DISSERTATION

Symmetry- and Similarity-Aware Volumetric Meshing

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

FLORIAN RUDOLF

[redacted]

Matr. Nr. 0326156

[redacted]

Wien, im September 2016 _____

To Maria and Hansi, thank you.

Abstract

Volumetric mesh generation plays an important role in computer-aided engineering processes. Often, objects used in computer-aided engineering, for example a gear, show symmetries or similarities. So far, available volumetric mesh generation and adaptation algorithms do not consider symmetries or similarities and thus ignore potential memory and algorithm optimizations. For example, instead of generating and storing a mesh of a regular 16-polygon, the mesh of one single *slice* can be generated and stored together with the information that this so-called *mesh template* is copied and rotated 16 times to yield the desired 16-polygon mesh. In this particular case, improvements in memory and algorithm runtime of a factor of 16 are expected.

This work investigates how the generation, usage, and storage of volumetric meshes can benefit from symmetries and similarities. In particular, impacts and optimizations in memory usage, algorithm runtime, and mesh element quality are investigated. For this reason, a theory based on so-called *templated structures* is developed. These templated structures contain mesh templates which are instanced (potentially) multiple times using geometrical transformations to obtain the resulting mesh. The proposed theory uses an abstract approach to support both, symmetries as well as similarities. Furthermore, theoretical mechanisms are developed to tackle potential conformity issues at mesh instance interfaces.

Based on these theoretical approaches, data structures and algorithms for adapting and generating templated meshes are developed in this thesis. In particular, two different algorithms for templated mesh generation are proposed and investigated. These algorithms are also specialized for symmetries, being reflective and rotational symmetries and their combinations. Additionally, a selection of popular mesh adaptation algorithms is investigated for their use with *templated structures*.

The benefits of the proposed data structures and algorithms are investigated and discussed in a benchmark-based survey. Expected memory savings and runtime speedups of the *templated mesh* generation process are indeed achieved for all considered two-dimensional and most three-dimensional objects. For three-dimensional objects with high rotational symmetry orders, the improvements are lower than expected but at least a factor of 15. Memory savings drop significantly when including memory requirements for the system matrix of a finite element method. Therefore, a templated matrix data structure is developed, which compensates these losses. Mesh element qualities of templated meshes are as good as conventionally generated meshes in most scenarios and minimally worse otherwise.

Additionally, effects of symmetric and non-symmetric meshes on finite element based simulations are investigated. If the simulation domain is symmetric, the mathematical solution to the boundary value problem with boundary conditions transformed by the symmetry transformation is equal to the transformed solution of the initial problem. However, an analysis shows, that a symmetric mesh is required for this statement to hold for the numerical solutions with a finite element method.

Zusammenfassung

Die Erzeugung von volumetrischen Gittern spielt eine wichtige Rolle in diskretisierungs-basierten rechnergestützten Entwicklungsprozessen. In diesem Zusammenhang relevante Objekte besitzen oft Symmetrien oder bestehen aus gleichartigen Bausteinen, welche jedoch von Algorithmen zur Erzeugung oder Adaptierung von volumetrischen Gittern bisher nicht berücksichtigt werden, was zu ungenutztem Speicher- und Laufzeitoptimierungspotential führt. Beispielsweise kann bei einem regulären 16-Eck anstelle des Gitters des Gesamtobjekts nur das Gitter eines *Tortenecks* generiert und gespeichert werden. In diesem Fall ist eine Ersparnis an Speicherbegarf und Laufzeit von einen Faktor 16 zu erwarten, auch wenn die benötigte Zusatzinformation, in diesem Fall die 16-malige Kopie und Rotation des Tortenecks, einen zusätzlichen Aufwand bedeutet.

In dieser Arbeit wird untersucht, ob und wie Symmetrien und Ähnlichkeiten bei Erzeugung, Verwendung, und Speicherung von volumetrischen Gittern vorteilhaft genutzt werden können. Im Speziellen werden Auswirkungen und Optimierungen auf den Speicherverbrauch, die Algorithmenlaufzeit, und die Gitterelementqualität analysiert. Zu diesem Zweck wird eine Theorie entwickelt, welche sich mit potentiell mehrfacher Instanzierung von sogenannten *Vorlagenstrukturen* anhand von geometrischer Transformationen beschäftigt. Die vorgestellte Theorie benutzt einen abstrakten Ansatz, um sowohl Symmetrien als auch andere Ähnlichkeiten zu unterstützen. Zusätzlich werden Mechanismen vorgestellt, welche die Kohärenz der Schnittstellen von verschiedenen Vergitterungsinstanzen sicherstellen.

Basierend auf diesen theoretischen Ansätzen werden Datenstrukturen und Algorithmen zum Erzeugen und Adaptieren von Gittern entwickelt. Konkret werden zwei Algorithmen zum Erzeugung von Vorlagengittern vorgestellt und untersucht, welche auch für Spiegelungs- und Rotationssymmetrien sowie deren Kombinationen spezialisiert werden. Zusätzlich wird eine Auswahl von populären Gitteradaptierungsalgorithmen auf deren Benutzbarkeit mit Vorlagenstrukturen untersucht.

Die Vorteile der vorgestellten Datenstrukturen und Algorithmen werden in einer Leistungsvergleichstudie untersucht und diskutiert. Für alle zwei-dimensionalen und für die meisten dreidimensionalen Objekte sind die Verbesserungen beim Speicherverbrauch und in der Algorithmenlaufzeit von der Erzeugung von Vorlagengittern mindestens so hoch wie erwartet. Für dreidimensionale Objekte mit hoher Rotationssymmetrieordnung sind die Verbesserungen geringer, allerdings mindestens ein Faktor 15. Wenn der Speicherverbrauch der Systemmatrix einer Finite-Elemente-Methode berücksichtigt wird, sinken die Verbesserungen im Speicherverbrauch allerdings sigifikant. Um diese Verluste zu kompensieren, wird eine Datenstruktur für die Systemmatrix entwickelt, welche auf den Konzepten der Vorlagengitter basiert. Die Qualität der Gitterelemente von Vorlagengittern ist in den meisten Fällen mindestens so gut wie die von konventionell erzeugten Gittern und minimal schlechter sonst.

Zusätzlich werden die Effekte von symmetrischen und nicht symmetrischen Gittern auf Simulationen basierend auf der Finite-Elemente-Methode untersucht. Wenn das Simulationsgebiet symmetrisch ist, dann ist die mathematische Lösung des Simulationsproblems, bei dem die Randbedingungen des Initialproblem mit der Symmetrietransformation transformiert werden, gleich der transformierten Lösung des Initalproblems. Eine Analyse zeigt allerdings, dass ein symmetrisches Gitter benötigt wird damit, die vorige Aussage auch für die nummerischen Lösungen einer Finite-Elemente-Methode zutrifft.

Acknowledgement

Looking back from now, writing this thesis and finishing my PhD studies was hard, challenging, frustrating, motivating, exhausting, enlightening, demanding, educative, ... In short, I had a really intense couple of months. Now, that this time has come to an end, I would like to take the opportunity to say some personal words.

First and foremost, I offer my sincerest gratitude to my supervisor Prof. Siegfried Selberherr for giving me the opportunity to join the Institute for Microelectronics as well as supporting me all the way through my time there. He not only provided knowledge and wisdom where it was needed but also gave me freedom to find my way in academia. Prof. Selberherr taught me to always look at the big picture instead of getting lost in details. I learned a lot from his advice, and the quality of my scientific work would not be at this level without him. Thank you a lot!

I want to thank Prof. Helmut Pottmann, who gave me mathematical and geometrical advice to increase the quality of this work. I am also grateful for his invitations to the Vienna geometry society where I got the opportunity to visit interesting and educative talks.

I am also thankful to the Institute for Microelectronics, especially Prof. Erasmus Langer and Prof. Tibor Grasser, for providing a calm, friendly, and productive work environment.

The next few words are dedicated to my very good friend and mentor Karl *Karli* Rupp. After finishing my masters degree in 2012, he introduced me to Prof. Selberherr who then became my supervisor. Later, when I had to choose the topic of my dissertation, Karli encouraged me to take the more risky but also more interesting topic, which was ultimately the right choice. Karli always was there for me and gave me good advice, for academic as well as personal matters. For these and all the unmentioned support I am more than grateful.

I also thank my colleague Josef Weinbub for his support in the last four years. We had very interesting and enlightening scientific discussions which helped me shaping my academic work. I especially want to thank him for his technical but also linguistic feedback which increased the quality of my scientific publications. His motivating and pushing phrases, especially during the work on my PhD thesis, helped me to keep on going and not to lose hope. Thank you!

This list is not complete without thanking Andreas *Ivan* Morhammer for being a supportive colleague, especially when I had mathematical issues with my scientific work. The theoretical and mathematical discussions with Andreas helped me to develop a solid theoretical basis for this work. Thank you!

Finally, I would also like to thank all the people close to me for trusting and believing in me, emotionally and personally supporting me, guiding me, being the family and friends that I needed, and much, much more. Thank you, Maria, Hansi, Alex, Mäx, Klaus, little Veronika, Josipa, *big* Veronika, David, Karini, Bertl, Rafeala (for five and a half beautiful years) and all the girls and guys from the *A-Team*.

Contents

List of Definitions

List of Lemmas

List of Figures

List of Algorithms

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
CAE	Computer-aided engineering
TCAD	Technology computer-aided design
CFD	Computational fluid dynamics
FDM	Finite difference method
FVM	Finite volume method
FEM	Finite element method
PDE	Partial differential equation
SVB	Shared vertex bookkeeping
TS	Templated structure
SI	Structure instance
IG	Instance graph
BPP	Boundary patch partition
SAO	Small angle optimization
ST	Slice template

Nomenclature

$\mathbb{R}$	The real numbers
$\mathbb{R}^n$	An n-dimensional vector space over the real numbers
a	A scalar
$\boldsymbol{a}$	A vector, point, or vertex of $\mathbb{R}^n$
$\mathbf{0}$	The zero vector or the origin
f	A function
$\mathrm{img}(f)$	The image of a function
$\mathrm{dom}(f)$	The domain of a function
A	A subset of $\mathbb{R}^n$
$\mathcal{P}(A)$	The power set of a set
$\mathcal{P}^\star(A)$	The power set of a set without the empty set: $\mathcal{P}^\star(A) = \mathcal{P}(A) \setminus \emptyset$
$\mathrm{cl}(A)$	The closure of a set (cf. Definition A.1)
$\mathrm{int}(A)$	The interior of a set (cf. Definition A.1)
$\mathrm{bnd}(A)$	The boundary of a set (cf. Definition A.1)
$\mathrm{int}^\star(A)$	The relative interior of a set (cf. Definition A.8)
$\mathrm{bnd}^\star(A)$	The relative boundary of a set (cf. Definition A.8)
$\mathrm{DIM}(X)$	The dimension of a set (cf. Definition A.6)
$\mathbb{H}_{\boldsymbol{n},d}$	A hyperplane with normal vector $\boldsymbol{n}$ and distance to origin d (cf. Definition A.5)
$\overline{\mathcal{B}}^n_r(\boldsymbol{x})$	An n-ball with radius r and center (x) (cf. Definition A.3)
$\mathcal{B}^n_r(\boldsymbol{x})$	An open n-ball with radius r and center (x) (cf. Definition A.3)
$\mathcal{H}^n_r(\boldsymbol{x})$	An n-half-ball with radius r and center (x) (cf. Definition A.3)
$\mathcal{G}$	A geometry (cf. Definition 2.1)
$(\mathcal{G}, \widetilde{\xi})$	A multi-region geometry (cf. Definition 2.2)
$\mathcal{E}$	An element space (cf. Definition A.6)
$\mathcal{M}$	A mesh (cf. Definition 2.4)

