. Hybrid mesh smoothing based on Riemannian metric non-conformity minimization. *Finite Elements in Analysis and Design*, 46(1–2): 47–60, 2010.

[114] Bala Balendran. A direct smoothing method for surface meshes. In *Proceedings of the 8th International Meshing Roundtable*, pages 189–193, 1999.

[115] Maria Savchenko, Olga Egorova, Ichiro Hagiwara, and Vladimir Savchenko. An approach to improving triangular surface mesh. *JSME International Journal, Series C, Mechanical Systems, Machine Elements, and Manufacturing*, 48(2):137–148, 2005.

[116] Tian Xia and Eric Shaffer. Streaming tetrahedral mesh optimization. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 281–286, 2008.

[117] Zhijian Chen, Joseph R. Tristano, and Wa Kwok. Combined Laplacian and optimization-based smoothing for quadratic mixed surface meshes. In *Proceedings of the 12th International Meshing Roundtable*, 2003.

[118] J.M. Escobar, G. Montero, R. Montenegro, and E. Rodríguez. An algebraic method for smoothing surface triangulations on a local parametric space. *International Journal for Numerical Methods in Engineering*, 66(4):740–760, 2006.

[119] Rao V. Garimella, Mikhail J. Shashkov, and Patrick M. Knupp. Triangular and quadrilateral surface mesh quality optimization using local parametrization. *Computer Methods in Applied Mechanics and Engineering*, 193(9–11):913–928, 2004.

[120] Xiangmin Jiao, Duo Wang, and Hongyuan Zha. Simple and effective variational optimization of surface and volume triangulations. In *Proceedings of the 17th International Meshing Roundtable*, pages 315–332, 2008.

[121] Francois Courty, David Leservoisier, Paul-Louis George, and Alain Dervieux. Continuous metrics and mesh adaptation. *Applied Numerical Mathematics*, 56(2):117–145, 2006.

[122] Lori A. Freitag Diachin, Patrick Knupp, Todd Munson, and Suzanne Shontz. A comparison of inexact Newton and coordinate descent mesh optimization techniques. In *Proceedings of the 13th International Meshing Roundtable*, pages 243–254, 2004.

[123] Todd Munson. Mesh shape-quality optimization using the inverse mean-ratio metric. *Mathematical Programming, Series A*, 110(3):561–590, 2007.

[124] Patrick M. Knupp. Hexahedral mesh untangling & algebraic mesh quality metrics. In *Proceedings of the 9th International Meshing Roundtable*, pages 173–183, 2000.

[125] R. Löhner, K. Morgan, and O. Zienkiewicz. Adaptive grid refinement for the compressible Euler equations. In I. Babuǔka, O. Zienkiewicz, J. Gago, and E.d.A. Oliveira, editors, *Accuracy Estimates and Adaptivity for Finite Elements*, pages 281–297. John Wiley & Sons Ltd, 1986.

[126] Kenji Shimada and David C. Gossard. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In *Proceedings of the 3rd ACM symposium on Solid modeling and applications*, pages 409–419, 1995.

[127] Olga Egorova, Maria Savchenko, Nikita Kojekine, Irina Semonova, Ichiro Hagiwara, and Vladimir Savchenko. Improvement of mesh quality using a statistical approach. In *Proceedings of the 3rd IASTED international conference on visualization, imaging, and image processing*, volume 2, pages 1016–1021, 2003.

[128] Agustín Menéndez-Díaz, Celestino González-Nicieza, and Arturo Ernesto Álvarez-Vigil. Hexahedral mesh smoothing using a direct method. *Computers & Geosciences*, 31(4):453–463, 2005.

[129] Maria Savchenko, Olga Egorova, Ichiro Hagiwara, and Vladimir Savchenko. Hexahedral mesh improvement algorithm. *JSME International Journal. Series C. Mechanical Systems, Machine Elements and Manufacturing*, 48 (2):130–136, 2005.

[130] V. Savchenko, M. Savchenko, O. Egorova, and I. Hagiwara. Mesh quality improvement: Radial basis functions approach. *International Journal of Computer Mathematics*, 85(10):1589–1607, 2008.

[131] Glen Hansen, Andrew Zardecki, Doran Greening, and Randy Bos. A finite element method for three-dimensional unstructured grid smoothing. *Journal of Computational Physics*, 202(1):281–297, 2005.

[132] Alejandro R. Diaz, Noboru Kikuchi, and John E. Taylor. A method of grid optimization for finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 41(1):29–45, 1983.

[133] Mark Ainsworth and J. Tinsley Oden. *A Posteriori Error Estimation in Finite Element Analysis.* Wiley, 2000.

[134] Randolph E. Bank and R. Kent Smith. Mesh smoothing using a posteriori error estimates. *SIAM Journal on Numerical Analysis*, 34(3):979–997, 1997.

[135] C.C. Pain, A.P. Umpleby, C.R.E. de Oliveira, and A.J.H. Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190(29–30):3771–3796, 2001.

[136] R.E. Jones. A self-organizing mesh generation program. *Journal of Pressure Vessel Technology*, 96:193–199, 1974.

[137] Mesquite: mesh quality improvement toolkit. `http://www.cs.sandia.gov/optimization/knupp/Mesquite.html`, 2012.

[138] W. Rutherford. VII Quest. 1439. *Ladies' Diary*, 122:47, 1825.

[139] H.S.M. Coxeter and S.L. Greitzer. *Geometry revisited.* The Mathematical Association of America: New Mathematical Library, Random House, New York, 1967.