$(\mathcal{M}, \xi)$	A multi-region mesh (cf. Definition 2.6)
$\text{elem}_k(\mathcal{E})$	The set of all k-dimensional elements of an element space (cf. Definition A.7)
$\text{facets}(\mathcal{E})$	The set of all facets of an element space (cf. Definition A.7)
$\text{facets}_{\mathcal{E}}(E)$	The set of all facets of an element (based on an element space) (cf. Definition A.9)
$\text{faces}_{\mathcal{E}}(E)$	The set of all faces of an element (based on an element space) (cf. Definition A.9)
$\text{aff}(X)$	The affine hull of a set of points (cf. Definition A.15)
$\text{conv}(X)$	The convex hull of a set of points (cf. Definition A.16)
$\text{simplex}(X)$	A simplex based on a set of points (cf. Definition A.17)
$\mathfrak{M}^n$	The n-dimensional manifold space (cf. Definition A.6)
$\mathfrak{E}^n$	The n-dimensional mesh element space (cf. Definition A.25)
$\mathfrak{L}^n$	The n-dimensional geometry space (cf. Definition A.25)
$\text{ip}(A, B)$	The intersection of two partitions (cf. Definition A.29)
$\text{refine}(S, P)$	The partition refinement of two partitions (cf. Definition A.30)
$\text{us}(\mathcal{M})$	The underlying space of a mesh (cf. Definition A.6)
$\text{geo}(\mathcal{M})$	The geometry of a mesh (cf. Definition 2.11)
$\mathbb{X}$	A templated structure (cf. Definition 4.3)
$\text{templ}(\mathbb{X}, i)$	The i-th template of a templated structure (cf. Section 4.1)
$\text{inst}(\mathbb{X}, i, j), \text{inst}(\mathbb{X}, T_{i,j})$	The j-th instance of the i-th template (cf. Section 4.1)
$\text{tf}_{i,j}$	The transformation function of the j-th instance of the i-th template (cf. Definition 4.3)
$\text{rid}_{i,j}$	The region indicator of the j-th instance of the i-th template (cf. Definition 4.3)
Γ	A templated mesh (cf. Definition 4.4)
Λ	A templated geometry (cf. Definition 4.4)
$\text{AT}(\mathbb{X})$	The apply-template operator (cf. Section 4.1)
$\text{interf}(\mathbb{X}, I_1, \ldots, I_k)$	The instance interface of a templated structure based on a set of instances (cf. Definition 4.6)
$\text{interf}_{\text{geo}}(\mathbb{X}, I_1, \ldots, I_k)$	The geometry instance interface of a templated structure based on a set of instances (cf. Definition 4.6)
$\mathbb{T}_{T,U}$	A template boundary mapping (cf. Definition 4.7)
$\mathfrak{T}_{\mathbb{X}}$	All valid template boundary mappings of a templated structure (cf. Section 4.3)
$\text{bndpart}(\mathbb{X})$	The boundary patch partition of a templated structure (cf. Section 4.3)
$\sim_{\mathbb{X}}$	The boundary patch relation (cf. Definition 4.8)

Chapter 1

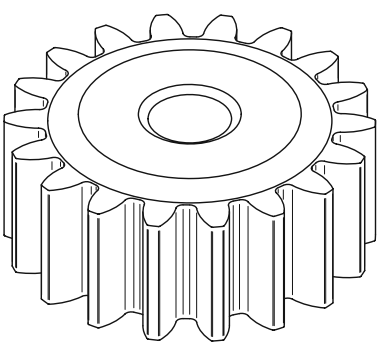
Introduction and Motivation

During the last decades, computer-aided engineering (CAE) methods have become very popular for simulating the behavior of science and engineering problems. For the vast majority of these simulations, differential equations, or their higher-dimensional versions – partial differential equations (PDEs) – are used to mathematically describe the desired physical processes [33]. For example, in technology computer-aided design (TCAD), the continuity equation with the drift-diffusion model is used to describe the behavior of the electrostatic potential and electronic charge carriers within a semiconductor [125]. In computational fluid dynamics (CFD), the Navier-Stokes equations are used to describe the motion of fluids [25]. The combinations of Euler's equations of motions and the Euler-Cauchy stress principle yields a system of PDEs, which is used for stress analysis in structural mechanics [110].

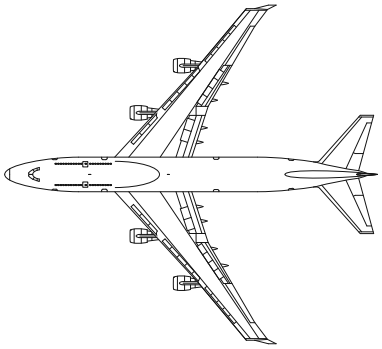
Besides the mathematical model, a simulation scenario additionally requires a simulation domain, on which the PDE is applied, and boundary conditions which represent influences from the outside. This is called a boundary value problem or, generally speaking, a simulation problem [23]. However, even if the existence and uniqueness of the solution of a boundary value problem can be proven mathematically, the number of PDEs with an explicit analytical solution is small [51]. The two examples named above, i.e., the drift-diffusion equations and the Navier-Stokes equations, do not have explicit analytical solutions in general. Therefore, numerical approaches have to be used to calculate approximate solutions. Popular numerical approaches are the finite difference method (FDM) [112], the finite volume method (FVM) [111], and the finite element method (FEM) [57]. These methods require that the simulation domain, which in its most general form is represented by a geometry, is decomposed (also referred to as discretized) into a discrete mesh to obtain a finite representation of the simulation domain. Hence, meshes are an integral component of a simulation process.

Geometries of objects in CAE scenarios, like a gear in a stress simulation, are usually designed using computer-aided design (CAD) tools. Many objects in CAE applications inherently require *symmetries* to function. For example, a gear which is not rotationally symmetric will not work properly. Consequently, symmetries (cf. Figure 1.1) and *similarities* (cf. Figure 1.2) are of special interest, because they can be used to understand the structure and shape of the geometry. In particular, objects with symmetries and similarities can be constructed from smaller pieces of the object using rigid transformations as shown in Figure 1.3.

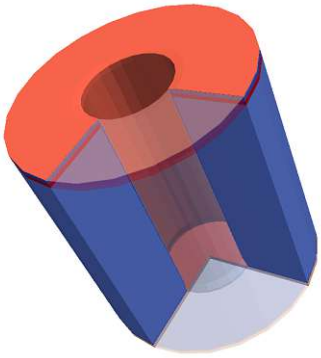
The main motivation of this work is to leverage symmetries and similarities in objects when creating, using, and storing meshes which are ignored by conventional mesh generation and adaptation algorithms. Using the fact that objects with symmetries and similarities can be constructed from smaller pieces (cf. Figure 1.3), a data structure can be defined, where that piece is only stored once hence saving memory. The same idea can be applied for mesh generation processes, where generating a mesh of smaller pieces is considerably less time consuming than meshing the entire geometry.



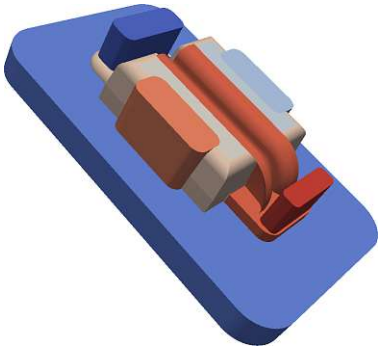
(a) Gear, rotational symmetry[1]



(b) Airplane, reflective symmetry[2]



(c) Open TSV device [28], rotational symmetry



(d) FlexFET device [74], reflective symmetry

Figure 1.1: Objects with *symmetries* used in CAE

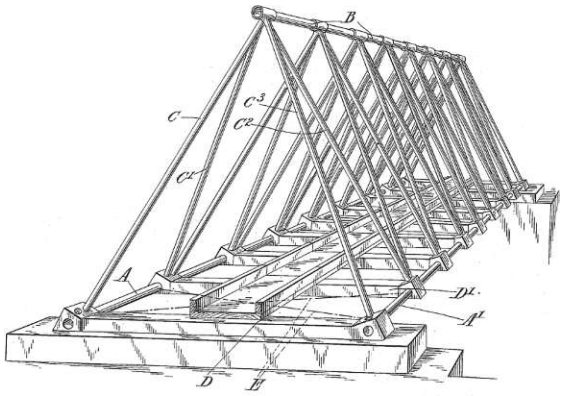


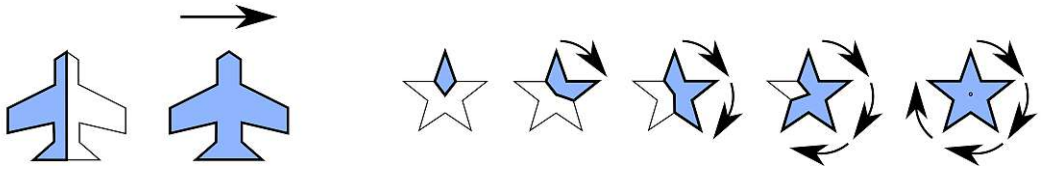
Figure 1.2: A sketch of a bridge[3] with *similarities*

There are multiple parts of this bridge construction, which are *similar* to each other, for example the bars C and C^1 or A and A^1.

[1] Gear image by Inductiveload [Public domain], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3ASpur_Gear_12mm%2C_18t.svg

[2] Aircraft image by Kaboldy (Own work) [CC BY-SA 4.0], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3ABoeing_747-400_3view.svg

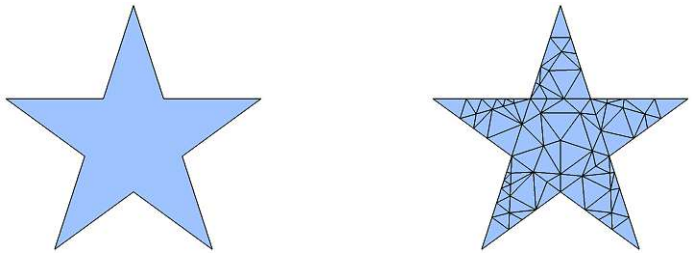
[3] Image by Charles Edward Inglis [Public domain], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3AInglis_bridge_patent.png



(a) Airplane, reflective symmetry **(b)** Star, rotational symmetry

Figure 1.3: Re-construction of objects with *symmetries*

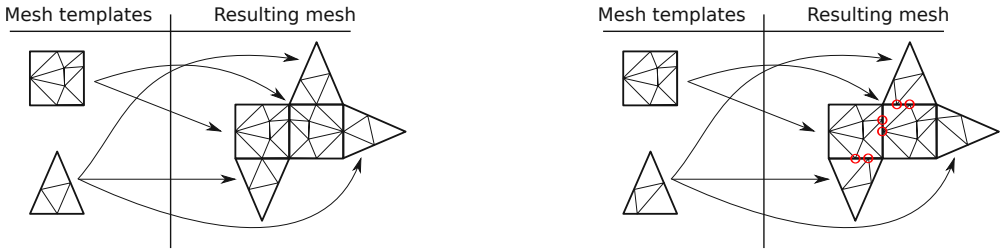
The objects can be reconstructed by using smaller pieces, those being one side of the plane and a jag of the star, respectively. The smaller piece is copied and reflected or rotated around the symmetry center to obtain the desired object.



(a) Symmetric object **(b)** Non-symmetric mesh

Figure 1.4: Symmetric geometry and exemplary non-symmetric mesh

Even though the object itself has a rotational and a reflective symmetry, the mesh has neither.



(a) Conforming templated mesh **(b)** Non-conforming templated mesh

Figure 1.5: Templated meshes

The resulting mesh is obtained by applying the transformation functions on the mesh templates. *Non-conformities* are visualized by red circles.

Aside from memory and runtime aspects, ignoring symmetries during the mesh generation process potentially results in non-symmetric meshes (cf. Figure 1.4). If the simulation domain is symmetric, the boundary conditions can be transformed by the symmetry transformation. The mathematical solution to the simulation problem with transformed boundary conditions is equal to the transformed solution to the initial problem. However, this is not the case for a numerical solution if a non-symmetric mesh is used. It is therefore a central goal of this work to provide algorithms which generate high-quality meshes having the same symmetry as the corresponding simulation domain.

The ideas for the main approaches presented in this work originate from the field of computer graphics. In particular, the approach is motivated by a technique called geometry instancing [92], where a mesh template is used to render a high number of instances into a scene. Every instance has its own location and orientation within the scene and potentially additional differences, like scaling, color, or textures. In this work, the concepts of geometry instancing are applied to volumetric meshes used for simulations. The theories developed in this work are focused on similarities as they also are able to represent symmetries. For example, the *slice* of an object with a rotational symmetry is similar to all other slices of that object. A set of *mesh templates* each coupled with a set of transformations, called a *templated mesh*, is used to define a resulting mesh as visualized in Figure 1.5a. However, most algorithms for meshes used in computer graphics cannot be directly applied to volumetric meshes used in simulations. In computer graphics and rendering applications, surface meshes are used rather than volumetric meshes. These surface meshes are not required to be *conforming*, meaning that, e.g., the surface mesh is allowed to have holes or self-intersections. The conformity property, however, is an important requirement for meshes used in science and engineering as it is mandatory for discretization-based simulation methods like the FEM. This property is not automatically guaranteed when using templated meshes as can be seen on the right of Figure 1.5b. Therefore, algorithms for *templated mesh* generation and templated mesh adaptation have to take special care to ensure element conformity.

Theoretical approaches for templated meshes are developed in this work. These approaches are used to formulate new mesh generation and adaptation algorithms for their application with templated meshes. This work focuses on templated mesh generation algorithms, where two new general algorithms are proposed. Special (less complicated) mesh generation algorithms for geometries with symmetries are formulated as well. Additionally, data structures for storing templated meshes are developed, which optimize memory usage. The proposed data structures and mesh generation algorithms have been implemented in the open source meshing framework ViennaMesh [19]. Using these implementations, a benchmark-based survey is performed to evaluate improvements in memory usage and algorithm runtime as well as element quality and errors in simulation solutions. In particular, this work addresses the following research questions with a focus on challenges in micro- and nanoelectronics:

(i) How can templated meshes and templated geometries be defined? Are there any restrictions or issues? Which restrictions and which issues apply to objects with symmetries?

(ii) How can properties, like the Delaunay property, be abstracted to templated meshes?

(iii) Which algorithms for conventional meshes also work for templated meshes? Which modifications are required for these algorithms?

(iv) How can a templated mesh be generated based on symmetric geometries?

(v) What are the effects on mesh element quality of a templated mesh generation algorithm compared to a conventional mesh generation algorithm? How do non-symmetric meshes (of symmetric geometries) affect the solution of simulation processes?

(vi) How much memory can be saved when using templated meshes?

(vii) What are the effects on the runtime performance for templated mesh generation algorithms?

Chapter 2 introduces general definitions and theorems for geometries and meshes with a special focus on mesh properties which are important for discretization-based simulation methods, like the Delaunay property or mesh element quality. The necessary mathematical basics are covered in Appendix A.

Related work on meshes as well as algorithms for generating and adapting meshes is covered in Chapter 3. Additionally, the results of a literature search on automatic similarity and symmetry detection and symmetry-aware mesh processing is provided.

The theoretical background for templated structures, including definitions and theorems, is presented in Chapter 4. In this chapter, the major theoretical results of this work are deduced and discussed.

Two volumetric mesh generation algorithms for templated structures are presented in Chapter 5. Additionally, this chapter investigates a popular selection of mesh adaptation algorithms for their use with templated meshes and discusses restrictions and modifications.

In Chapter 6, decomposition methods for shapes, especially shapes with similarities and symmetries, are presented and special properties of objects with symmetries are discussed. Additionally, the theoretical results and the proposed algorithms presented in the previous chapters are applied to symmetric meshes and geometries.

A benchmark-based survey is presented in Chapter 7, which investigates the benefits of the data structures and algorithms proposed in this work.

The last chapter, Chapter 8, concludes the work by evaluating the research questions and gives an outlook on possible future work.

Chapter 2

Geometries and Meshes

Basic definitions and lemmas for geometries and meshes as well as proofs of lemmas essential to this work are presented in this chapter. Most definitions in this chapter are based on previous work [52, 127] but are extended to support non-simplex elements. The mathematical and geometrical fundamentals are covered in Appendix A.

2.1 Geometries

A simulation domain has to be specified by a geometry. To support discretization-based simulation methods, like the FEM, this geometry is required to have a Lipschitz $C^{0,1}$ boundary [59]. In this work the geometry space $\mathfrak{L}^n$ (cf. Definition A.25) is used to represent geometries.

Definition 2.1 (Geometry, linear geometry)**.** Let $\mathcal{G} \subseteq \mathbb{R}^n$. $\mathcal{G}$ is called a geometry, if there are sets $G_1, \ldots, G_m \in \mathfrak{L}^n$ which are connected and the geometry can be represented as a union of these sets: $\mathcal{G} = \bigcup_{i=1}^{m} G_i$. A geometry is linear, if there are polyhedrons $X_i, i = 1, \ldots, k$ with dimension of n and $\mathcal{G} = \bigcup_{i=1}^{k} X_i$.

A geometry is not required to be connected. However, there always is a finite partition of a geometry, which consists of connected sets. This is required for guaranteeing meshes with a finite number of mesh elements. A geometry $\mathcal{G}$ is called n-dimensional, if $\mathcal{G} \in \mathfrak{L}^n$ and if $\mathrm{DIM}(\mathcal{G}) = n$. If $\mathrm{DIM}(\mathcal{G}) = n - 1$, $\mathcal{G}$ is called an n-dimensional hull geometry.

It is beneficial to partition the simulation domain for many simulation scenarios. For example, different parts of a domain to be simulated are made of different materials. It is of advantage to reflect these parts, called regions, in the geometry as well. An example for a multi-region geometry is given in Figure 2.1, where the two materials of a hammer are represented by separate regions.

A simple approach for representing multi-region geometries is to add an additional region function $f : \mathbb{R}^n \to \mathbb{N}$, which maps a point of the geometry to its region. However, if a geometry has more than two regions, this approach might lead to regions which are not closed. Therefore, a multi-region geometry is defined using a function $\widetilde{\xi}$, which maps a point to the set of regions in which the point is located. For example, for all points of the interior of a region, the function $\widetilde{\xi}$ maps to just one region and for points, where multiple regions touch, it maps to all regions.

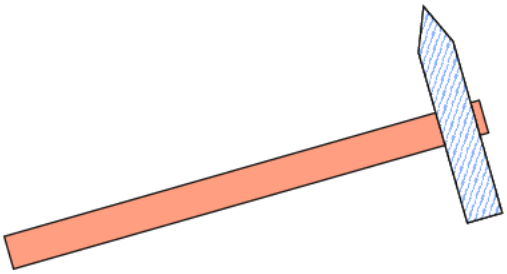


Figure 2.1: Multi-region geometry

The wooden shaft (visualized in red) and steel head (visualized in lined blue) are separate regions.

Definition 2.2 (Multi-region geometry, region, region interface)**.** Let $\mathcal{G} \subseteq \mathbb{R}^n$ be a geometry, $\widetilde{\xi}(x) : \mathcal{G} \to \mathcal{P}^\star(\{1, \ldots, m\})$ a function, and $\text{region}((\mathcal{G}, \widetilde{\xi}), i) := \{x \in \mathbb{R}^n, i \in \widetilde{\xi}(x)\}$. $(\mathcal{G}, \widetilde{\xi})$ is called a multi-region geometry, if

(i) every $\text{region}((\mathcal{G}, \widetilde{\xi}), i), i \in \mathbb{N}$ is a geometry with $\text{DIM}(\text{region}((\mathcal{G}, \widetilde{\xi}), i)) = \text{DIM}(\mathcal{G})$ and

(ii) $\{\text{region}((\mathcal{G}, \widetilde{\xi}), 1), \ldots, \text{region}((\mathcal{G}, \widetilde{\xi}), m)\}$ is a manifold partition of $\mathcal{G}$ (see Definition A.27)

The set $\text{region}((\mathcal{G}, \widetilde{\xi}), i)$ is called geometry regions and $\widetilde{\xi}$ is called geometry region indicator function. The number of regions of a multi-region geometry $(\mathcal{G}, \widetilde{\xi})$ is denoted as $\text{rc}((\mathcal{G}, \widetilde{\xi}))$. The interface of multiple different regions $\text{region}((\mathcal{G}, \widetilde{\xi}), i_1), \ldots .\text{region}((\mathcal{G}, \widetilde{\xi}), i_k), i_j \neq i_g, j \neq g$ is their intersection (which is also the intersection of their boundaries):

$$\text{interf}((\mathcal{G}, \widetilde{\xi}), i_1, \ldots, i_k) := \bigcap_{j=1}^{k} \text{region}((\mathcal{G}, \widetilde{\xi}), i_j) = \bigcap_{j=1}^{k} \text{bnd}^\star(\text{region}((\mathcal{G}, \widetilde{\xi}), i_j)) \tag{2.1}$$

The interface of geometry regions is also equal to all points which are in all regions.

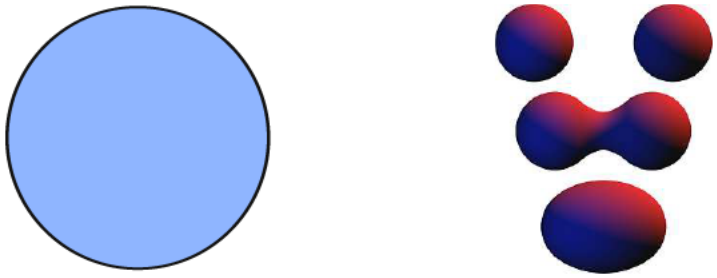
$$\text{interf}((\mathcal{G}, \widetilde{\xi}), i_1, \ldots, i_k) = \{x \in \mathbb{R}^n | \{i_1, \ldots, i_k\} \subseteq \widetilde{\xi}(x)\} \tag{2.2}$$

Every multi-region geometry with $\widetilde{\xi}(x)$ being a constant function can be identified with a geometry and vice versa.

To enable computational use of geometries, a representation is needed. There are three popular types of geometry representation:

- Implicit representation
- Boundary representation
- Constructive solid geometries (CSG)

Geometries of simulation domains in CAE are usually created using computer-aided design (CAD) tools and represented with boundary representation or constructive solid geometries. Data formats, like STEP [87] or IGES [46], are typically used to interface CAD applications with mesh generation software.



(a) Closed unit 2-disk **(b)** Metaballs[1]

Figure 2.2: Implicit geometries

2.1.1 Implicit Representation

Let $\mathcal{G}$ be a geometry. A function $F : \mathbb{R}^n \to \mathbb{R}$ with $\mathcal{G} = \{\boldsymbol{x} \in \mathbb{R}^n | F(\boldsymbol{x}) \geq 0\}$ and $\text{bnd}^\star(\mathcal{G}) = \{\boldsymbol{x} \in \mathbb{R}^n | F(\boldsymbol{x}) = 0\}$ is called the implicit representation of $\mathcal{G}$. The function used for this representation is not unique. However, there is a guarantee that for every geometry $\mathcal{G}$ there exists an implicit representation. The existence of such a function F for a given geometry $\mathcal{G}$ can easily be shown by setting

$$F(\boldsymbol{x}) := \begin{cases} \min_{\boldsymbol{y} \in \text{bnd}^\star(\mathcal{G})}(\|\boldsymbol{x} - \boldsymbol{y}\|_2) & \text{if } \boldsymbol{x} \in \mathcal{G} \\ -\min_{\boldsymbol{y} \in \text{bnd}^\star(\mathcal{G})}(\|\boldsymbol{x} - \boldsymbol{y}\|_2) & \text{if } \boldsymbol{x} \notin \mathcal{G} \end{cases} \tag{2.3}$$

For example, the closed n-ball with radius r and center $\mathbf{0}$ can be represented using the function $F(\boldsymbol{x}) = \|\boldsymbol{x}\|_2 - r$. Another popular example of objects represented by implicit functions are metaballs [70]. Metaballs are *organic-looking* ball-like objects fulfilling an implicit function which is based on distances to ball centers. These two examples of geometries represented with implicit functions are visualized in Figure 2.2. Implicit geometries can also be used to represent multi-region geometries with one implicit function F_i for each region. Although geometries represented by mathematical expressions are robust with respect to modeling, they rarely are used in engineering applications, because finding a function in closed form is challenging for general objects.

2.1.2 Boundary Representation and Piecewise Linear Complexes

A geometry $\mathcal{G} \subseteq \mathbb{R}^n$ with $\text{DIM}(\mathcal{G}) = n$ can also be represented by its boundary. Because a geometry has to be a bounded set (cf. Appendix A), it is always clear on which side of the boundary the *inside* of the geometry is located. The geometry $\mathcal{G}$ can be reconstructed from a boundary representation by using the convex hull of the boundary $\text{conv}(\text{bnd}(\mathcal{G}))$. $\text{conv}(\text{bnd}(\mathcal{G}))$ is then intersected with the boundary $\text{bnd}(\mathcal{G})$ to obtain a finite number of connected manifolds: $\mathcal{M} = \{M_1, \ldots, M_k\}$. Each of these connected manifolds $M_i \in \mathcal{M}$ is discarded, if the manifold M_i is *outside* of $\text{bnd}(\mathcal{G})$. Then, starting from the *outside*, the remaining manifolds in $\mathcal{M}$ are alternating either part of the geometry $\mathcal{G}$ or holes. This reconstruction process is visualized in Figure 2.3.

When representing multi-region geometries with boundaries, additional information is required, because the boundary of the geometry does not include information of the region interfaces. Therefore, each region has to be represented on its own with its boundary. Another approach is to unite all region boundaries and specify additional hole and seed points.