[140] P. Von Finsler and H. Hadwiger. Einige Relationen im Dreieck. *Commentarii Mathematici Helvetici*, 10(1):316–326, 1937.

[141] I.M. Yaglom. *Geometric transformations.* Random House, New York, 1962.

[142] L. Kiepert. Solution de question 864. *Nouv Ann Math*, 8:40–42, 1869.

[143] E.R. Berlekamp, E.N. Gilbert, and F.W. Sinden. A polygon problem. *The American Mathematical Monthly*, 72(3):233–241, 1965.

[144] Pavel Pech. The harmonic analysis of polygons and Napoleon's theorem. *Journal for Geoemetry and Graphics*, 5(1):13–22, 2001.

[145] Philip J. Davis. *Circulant matrices.* Chelsea Publishing, 2nd edition, 1994.

[146] Robert M. Gray. Toeplitz and circulant matrices: A review. *Foundations and Trends in Communications and Information Theory*, 2(3):155–239, 2006.

[147] Dimitris Vartziotis and Joachim Wipper. On the construction of regular polygons and generalized Napoleon vertices. *Forum Geometricorum*, 9:213–223, 2009.

[148] Horst Martini. On the theorem of Napoleon and related topics. *Mathematische Semesterberichte*, 43(1):47–64, 1996.

[149] Philip J. Davis. Cyclic transformations of n-gons and related quadratic forms. *Linear Algebra and its Applications*, 25:57–75, 1979.

[150] K. Petr. Ein Satz über Vielecke. *Archiv der Mathematik und Physik: mit besonderer Rücksicht auf die Bedürfnisse der Lehrer an höheren Unterrichtsanstalten*, 13:29–31, 1908.

[151] Jesse Douglas. On linear polygon transformations. *Bulletin of the American Mathematical Society*, 46:551–560, 1940.

[152] B.H. Neumann. Some remarks on polygons. *Journal of the London Mathematical Society*, 16:230–245, 1941.

[153] Geng-zhe Chang and Philip J. Davis. A circulant formulation of the Napoleon-Douglas-Neumann theorem. *Linear Algebra and its Applications*, 54:87–95, 1983.

[154] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 3rd edition, 2000.

[155] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.

[156] Khronos OpenCL Working Group. *The OpenCL Specification, Version 1.2, Document Revision 15*. Khronos Group, 2011.

[157] *NVIDIA CUDA C Programming Guide, Version 4.2*. NVIDIA, 2012.

[158] *OpenACC Application Programming Interface, Version 1.0*. OpenACC-Standard.org, 2011.

[159] M. Papadrakakis, G. Stavroulakis, and A. Karatarakis. A new era in scientific computing: Domain decomposition methods in hybrid CPU-GPU architectures. *Comput Methods Appl Mech Eng*, 200(13–16):1490–1508, 2011.

[160] OpenMP Architecture Review Board. *OpenMP Application Program Interface, Version 3.1*, 2011.

[161] C.J. Stimpson, C.D. Ernst, P. Knupp, P.P. Pébay, and D. Thompson. The Verdict Geometric Quality Library. Technical Report SAND2007-1751, Sandia National Laboratories, 2007.

[162] GNU Compiler Collection (GCC). `http://gcc.gnu.org/`, last access: July 4, 2012.

[163] Dimitris Vartziotis, Alkis Poulis, Victor Fäßler, Costas Vartziotis, and Charis Kolios. Integrated digital engineering methodology for virtual orthopedics surgery planning. In *Proceedings of the International Special Topic*

Conference on Information Technology in Biomedicine, Ioannina, Greece, 2006.

[164] Dimitris Vartziotis. Modular endoprosthesis for total hip arthroplasty (in greek). Greek Patent, Publication number: GR20050100072 (A), 2006.

[165] Aim@Shape Mesh Repository. `http://shapes.aim-at-shape.net/`, last access: July 17, 2012.

[166] Netgen: automatic 3D tetrahedral mesh generator. `http://sourceforge.net/projects/netgen-mesher/`, last access: October 31, 2012.

[167] Klingner and Shewchuk: Example meshes. `http://www.cs.berkeley.edu/b-cam/Papers/Klingner-2007-ATM/index.html`, accessed March 17, 2009.

[168] Barbara Cutler, Julie Dorsey, and Leonard McMillan. Simplification and improvement of tetrahedral models for simulation. In R. Scopigno and D. Zorin, editors, *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 93–102, 2004.

[169] TWT. Aletis vehicle design and prototyping: Aerodynamic analysis. Technical Report 04/04, TWT GmbH, 2001.

[170] Gamma Project Mesh Database. `http://www-roc.inria.fr/gamma/download/`, last access: September 9, 2010.

[171] J. Heuser. Aorta scheme image. `http://de.wikipedia.org/w/index.php?title=Datei:Aorta_scheme.jpg&filetimestamp=20060328102541`, accessed November 10, 2010.

[172] *MPI: A Message-Passing Interface Standard, Version 2.2.* Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee, 2009.

[173] Drexel University, Geometric & Intelligent Computing Laboratory model repository. `http://edge.cs.drexel.edu/repository/`, last access: July 3, 2012.

[174] J.N. Reddy and D.K. Gartling. *The Finite Element Method in Heat Transfer and Fluid Dynamics.* CRC Press, 3rd edition, 2010.

[175] GetFEM++: An open-source finite element library. `http://home.gna.org/getfem/`, last access: April 8, 2011.

Index