[1] Metaballs image by GlydeG (Own Work. Rendered with Bryce 6.) [Public domain], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3AMetaballs.png

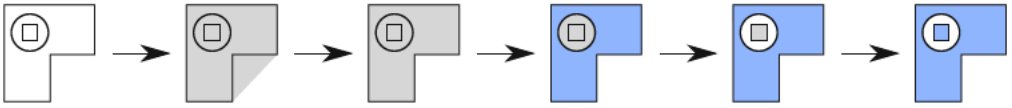


Figure 2.3: Reconstruction process of a boundary representation

The boundary is visualized on the left. At first, the convex hull is generated. Then, the boundary is used to slice the convex hull into a manifold partition. Manifold partition elements *outside* are discarded. Starting from *outside*, the remaining manifolds are alternating either part of the geometry or holes. Unassigned manifold partition elements are visualized gray, manifold partition elements assigned to the geometry are colored in blue.

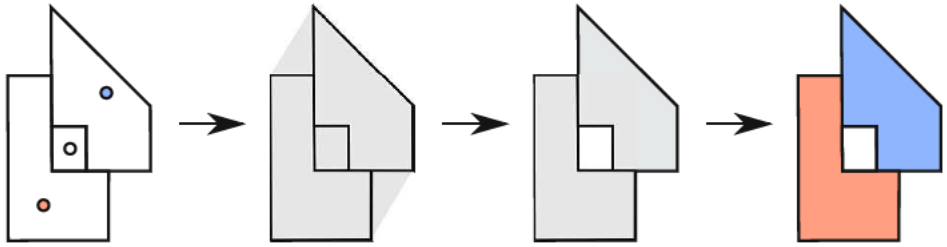


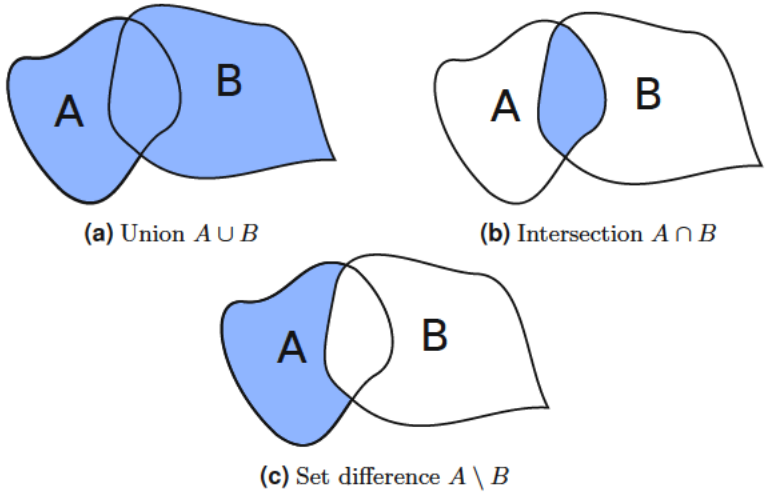
Figure 2.4: Reconstruction process of a multi-region boundary representation

The union of all region boundaries and the hole and seed points are visualized on the left, using colored circles to depict seed points and white circles for hole points. At first, the convex hull is generated. Then, the union of the region boundaries is used to slice the convex hull into a manifold partition. Manifold partition elements *outside* of the union of the region boundaries or, where a hole point is inside, are discarded. All other manifold partition elements are assigned to the corresponding region using the seed points.

Let $(\mathcal{G}, \widetilde{\xi})$ be a multi-region geometry, $B = \bigcup_{i=1}^{\mathrm{rc}((\mathcal{G},\widetilde{\xi}))} \mathrm{bnd}^{\star}(\mathrm{region}((\mathcal{G},\widetilde{\xi}), i))$ be the union of the region boundaries, $H = \{\boldsymbol{h}_1, \ldots \boldsymbol{h}_m\} \subseteq \mathbb{R}^n$ be the set of hole points, and $S_i = \{\boldsymbol{s}_{i,1}, \ldots, \boldsymbol{s}_{i,p_i}\} \subseteq \mathrm{region}((\mathcal{G},\widetilde{\xi}), i)$ be the set of seed points for each region: Then (B, H, S) is a boundary representation of $(\mathcal{G}, \widetilde{\xi})$ using hole and seed points. In general, B is not a manifold, because there does not exist a homeomorphism to $\mathbb{R}^n$ at points where a region interface and the geometry boundary meet. The reconstruction process for a multi-region geometry represented with its boundary and additional hole and seed points is similar to the algorithm above. The only difference is, that the hole points are used to discard additional manifolds and seed points are used to assign the remaining M_i to a region, as shown in Figure 2.4. The reconstruction for multi-region geometries with boundary representation is unique when using hole and seed points.

In practice, the boundary itself is often described using piecewise linear functions (for linear geometries) or splines [118]. Boundary representations are very popular in industrial and CAD applications. In particular, two important types of boundary representations are named: triangular hulls and piecewise linear complexes (PLC). Triangular hulls in $\mathbb{R}^n$ are commonly used in computer graphics, but are also popular in CAD applications, e.g., the STereoLithography (STL) format [93].

A PLC is an abstraction of an element complex with the property that the boundary of an element and the intersection of two distinct elements can be the union of other elements rather than a single element. A PLC uses linear k-cells as mesh elements.



(a) Union $A \cup B$ **(b)** Intersection $A \cap B$

(c) Set difference $A \setminus B$

Figure 2.5: CSG set operations of two sets

Definition 2.3 (Piecewise linear complex (PLC))**.** Let $\mathcal{P} \subseteq \mathfrak{L}^n$. $\mathcal{P}$ is a PLC if

(i) every element p of $\mathcal{P}$ is a linear k-cell, $k \leq n$,

(ii) the subset of all 0 and 1-dimensional elements is a face-complete (c.f. Definition A.10) element complex,

(iii) for every k-dimensional element $p \in \mathcal{P}$ there are $(k-1)$-dimensional elements $f_1, \ldots, f_j$ in $\mathcal{P}$ which cover up the boundary of p ($\text{bnd}^\star(p) = \bigcup_{i=1}^{j} f_i$), and

(iv) the intersection of two different elements $p_1, p_2 \in \mathcal{P}$ is either empty or a finite union of elements of $\mathcal{P}$ with a dimension smaller or equal to the minimum dimension of p_1 and p_2.

The underlying space of a PLC is a linear geometry. Therefore, a PLC represents a linear geometry and every polyhedron can be represented by a PLC. A PLC $\mathcal{P}$ is often represented with its boundary facets and additional hole and seed points to support multiple regions and holes. Additionally, PLCs play an important role in mesh generation algorithms based on Delaunay refinement (cf. Section 2.3) as a lot of mathematical statements for meshes have been proven for PLCs [127].

2.1.3 Constructive Solid Geometries

Constructive solid geometry (CSG) is a geometry representation, where basic shapes, like cubes or spheres, are used together with set operations to represent a geometry [38]. Commonly supported set operations are set intersection, set union, and set difference as visualized in Figure 2.5. The basic shapes and the set operations are used to form a hierarchical CSG tree. An example of a CSG tree for a geometry is shown in Figure 2.6. CSG representations also support multi-region geometries, if a CSG tree is created for every region.

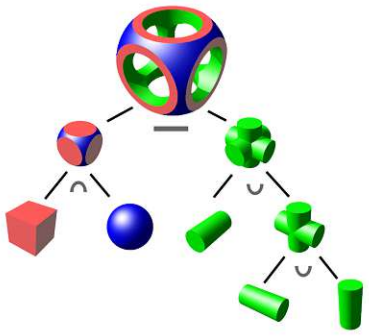


Figure 2.6: CSG tree[1]

A cube, a sphere, and a cylinder are used as basic shapes and are combined with union, intersection, and set difference operations.

2.2 Meshes

A mesh is defined as a finite, face-complete element complex without any *dangling* elements.

Definition 2.4 (Mesh)**.** Let $\mathcal{M} \subseteq \mathcal{E}$ be subset of an element complex $\mathcal{E}$. $\mathcal{M}$ is called a mesh, if

(i) $\mathcal{M}$ is finite,

(ii) $\mathcal{M}$ is face-complete (in $\mathcal{E}$), and

(iii) $\mathcal{M}$ has no dangling elements, meaning that every element $E \in \mathcal{M}$ is either a cell or E has at least one co-face cell (cf. Definition A.11).

Requirement (iii) ensures that cell elements are the only elements, which are not faces of any other element (except the cell itself). A mesh $\mathcal{M}$ is called n-dimensional, if $\mathcal{M} \in \mathcal{P}(\mathbb{R}^n)$ and if $\mathrm{DIM}_{\mathrm{cell}}(\mathcal{M}) = n$. If $\mathrm{DIM}_{\mathrm{cell}}(\mathcal{M}) = n - 1$, $\mathcal{M}$ is called an n-dimensional hull mesh or n-dimensional surface mesh. A mesh $\mathcal{M}$ with only simplex elements is called a simplex mesh, a two-dimensional (2D) mesh $\mathcal{M}$ with all cells being quadrilaterals is called an all-quad mesh, and a three-dimensional (3D) mesh $\mathcal{M}$ with all cells being hexahedrons is called an all-hex mesh.

Lemma 2.1 (Intersection of meshes)**.** *The intersection of two meshes $\mathcal{M}_1$ and $\mathcal{M}_2$ is finite, face-complete, and conforming. If the union of both meshes $\mathcal{M}_1 \cup \mathcal{M}_2$ is conforming, then the underlying space of the intersection is also the intersection of the underlying spaces:* $\mathrm{us}(\mathcal{M}_1 \cap \mathcal{M}_2) = \mathrm{us}(\mathcal{M}_1) \cap \mathrm{us}(\mathcal{M}_2)$.

Proof.

(i) To prove: $|\mathcal{M}_1 \cap \mathcal{M}_2| < \infty$:
$\mathcal{M}_1 \cap \mathcal{M}_2$ is finite, because $\mathcal{M}_1$ and $\mathcal{M}_2$ are meshes and therefore finite.

(ii) To prove: $\mathcal{M}_1 \cap \mathcal{M}_2$ is face-complete:
Every element $E \in \mathcal{M}_1 \cap \mathcal{M}_2$ is also an element of $\mathcal{M}_1$ and $\mathcal{M}_2$. Because $\mathcal{M}_1$ and $\mathcal{M}_2$ are meshes, they are both face-complete and $\mathrm{faces}(E)$ is a subset of $\mathcal{M}_1$ and $\mathcal{M}_2$. Therefore, $\mathrm{faces}(E)$ is also a subset of $\mathcal{M}_1 \cap \mathcal{M}_2$ and $\mathcal{M}_1 \cap \mathcal{M}_2$ is face-complete.

[1] Image by Zottie (Own work) [GFDL or CC-BY-SA-3.0], via Wikimedia Commons, https://commons.wikimedia.org/wiki/File%3ACsg_tree.png

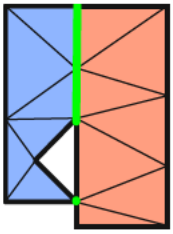


Figure 2.7: Mesh intersection

The intersection of these two meshes (colored in green), being two edges and an isolated vertex, is not a mesh, because the isolated vertex is a dangling element.

(iii) To prove: $\mathcal{M}_1 \cap \mathcal{M}_2$ is conforming:
Let E_1, E_2 be two elements of $\mathcal{M}_1 \cap \mathcal{M}_2$ with non-empty intersection. Both elements are also in $\mathcal{M}_1$. Because $\mathcal{M}_1$ is a conforming mesh, $E_1 \cap E_2$ is a face of both E_1 and E_2. Therefore, $\mathcal{M}_1 \cap \mathcal{M}_2$ is conforming.

(iv) To prove: $\mathcal{M}_1 \cup \mathcal{M}_2$ is conforming $\Rightarrow \text{us}(\mathcal{M}_1 \cap \mathcal{M}_2) = \text{us}(\mathcal{M}_1) \cap \text{us}(\mathcal{M}_2)$:
For every point $x \in \text{us}(\mathcal{M}_1 \cap \mathcal{M}_2)$ there is an element $E \in \mathcal{M}_1 \cap \mathcal{M}_2$, which contains x. E is an element of both meshes $\mathcal{M}_1$ and $\mathcal{M}_2$ and x is included in the underlying spaces of $\mathcal{M}_1$ and $\mathcal{M}_2$. Therefore, x is also included in the intersection of the underlying spaces.

For every x which is in $\text{us}(\mathcal{M}_1)$ and in $\text{us}(\mathcal{M}_2)$, there are elements $E_1 \in \mathcal{M}_1$ and $E_1 \in \mathcal{M}_2$, which both contain x. Because $\mathcal{M}_1 \cup \mathcal{M}_2$ is conforming, the intersection $E_1 \cap E_2$ (which also contains x) is a face of both E_1 and E_2. Therefore, $E_1 \cap E_2$ is included in both meshes $\mathcal{M}_1$ and $\mathcal{M}_2$ and also in their intersection $\mathcal{M}_1 \cap \mathcal{M}_2$. Ultimately, the set $E_1 \cap E_2$, which contains x, is a subset of the underlying space of the mesh intersection.

Therefore, $\text{us}(\mathcal{M}_1 \cap \mathcal{M}_2)$ is a subset of $\text{us}(\mathcal{M}_1) \cap \text{us}(\mathcal{M}_2)$ and $\text{us}(\mathcal{M}_1) \cap \text{us}(\mathcal{M}_2)$ is a subset of $\text{us}(\mathcal{M}_1 \cap \mathcal{M}_2)$, which make these two sets equal.

□

In general, however, the intersection of two meshes is not a mesh because it may contain dangling elements as shown in Figure 2.7. The second part of this Lemma states, that if the union of two meshes is conforming (i.e. the union is a mesh), then their intersection covers the same space as the intersection of the geometries of the meshes. This is not true for the intersection of two meshes, where their union is not conforming. For example, let $\mathcal{M}_1$ be a mesh which contains a line and its boundary vertices and $\mathcal{M}_2$ be a mesh which solely contains the vertex located at the center of the line. Then, $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$ but the intersection of $\text{us}(\mathcal{M}_1)$ and $\text{us}(\mathcal{M}_2)$ is the middle point of the line. The intersection of meshes will play an important role for the boundary patch partition in Section 4.3.

Similar to manifold partitions (cf. Appendix A), meshes can also be partitioned using the following approach:

Definition 2.5 (Mesh partition). Let $\mathcal{M}$ be a mesh and $M = \{M_1, \ldots, M_k\}, M_i \subseteq \mathcal{M}$. M is called a mesh partition of $\mathcal{M}$, if

(i) every element M_i is a non-empty mesh,

(ii) $\mathcal{M}$ is the union of all M_i: $\mathcal{M} = \bigcup_{A \in M} A$, and

(iii) the cell dimension of the intersection of two different elements $A, B \in M$ is less than the maximum cell dimension of A and B.

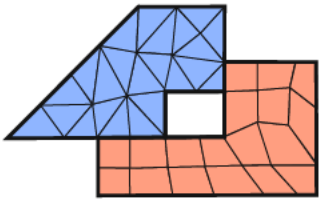


Figure 2.8: 2D multi-region mesh

A mesh partition does not necessarily consist of meshes with the same dimension. For example, a mesh containing one triangle (and all its faces) can have a mesh partition with the triangle (and all its faces) and all triangle edges (and all of their vertices). However, any cell of $\mathcal{M}$ is in exactly one element of a mesh partition of $\mathcal{M}$. Therefore, the underlying space of $\mathcal{M}$ is equal to the union of the underlying spaces of all elements of the partition.
Similar to geometries, meshes are also defined with multi-region support.

Definition 2.6 (Multi-region mesh)**.** Let $\mathcal{M}$ be a mesh in $\mathbb{R}^n$, $\xi : \text{cells}(\mathcal{M}) \to \{\{1\}, \ldots, \{m\}\}$ be a function which naturally extends to all $E \in \mathcal{M}$ as $\xi(E) := \bigcup_{F \in C} \xi(F)$, and $\text{region}((\mathcal{M}, \xi), i) := \{E \in \mathcal{M} | i \in \xi(E)\}$ with C being all cells where E is a face: $C := \text{cofaces}_{\text{DIM}_{\text{cell}}(\mathcal{M})}(E)$. $(\mathcal{M}, \xi)$ is called a multi-region mesh, $\text{region}((\mathcal{M}, \xi), i)$ are called the mesh regions, and ξ is called mesh region indicator function. The number of regions of a multi-region mesh $(\mathcal{M}, \xi)$ is denoted as $\text{rc}((\mathcal{M}, \xi))$.

For a multi-region mesh $(\mathcal{M}, \xi)$, $E \in (\mathcal{M}, \xi)$ is also written for $E \in \mathcal{M}$. An example of a 2D multi-region mesh is shown in Figure 2.8.

Lemma 2.2 (Region is a mesh)**.** *For a multi-region mesh* $(\mathcal{M}, \xi)$*, every region* $\text{region}((\mathcal{M}, \xi), i)$ *is a mesh.*

Proof.

(i) $|\text{region}((\mathcal{M}, \xi), i)|$ is finite:
$\text{region}((\mathcal{M}, \xi), i)$ is a subset of the finite mesh $(\mathcal{M}, \xi)$ and therefore also finite.

(ii) $\text{region}((\mathcal{M}, \xi), i)$ is face-complete:
Every element $E \in \text{region}((\mathcal{M}, \xi), i)$ is, by definition, also in $\mathcal{M}$. The faces of E are included in $\mathcal{M}$, because $\mathcal{M}$ is face-complete. From the definition of ξ follows, that for every faces F of E, $\xi(F)$ contains i. Therefore, every face of E is also in $\text{region}((\mathcal{M}, \xi), i)$ and $\text{region}((\mathcal{M}, \xi), i)$ is face-complete.

(iii) $\text{region}((\mathcal{M}, \xi), i)$ is conforming:
Let E_1, E_2 be two elements of $\text{region}((\mathcal{M}, \xi), i)$ with non-empty intersection. Then, E_1 and E_2 are also in $\mathcal{M}$. Because of the conformity of $\mathcal{M}$, $E_1 \cap E_2$ is a face of both E_1 and E_2 and because $\text{region}((\mathcal{M}, \xi), i)$ is face-complete, $E_1 \cap E_2$ is an element of $\text{region}((\mathcal{M}, \xi), i)$. Therefore, $\text{region}((\mathcal{M}, \xi), i)$ is conforming.

(iv) $\text{region}((\mathcal{M}, \xi), i)$ has no dangling elements (except cells):
From the definition of ξ follows, that for every non-cell element E of $\text{region}((\mathcal{M}, \xi), i)$ there is a co-face cell C of E in $\text{region}((\mathcal{M}, \xi), i)$. Therefore, $\text{region}((\mathcal{M}, \xi), i)$ has no dangling elements.

□

Every mesh $\mathcal{M}$ is also a multi-region mesh $(\mathcal{M}, \xi)$, where the mesh region indicator function ξ maps every element to $\{1\}$. On the other hand, a multi-region mesh $(\mathcal{M}, \xi)$, with a mesh region indicator function ξ mapping all elements to $\{1\}$, can be identified by the mesh $\mathcal{M}$. Therefore, all statements for multi-region meshes also hold for meshes.

Definition 2.7 (Boundary of a mesh, region interface)**.** The boundary of a mesh $\mathcal{M}$ is defined as all elements which are also in the boundary of its underlying space: $\text{bnd}(\mathcal{M}) := \{E \in \mathcal{M} | E \subseteq \text{bnd}^\star(\text{us}(\mathcal{M}))\}$. The interface of multiple mesh regions

$$\text{region}((\mathcal{M}, \xi), i_1), \ldots, \text{region}((\mathcal{M}, \xi), i_k), i_j \neq i_g, j \neq g \tag{2.4}$$

is defined as all elements which are in all specified regions:

$$\text{interf}((\mathcal{M}, \xi), i_1, \ldots, i_k) := \bigcap_{j=1}^{k} \text{region}((\mathcal{M}, \xi), i_j) \tag{2.5}$$

Similar to geometry interfaces, the interface of mesh regions is equal to:

$$\text{interf}((\mathcal{M}, \xi), i_1, \ldots, i_k) = \{E \in \mathcal{M}, \forall x \in E : \{i_1, \ldots, i_k\} \subseteq \xi(x)\} \tag{2.6}$$

The boundary $\text{bnd}(\mathcal{M})$ of a mesh $\mathcal{M}$ is again a mesh with $\text{DIM}_{\text{cell}}(\text{bnd}^\star(\mathcal{M})) = \text{DIM}_{\text{cell}}(\mathcal{M}) - 1$. Because a mesh region is also a mesh, the definition of the boundary of a mesh region is the same as the boundary of a mesh.

A *triangulation* of a finite point set $S \subseteq \mathbb{R}^n$ is a special type of mesh which is important for constructing high quality meshes. Additionally, proofs for quality-guarantees of mesh generation algorithms usually require a triangulation [127].

Definition 2.8 (Triangulation)**.** Given a point set $S = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\} \subseteq \mathbb{R}^n$, a simplex mesh $\mathcal{M}$ is called a triangulation of S, if $\text{us}(\mathcal{M}) = \text{conv}(S)$ and $\text{elem}_0(\mathcal{M}) = S$.

2.2.1 Geometry-Conformity

In simulation scenarios, a geometry for the simulation domain is given, for which a mesh has to be generated. In this section, terms are defined to identify, if a certain mesh is *the mesh of a geometry*. At first, the term *respect* is defined for elements and sets of elements, which are *compatible*.

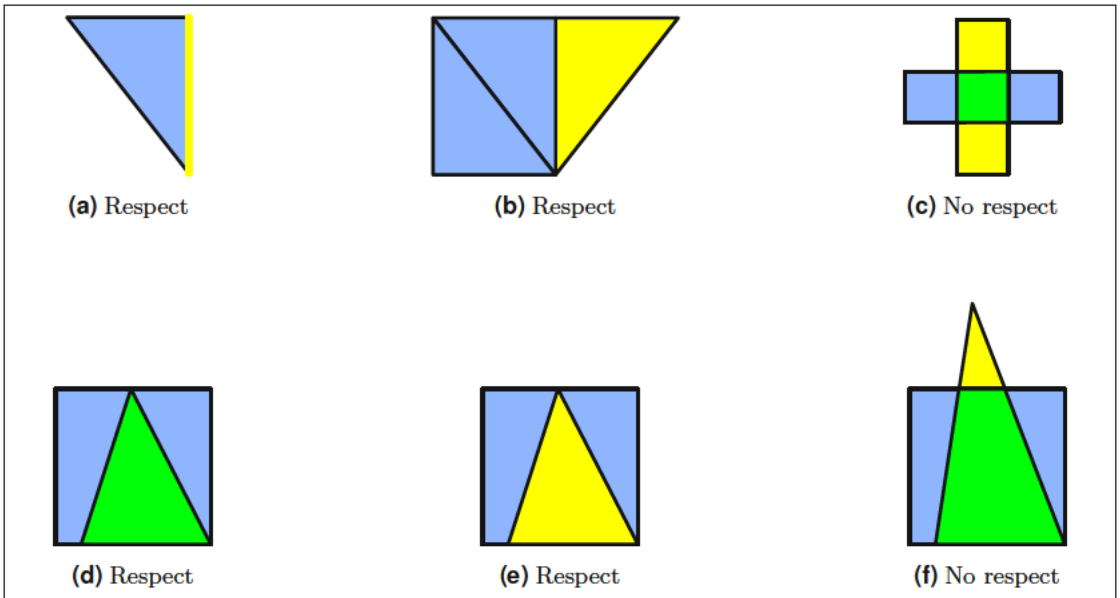
Definition 2.9 (Respect)**.** Let $E, X \in \mathfrak{L}^n$ and $\mathcal{E}, \mathcal{M} \subseteq \mathfrak{L}^n$ be two element spaces. E is said to respect X, if $E \cap S$ is either empty or can be represented by a finite union of faces of E:

$$E \cap X \neq \emptyset \Rightarrow \exists F_1, \ldots, F_k \in \text{faces}(E) : E \cap X = \bigcup_{i=1}^{k} F_i \tag{2.7}$$

E is said to respect $\mathcal{E}$, if for all $Y \in \mathcal{E}$, E respects Y and $\mathcal{M}$ is said to respect $\mathcal{E}$, if every element M in $\mathcal{M}$ respects $\mathcal{E}$.

In words, a set E respects another set X, if the intersection $E \cap X$ can be represented by a union of faces of E. Figure 2.9 visualizes the respect property for sets and set spaces. The following holds for the respect property:

Lemma 2.3 (Respectness of intersection)**.** *Let $E, A, B \in \mathfrak{L}^n$ be sets, where E respects A and E also respects B. Then, E respects $A \cap B$.*



(a) Respect **(b)** Respect **(c)** No respect

(d) Respect **(e)** Respect **(f)** No respect

Figure 2.9: *Respect* property

Elements are drawn in either blue or yellow, overlapping sections are colored green. In (a), the yellow line respects the blue triangle and vice versa, because the intersection of the triangle and the line is the line which is a face of both. In (d), the triangle, which is overlapped by a quadrilateral, respects the blue quadrilateral but the quadrilateral does not respect the triangle because the intersection of these two elements is the triangle which in turn is a face of the triangle but not of the quadrilateral. In (b) and (e), the yellow triangle respects the blue mesh, because it respects every mesh element. The intersection of the two quadrilaterals in (c), visualized in green, is not a face of any quadrilateral and therefore they do not respect each other. Similarly, in (f), the triangle and the quadrilateral do not respect each other, because the green intersection is not a face of any of these two.

Proof. If either $E \cap A$, $E \cap B$, or $A \cap B$ is empty, the statement is trivially true. Let $F_1, \ldots, F_k \in \text{faces}(E)$ be the sets for representing $E \cap A = \bigcup_{i=1}^{k} F_i$ and $G_1, \ldots, G_m$ be the sets for representing $E \cap B = \bigcup_{i=1}^{m} G_i$. Then,

$$E \cap (A \cap B) = (E \cap A) \cap (E \cap B) = \left(\bigcup_{i=1}^{k} F_i\right) \cap \left(\bigcup_{i=1}^{m} G_i\right) = \bigcup_{i=1}^{k} \bigcup_{j=1}^{m} F_i \cap G_j \tag{2.8}$$

Because F_i and G_j are both faces of E, $F_i \cap G_j$ is also a face of E. Therefore, E respects $A \cap B$. □

A mesh and therefore all of its elements must respect its geometry. Additionally, the mesh has to cover up the geometry.

Definition 2.10 (Geometry-conformity)**.** A mesh $\mathcal{M}$ geometry-conforms to a geometry $\mathcal{G}$, if $\text{us}(M) = \mathcal{G}$. A multi-region mesh $(\mathcal{M}, \xi)$ geometry-conforms to a multi-region geometry $(\mathcal{G}, \widetilde{\xi})$, if $\text{us}(\mathcal{M}) = \mathcal{G}$ and if $\forall i = 0, \ldots, \text{rc}(\mathcal{G}) : \text{us}(\text{region}((\mathcal{M}, \xi), i)) = \text{region}((\mathcal{G}, \widetilde{\xi}), i)$. $\text{region}((\mathcal{M}, \xi), i)$ is called the mesh region of the geometry region $\text{region}((\mathcal{G}, \widetilde{\xi}), i)$.

It can be shown, that if a mesh geometry-conforms to a geometry, it also respects the geometry.

Lemma 2.4 (Respectness of meshes with geometry-conforming property)**.** *Let $\mathcal{G}$ be a geometry, $\mathcal{M}$ be a mesh which geometry-conforms to $\mathcal{G}$, $(\mathcal{G}, \widetilde{\xi})$ be a multi-region geometry, and $(\mathcal{M}, \xi)$ be a multi-region mesh which geometry-conforms to $(\mathcal{G}, \widetilde{\xi})$. Then the following holds:*

(i) Every element E of the mesh $\mathcal{M}$ respects the geometry $\mathcal{G}$.

(ii) Every element E of a mesh region $\mathrm{region}((\mathcal{M}, \xi), i)$ *respects the corresponding geometry region* $\mathrm{region}((\mathcal{G}, \widetilde{\xi}), i)$.

Proof. Every mesh element $E \in \mathcal{M}$ is a subset of the underlying space of the mesh. Consequently, E is also a subset of the geometry $\mathcal{G}$ and $E \cap \mathcal{G} = E$. Because every element is a face of itself, E respects $\mathcal{G}$. The second statement follows from the first. □

The term geometry-conforming of a mesh and a geometry is not related to the mesh space property conforming.

In general, a mesh $(\mathcal{M}, \xi)$, which geometry-conforms to a geometry $\mathcal{G}$, is not unique. It is not even guaranteed that there is a mesh in some given n-manifold complex, which geometry-conforms to a given geometry. For example, there is no simplex mesh which geometry-conforms to the unit three-ball. In these cases, a mesh can only approximate a given geometry. However, for each mesh, there is a unique geometry, to which the mesh is geometry-conforming to.

Definition 2.11 (Geometry of a mesh)**.** Let $(\mathcal{M}, \xi)$ be a multi-region mesh. The structure

$$\mathrm{geo}((\mathcal{M}, \xi)) = \left(\mathrm{us}(\mathcal{M}), \widetilde{\xi}(x) = \bigcup_{E \in \mathcal{M} : x \in E} \xi(E) \right) \tag{2.9}$$

is called the geometry of the multi-region mesh $(\mathcal{M}, \xi)$.

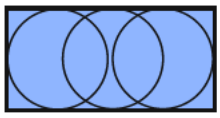
By definition, it is assured that the geometry of a multi-region mesh $(\mathcal{M}, \xi)$ is a geometry and $(\mathcal{M}, \xi)$ is geometry-conforming to the geometry $\mathrm{geo}((\mathcal{M}, \xi))$.

2.2.2 Mesh Element Quality and Mesh Quality

For many boundary values problems, the shape of the mesh elements directly affects the numerical convergence and accuracy of the solution [54][105]. A mesh element is said to have a *good* quality, if these effects are beneficial. However, it mainly depends on the application, the boundary value problem, and the element type, which shapes of an element are considered to be *good*. There are many different methods on how to measure the quality of mesh elements [66][131]. The following geometric properties are needed for most of these measures.

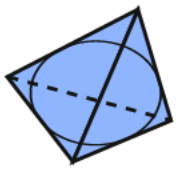
Definition 2.12 (Shortest edge, longest edge)**.** Let E be a mesh element. $l_{\min}(E)$ is defined as the smallest face edge of E, $l_{\max}(E)$ is defined as the largest face edge of E.

Definition 2.13 (Volume, relative volume)**.** Let μ be the Lebesgue measure and $\mathcal{H}^k$ be the k-dimensional Hausdorff measure [77]. The volume of E for $E \subseteq \mathbb{R}^n$ is defined as $\mathrm{Vol}(E) := \mu(E)$ and the relative volume of E is defined as $\mathrm{Vol}^\star(E) := \mathcal{H}^{\mathrm{DIM}(E)}(E)$.



(a) Inballs of a rectangle

(b) Inball of a triangle



(c) Inball of a tetrahedron

Figure 2.10: Inballs for a rectangle, for a triangle, and for a tetrahedron

Note, that the inball for the rectangle is not unique.

Definition 2.14 (n-circumball, circumradius)**.** Let $S \subseteq \mathbb{R}^n$ be a simplex. An n-ball, the boundary of which passes through all vertices of S, is called n-circumball of S. The (unique) n-circumball of an n-simplex S is denoted as $\overline{\mathcal{B}}^n(S)$ and the circumradius $R_{\text{circ}}(S)$ is defined as the radius of the smallest n-circumball of S.

Definition 2.15 (Min-containment ball, inradius, inball)**.** Let E be an element in $\mathbb{R}^n$, the min-containment ball of E is defined as the smallest n-ball $\overline{\mathcal{B}}^n_r(\boldsymbol{x}) \supseteq E$ and the min-containment radius $R_{\text{min-cont}}(E)$ is defined as the radius of the min-containment ball of E. The inradius of E is defined as: $R_{\text{in}}(E) := \max_{\overline{\mathcal{B}}^n_r(\boldsymbol{x}) \subseteq E}\{r\}$. A n-ball $\overline{\mathcal{B}}^n_r(\boldsymbol{x})$ with $r = R_{\text{in}}(E)$ and $\overline{\mathcal{B}}^n_r(\boldsymbol{x}) \subseteq E$ is called an inball.

While there is one unique min-containment ball for every mesh element, there might be more than one inball for certain elements. For example, a non-quadratic rectangle has infinitely many inballs. However, the inball of a simplex is unique. Inballs for a quadrilateral, a triangle, and a tetrahedral are visualized in Figure 2.10.

Element measures are mappings which assign a quality value to each cell of a mesh. When using the FEM, angles near 180° cause interpolation errors which further lead to discretization errors. On the other hand, angles near 0 lead to ill-conditioned stiffness-matrices [66].

Definition 2.16 (Smallest/largest angle, smallest/largest dihedral angle)**.** Let E be an element with $\text{DIM}(E) = 2$, $\angle_{\min}(E)$ and $\angle_{\max}(E)$ is defined as the smallest and largest interior angle of the element, respectively. The smallest angle of an element E with $\text{DIM}(E) = 3$ is defined using all interior angles measured in 2D hyperplanes which are orthogonal to a face line. $\angle_{\max}(E)$ is similarly defined as the largest interior angle.

The smallest and largest angles of a tetrahedron are equal to the smallest and largest dihedral angles, respectively. For lines with non-planar co-facets (cf. Definition A.11), e.g., the lines of a quadrilateral, all interior angles along the line have to be taken into account when finding the smallest angle.

Definition 2.17 (Radius-edge ratio)**.** Let $S \subseteq \mathbb{R}^n$ be a simplex, the radius-edge ratio of S is defined as $\phi_{\text{radius-edge}}(S) := R_{\text{circ}}(E)/l_{\min}(E)$.

If the spatial dimension n is two, the minimal angle and the radius-edge ratio are related: $\phi_{\text{radius-edge}}(S) = 1/(2\sin(\angle_{\min}(S)))$. However, the radius-edge ratio is a flawed measure for tetrahedrons in $\mathbb{R}^3$, because slivers, i.e., tetrahedrons which are not considered to be *good* elements, can have radius-edge ratios as small as $1/\sqrt{2}$. Other measures like the radius ratio or the numerically more robust volume-length measure, offer improved quality measurements for mesh elements in $\mathbb{R}^3$.

Definition 2.18 (Radius ratio)**.** Let E be a simplex mesh element in $\mathbb{R}^n$, the radius ratio is defined as $\phi_{\text{radius}}(E) := R_{\text{in}}(E)/R_{\text{circ}}(E)$. The following definition is used for non-simplex mesh elements: $\phi_{\text{radius}}(E) := R_{\text{in}}(E)/R_{\text{min-cont}}(E)$.

Definition 2.19 (Volume-length measure)**.** Let E be a mesh element in $\mathbb{R}^n$, the volume-length is defined as

$$\phi_{\text{volume-length}}(E) := \frac{\text{Vol}^\star(E)}{\sqrt{\sum_{F \in \text{faces}_2(E)} \text{Vol}^\star(F)^2}} \tag{2.10}$$

Based on an element quality measure, a quality *score* can be calculated. The only difference between a quality measure and a quality *score* is, that elements with *good* quality have a high score. The fundamental element-level quality score enables to evaluate an entire multi-region mesh quality.

Definition 2.20 (Quality vector)**.** Let $(\mathcal{M}, \xi)$ be a multi-region mesh and Φ be an element quality measure. The quality vector $\Psi_\Phi(\mathcal{M})$ is defined as the sorted tuple of quality scores of all cells $C \in \text{cells}(\mathcal{M})$.

The quality of two meshes can now be compared by using a lexicographically ordering on the quality vectors. A mesh $\mathcal{M}_1$ has a better quality than a mesh $\mathcal{M}_2$, if $\Psi_\Phi(\mathcal{M}_1) > \Psi_\Phi(\mathcal{M}_2)$. Creating the entire quality vector of a mesh is too time-consuming. However, identifying the k worst elements is usually sufficient [75]. Nevertheless, for theoretical analysis, the quality vector with the lexicographical ordering still plays an important role.

2.3 Delaunay Elements and Meshes

Another very powerful element property for simplex elements is the Delaunay property [31].

Definition 2.21 (Delaunay, strongly Delaunay, locally Delaunay)**.** Let $\mathcal{M}$ be a simplex mesh in $\mathbb{R}^n$. An element $S \in \mathcal{M}$ is said to be Delaunay in $\mathcal{M}$, if S has an open circumball $\mathcal{B}_r^n(\boldsymbol{x})$ which does not include any vertices of $\mathcal{M}$. An element $S \in \mathcal{M}$ is said to be strongly Delaunay in $\mathcal{M}$, if S has a closed circumball $\overline{\mathcal{B}}_r^n(\boldsymbol{x})$ which does not include any vertices of $\mathcal{M}$. A simplex mesh $\mathcal{M}$ is called Delaunay, if all cells are Delaunay.

A facet $F \in \mathcal{M}$ is said to be locally Delaunay in $\mathcal{M}$, if S has an open circumball $\mathcal{B}_r^n(\boldsymbol{x})$ which does not include any vertices of any co-face cells.

The Delaunay property is visualized in Figure 2.11. While the Delaunay property is a global property (all mesh vertices have to be checked), *locally Delaunay* only requires the evaluation of boundary vertices of neighbor simplices. Locally Delaunay is a weaker property for simplex elements than Delaunay, but if all facets of a mesh are locally Delaunay, the cells are in fact Delaunay.

Lemma 2.5 (Delaunay Lemma)**.** *Let $\mathcal{M}$ be a simplex mesh in $\mathbb{R}^n$ with $\text{DIM}_{\text{cell}}(\mathcal{M}) = n$, the following statements are equivalent:*

- *$\mathcal{M}$ is Delaunay.*
- *Every facet of $\mathcal{M}$ is Delaunay in $\mathcal{M}$.*
- *Every facet of $\mathcal{M}$ is locally Delaunay in $\mathcal{M}$.*

(a) Triangle which is Delaunay

(b) Triangle which is not Delaunay

Figure 2.11: Delaunay property

In (a), a triangle (colored in blue) which is Delaunay in the mesh is shown. (b) visualizes a triangle (colored in blue) which is not Delaunay due to the red vertex which is inside the triangle's circumball and therefore breaks the Delaunay property of the triangle.

A proof for this Lemma can be found in [127]. The Delaunay Lemma guarantees that the locally Delaunay property of facets, which is easier and more efficient to evaluate, results in the Delaunay property for cells. For each finite point set $S \subseteq \mathbb{R}^n$, there always exists a triangulation of S, which is Delaunay.

Delaunay is a very strong property for simplex meshes. For a simplex mesh $\mathcal{M}$ which is Delaunay, the dual is equal to the Voronoi diagram of $\mathcal{M}$. Some discretization-based simulation methods, like the FVM, utilize the Voronoi diagram during the discretization process [127]. For a given point set $S = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\} \subseteq \mathbb{R}^2$, a Delaunay triangulation $\mathcal{M}$ of S maximizes the minimum angle and minimizes the largest circumball [127]. Unfortunately, this is not true for Delaunay triangulations in $\mathbb{R}^3$. To improve the quality of these meshes, post-processing, like mesh adaptation, is required. Additionally, a Delaunay triangulation minimizes the largest min-containment ball for arbitrary dimensions n [136].

For a mesh $\mathcal{M}$ which is generated based on a PLC $\mathcal{P}$, a weaker Delaunay definition is of advantage.

Definition 2.22 (Visibility)**.** Two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ are visible to each other, if the line simplex$(\boldsymbol{x}, \boldsymbol{y})$ (cf. Definition A.17) respects the PLC $\mathcal{P}$.

Definition 2.23 (Constrained Delaunay)**.** Let $\mathcal{P}$ be a PLC and $(\mathcal{M}, \xi)$ be a multi-region simplex mesh which respects and geometry-conforms to $\mathcal{P}$. A simplex $S \in (\mathcal{M}, \xi)$ is called constrained Delaunay, if all vertices of S are also vertices of the PLC $\mathcal{P}$ and if S has an open circumball $\mathcal{B}_r^n(\boldsymbol{x})$ which only includes mesh vertices of $\mathcal{M}$, which are not visible from any point of the interior of S. The mesh $(\mathcal{M}, \xi)$ is constrained Delaunay, if every cell is constrained Delaunay.

A simplex which is constrained Delaunay, therefore is Delaunay except for vertices where the visibility is blocked by the PLC, e.g., on internal interfaces. An example visualizing the constrained Delaunay property is given in Figure 2.12. A Lemma similar to the Delaunay Lemma can be formulated for the constrained Delaunay property.

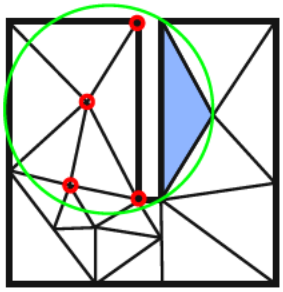


Figure 2.12: Constrained Delaunay property

The blue triangle certainly is not Delaunay, because there are four vertices (indicated by red circles) which lie within the triangle's circumball. However, the triangle is constrained Delaunay, because the PLC boundary blocks the visibility of these vertices to the triangle.

Lemma 2.6 (Constrained Delaunay Lemma). *Let $\mathcal{M}$ be a simplex mesh in $\mathbb{R}^n$ with $\mathrm{DIM}_{\mathrm{cell}}(\mathcal{M}) = n$. Then, the following statements are equivalent.*

- *$\mathcal{M}$ is constrained Delaunay.*
- *Every facet of $\mathcal{M}$ which is not included in any facet of P is constrained Delaunay in $\mathcal{M}$.*
- *Every facet of $\mathcal{M}$ which is not included in any facet of P is locally Delaunay in $\mathcal{M}$.*

As for the Delaunay Lemma, a proof for this Lemma is given in [127]. A multi-region mesh $(\mathcal{M}, \xi)$ is Delaunay, if all facets included in any PLC facet are locally Delaunay. For some PLCs it can be shown that there exists a triangulation which is constrained Delaunay.

Definition 2.24 (Edge-protected). An element space $\mathcal{E}$ is called edge-protected, if every line $L \in \mathrm{elem}_1(\mathcal{E})$ (cf. Definition A.7) is strongly Delaunay.

For PLCs which are edge-protected, there is a guarantee for the existence of a mesh which is constrained Delaunay:

Lemma 2.7 (Constrained Delaunay mesh of edge-protected PLC). *Let $(\mathcal{P}, \widetilde{\xi})$ be an edge-protected multi-region PLC. There is a triangulation $(\mathcal{M}, \xi)$ of the vertices of $(\mathcal{P}, \widetilde{\xi})$ which is constrained Delaunay and geometry-conforms to $(\mathcal{P}, \widetilde{\xi})$.*

A proof for this Lemma can be found in [127]. For 2D PLCs in the plane there is an even stronger statement: Every multi-region PLC $(\mathcal{P}, \widetilde{\xi})$ has a multi-region triangulation $(\mathcal{M}, \xi)$ of the vertices of $(\mathcal{P}, \widetilde{\xi})$ which respects and geometry-conforms to $(\mathcal{P}, \widetilde{\xi})$. However, in the 3D case, the edge-protected property is mandatory. To obtain a mesh which is Delaunay or constrained Delaunay, additional vertices have to be inserted. For every multi-region PLC, there exists a mesh which is Delaunay or constrained Delaunay [127].

2.4 Data Structure for Meshes

As mentioned in the introduction, one goal of this work is to optimize memory requirements of meshes. Therefore, a simple data structure for multi-region mesh is presented in this section which will later be used for memory usage comparison. The programming language C is used to define the data structures as C provides a common interface for a plethora of other programming languages. Let $\mathcal{M}$ be an n-dimensional mesh and $(\mathcal{M}, \xi)$ be an n-dimensional multi-region mesh to be stored in memory: The data structure of $\mathcal{M}$ and $(\mathcal{M}, \xi)$ are defined as shown in Listing 2.1 and Listing 2.2, respectively.

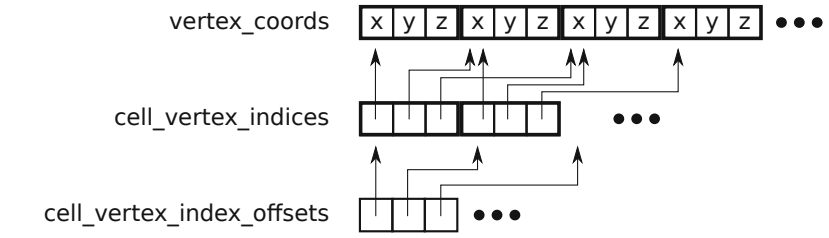


Figure 2.13: Mesh data structure layout

The vertices are stored in the linear coordinate array `vertex_coords` illustrated in the top. For each cell, the indices of all of its vertices are stored in the `cell_vertex_indices` array. The `cell_vertex_index_offsets` is used to determine the first and the last vertex index for each cell in the `cell_vertex_indices` array. The `cell_types` and `cell_region` arrays are simple linear buffers and not visualized in this illustration.

Listing 2.1: Mesh data structure

```
struct MESH
{
  int               point_dimension;
  int               vertex_count;
  NUMERIC_TYPE      vertex_coords[];

  int               cell_count;
  ELEMENT_TYPE      cell_types[];
  INDEX_TYPE        cell_vertex_indices[];
  INDEX_TYPE        cell_vertex_index_offsets[];
};
```

Listing 2.2: Multi-region mesh data structure

```
struct MRMESH
{
  MESH              mesh;
  REGION_ID_TYPE    cell_region[];
};
```

The only difference between these two mesh data structures is the `cell_region` array, which is present in the multi-region mesh data structure but not in the other one. The storage layout of the data structures, illustrated in Figure 2.13, is motivated by the compressed sparse row format for sparse matrices [141].

Usually, IEEE 754 32-bit single or 64-bit double floating point types [1] are used as `NUMERIC_TYPE`. For each vertex $\boldsymbol{v} \in \text{vertices}((\mathcal{M}, \xi))$, n numerical values are stored. Therefore the vertex coordinate array has a size of $n \times |\text{vertices}((\mathcal{M}, \xi))| \times \text{sizeof}(\text{NUMERIC\_TYPE})$.

A cell $C \in \text{cells}((\mathcal{M}, \xi))$ is defined using an identifier for its element type and a tuple of indices for each boundary vertex $\boldsymbol{v} \in \text{faces}_0(C)$. Only element types which can be uniquely defined using a tuple of boundary vertices can be used with this method. For example, a generic non-convex polyhedron cannot be defined by using only its boundary vertices. However, all elements presented in this work support this method. The element type identifier can be represented by an integer value with the size (in bits) equal to the binary logarithm of number of different element types, rounded up. A total of eight different element types are used in this work, so the size of an element type identifier is at least three bits. However, due to memory aliasing issues and to avoid bit shifting operations, an eight bit integer data type is used as `ELEMENT_TYPE`. The vertex index tuples of each cell are stored sequentially in the cell vertex index array using an arbitrary ordering of the cells.

The starting index for the i-th cell is stored in the cell vertex index offset array at location i. Therefore, the number of vertices for the i-th cell is equal to `cell_vertex_index_offsets[i+1] - cell_vertex_index_offsets[i]`. For convenience, the cell vertex index offset array has a size equal to the number of cells plus one. The last entry of the cell vertex index offset array is the total number of all vertex indices of all cells. `INDEX_TYPE` is an integer type with at least 32 bits. A 64 bit integer type is only required for meshes with more than 2^{32} vertices. For each cell in a multi-region mesh, there is an additional region identifier which is stored in the cell region array. Depending on the maximum number of regions supported, `REGION_ID_TYPE` is an integer type with either 8, 16, or 32 bits. Meshes with more than 2^{16} regions are very rare, thus a 16 bit integer is used. A configuration with `NUMERIC_TYPE` being a 64 bit double floating point type, `ELEMENT_TYPE` being an 8 bit integer type, `INDEX_TYPE` being a 32 bit integer type, and `REGION_ID_TYPE` being a 16 bit integer type is used in this work.

The total memory usage of a mesh mainly depends on the number of its cells and vertices. For meshes with only one cell type, like simplex meshes or all-quad meshes, the number of boundary vertices per cell can be trivially determined for all cells of the mesh. Therefore, the size of the cell vertex index array can be calculated. However, only an upper bound can be given for mixed meshes.

Therefore, the upper bound for the memory usage (in byte) of a mesh $\mathcal{M}$ and a multi-region mesh $(\mathcal{M}, \xi)$ in $\mathbb{R}^n$ is given by:

$$\text{sizeof}(\mathcal{M}) \leq |\text{vertices}(\mathcal{M})| \times 8 \times n + |\text{cells}(\mathcal{M})| \times \left(7 + 4 \times \max_{C \in \text{cells}(\mathcal{M})} (\text{faces}_0(C))\right) \tag{2.11}$$

$$\text{sizeof}((\mathcal{M}, \xi)) \leq |\text{vertices}((\mathcal{M}, \xi))| \times 8 \times n + |\text{cells}((\mathcal{M}, \xi))| \times \left(5 + 4 \times \max_{C \in \text{cells}((\mathcal{M}, \xi))} (\text{faces}_0(C))\right) \tag{2.12}$$

The variables `point_dimension`, `vertex_count`, and `cell_count` are not covered by that formula, because their size in memory is negligible.

Chapter 3

Related Work

In this chapter, the results of a literature research on volumetric meshing for simulations is presented. Section 3.1 covers mesh generation algorithms for simplex and all-quad/all-hex meshes as well as mesh generation software tools. Mesh adaptation algorithms with a focus on mesh optimization are presented in Section 3.2. Research on symmetry, similarity, and symmetry-aware mesh processing is discussed in Section 3.3.

3.1 Mesh Generation

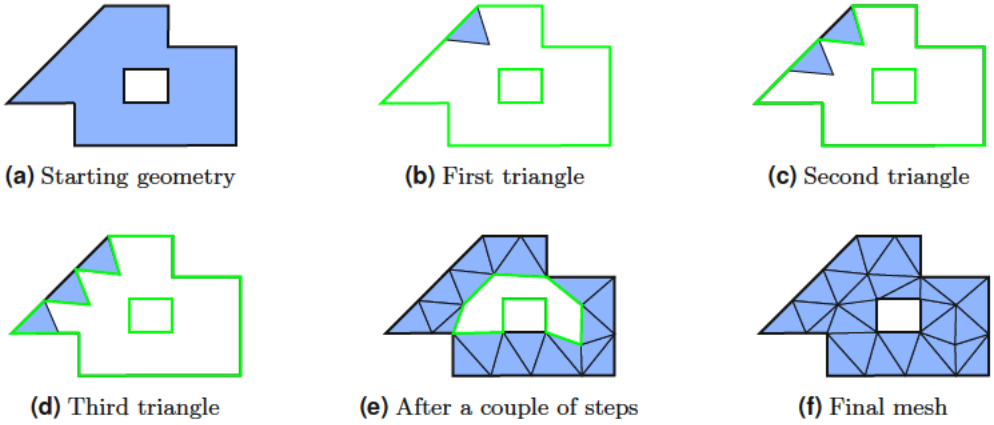
Over time, several algorithms have been developed for generating a mesh based on a given geometry. An overview on mesh generation techniques has been published, e.g., Owen [121]. Simplex mesh generation (Section 3.1.1) as well as quadrilateral and hexahedral mesh generation (Section 3.1.2) methods are presented in this section, followed by an overview of a selection of mesh generation software tools in Section 3.1.3.

3.1.1 Simplex Mesh Generation

There are three primary groups of mesh generation algorithms for simplex meshes [127]:

- Advancing front algorithms
- Delaunay refinement algorithms
- Grid, quad- and octtree algorithms

Advancing front algorithms [39][60] require the boundary of the geometry and all region interfaces to be represented by a hull mesh. For each region the meshed region boundary is the starting *front*. Iteratively, the algorithm constructs volumetric mesh elements on the interior of the front and updates the front. This process is repeated, until the whole interior is filled with mesh elements. Figure 3.1 shows how an advancing front algorithm works on a simple two-dimensional geometry. Advancing front algorithms usually produce elements with high quality at the region boundaries [127]. However, in situations where the fronts collide, the algorithms have less freedom when constructing volumetric elements and therefore the element quality degrades.

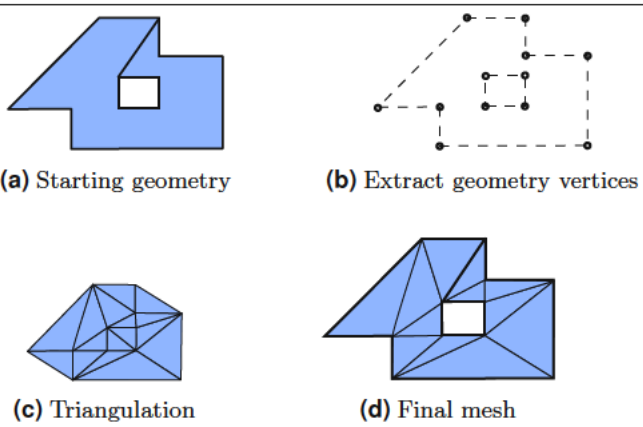


(a) Starting geometry **(b)** First triangle **(c)** Second triangle

(d) Third triangle **(e)** After a couple of steps **(f)** Final mesh

Figure 3.1: Advancing front mesh generation

In (a), the starting geometry is visualized. The boundary of this geometry is used as the initial front, drawn in green. Using this front, new volumetric elements are constructed iteratively on the inside and the front is updated ((b) to (d)). (e) shows the mesh after several of steps and the final mesh is shown in (f).



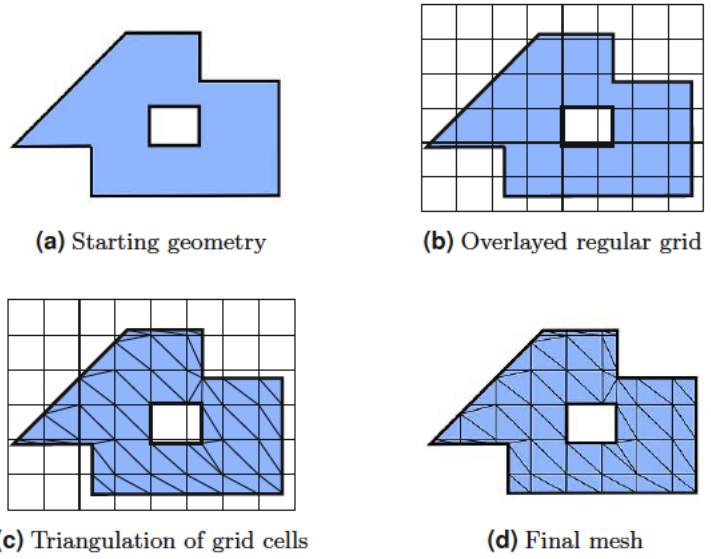
(a) Starting geometry **(b)** Extract geometry vertices

(c) Triangulation **(d)** Final mesh

Figure 3.2: Delaunay refinement mesh generation

(a) visualizes the starting geometry. At first, only the vertices of the geometry are considered (b) and a triangulation of these vertices is created as shown in Figure (c). Flip operations and vertex insertions are used to make every simplex of the triangulation Delaunay. To ensure geometry-conformity, additional constraints from the PLC are inserted into the final mesh, again using flipping or splitting operations (cf. Section 3.2). Triangles which are outside of the geometry are removed. The final constrained Delaunay mesh is visualized in Figure (d).

Using only the vertices of the geometry, Delaunay refinement algorithms [56][127] first create a triangulation which is Delaunay. Then, elements are flipped and/or split to ensure geometry-conformity of the mesh. Mesh elements not included in the underlying space of the geometry are discarded. If needed, Delaunay refinement (see Section 3.2.3) is used to ensure the Delaunay or constrained Delaunay property of the resulting mesh. Figure 3.2 sketches a Delaunay refinement algorithm for a two-dimensional geometry. In contrast to advancing front algorithms, Delaunay refinement algorithms usually produce elements with *good* quality in the interior while having elements with low quality near the boundary or region interfaces. Additionally, there are Delaunay refinement algorithms which create meshes with mathematical guarantees for mesh element quality [127].



(a) Starting geometry **(b)** Overlayed regular grid

(c) Triangulation of grid cells **(d)** Final mesh

Figure 3.3: Overlaying grid mesh generation

The starting geometry is visualized in (a). At first, a regular square grid is laid over the geometry (b). Square grid cells which do not intersect the geometry boundary can trivially be decomposed into triangles. For all other square grid cells, a triangulation has to be found which respects the boundary (c). In contrast, a quadtree algorithm would recursively refine square grid cells which intersect with the geometry boundary, until the intersection is simple enough to construct a valid triangulation for that square grid cell. Elements near the geometry boundary in the final mesh have bad quality (d).

Grid, quad-, and octtree algorithms [81][82] use an overlaying regular grid, quad-, or octtree to construct mesh elements. Grid vertices are moved to the geometry boundary or region interfaces to ensure geometry-conformity. Figure 3.3 shows an example of a mesh generated by a mesh generation algorithm using an overlaying grid. Meshes generated by this algorithm produce elements with very high quality in the interior but elements with very bad quality near the boundary or region interfaces [60].

3.1.2 Quadrilateral and Hexahedral Mesh Generation

Due to their topology, quadrilateral and hexahedral meshes are naturally related to regular grids. Meshes, which are topologically equivalent to a regular grid, are called structured. Structured mesh generation algorithms play an important role for all-quad and all-hex meshes [44]. Simplex meshes can also be structured, but they are far less common than structured quadrilateral or hexahedral meshes. However, due to their topological inflexibility, structured meshes are not covered in this thesis.

One of the first algorithms published, which generates unstructured all-quad 2D meshes, is called Paving [134]. Paving is based on an advancing front approach and therefore requires the geometry to be represented as a line mesh. Another popular method for creating quadrilateral meshes uses a triangle mesh and *morphs* the triangles into quadrilaterals [40][122]. A technique called *sweeping* can be used to generate a hexahedral mesh based on a quadrilateral mesh of the boundary [106]. When using sweeping, two opposing sides of the geometry are selected. A quad mesh is generated for both sides, and these quad meshes are extruded to a hexahedral mesh. Similar to simplex mesh generation, grid overlay methods using a regular grid (or a quad- or octtree) have also been developed for all-quad and all-hex mesh generation [113][138]. Lately, methods using frame fields were proposed for quadrilateral and hexahedral mesh generation [100][101].

	License	Element Types	Input Geometry
CGAL	GPL,LGPL	Tri,Tet	oracle*
CUBIT	commercial	Tri,Tet,Quad,Hex	IGES,STEP,STL
Gmsh	GPL	Tri,Tet	IGES,STEP,STL
MeshGems	commercial	Tri,Tet,Quad,Hex,Mixed	IGES,STEP,STL
Netgen	LGPL	Tri,Tet	CSG,STEP,STL
Pointwise	commercial	Tri,Tet,Quad,Hex,Mixed	CSG,IGES,STEP,STL
Tetgen	X11/MIT	Tet	PLC
Triangle	commercial,GPL-like	Tri	PLC
ViennaMesh	GPL	Tri,Tet	CSG,PLC,STEP,STL

Table 3.1: Overview of the meshing software presented in this work

The GNU general public license (GPL) [6], the GNU lesser general public license (LGPL) [6], and the X11/MIT license [16] are free open source licenses. The listed element types are triangles (Tri), tetrahedrons (Tet), quadrilaterals (Quad), hexahedrons (Hex), and mixed element types (Mixed). More information on the input geometry formats can be found in Section 2.1. * CGAL itself is a software library rather than a software tool and uses a generic mechanism called oracle to specify the geometry. Oracle implementations for polyhedrons, implicit functions and 3D images are provided by CGAL.

3.1.3 Mesh Generation Software

In this section, a selection of volumetric mesh generation software tools is presented. An overview is given in Table 3.1.

The Computational Geometry Algorithms Library (**CGAL**) [2] is a free open source library. Among the rich set of provided geometric algorithms, CGAL offers incremental Delaunay meshing algorithms for two- and three-dimensional geometries. A peculiarity of CGAL is the generic software design which significantly increases flexibility [36].

CUBIT [15] is a commercial mesh generation tool developed at the Sandia National Laboratories. It supports a wide range of different mesh generation algorithms with a focus on quadrilateral and hexahedral mesh generation.

Gmsh [5][35] is a free open source meshing framework providing a couple of meshing algorithms such as structured and unstructured mesh generation of two- and three-dimensional meshes. Gmsh uses other free open source meshing tools and combines these with pre- and post-processing steps. Unlike other meshing applications, Gmsh also includes a scripting language for controlling the meshing process.

MeshGems [7] is a commercial mesh generation software which supports triangular and quadrilateral surface mesh generation as well as tetrahedral, hexahedral, and mixed volumetric mesh generation. Additionally, the tool offers multiple algorithms for CAD geometry enhancement and error cleaning.

Netgen [9][130] is a free open source tetrahedral advancing front mesh generation tool with support for CSG and triangular hull mesh input geometries. Additional post-processing steps ensure the Delaunay property of the resulting meshes.

Pointwise [10] is a commercial mesh generation software supporting a wide range of CAD input geometries. It is capable of generating simplex, quadrilateral, hexahedral, and mixed element meshes.

Tetgen [14][53] is a free open source 3D incremental Delaunay mesh generation software with support for multiple regions. The tool uses PLCs as input geometries and is related to Triangle as the algorithms, the concepts, and the programming interface are very similar.

Triangle [17][64] is a very popular and widespread free open source 2D incremental Delaunay mesh generation software tool.

ViennaMesh [19][49] is a flexible free open source meshing framework based on a module approach. Besides providing a unified interface to several external meshing tools, it also provides additional mesh adaptation algorithms via an extensible platform.

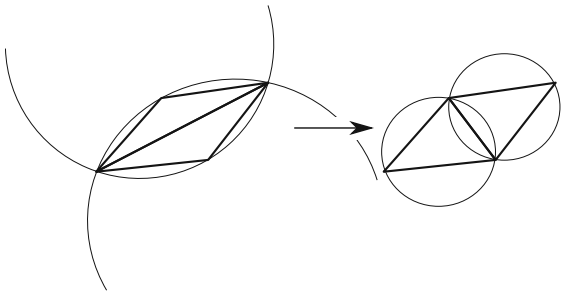


Figure 3.4: Edge flipping operation for triangles

The edge shared by the triangles is flipped to the other diagonal of the quadrilateral which is formed by the union of two triangles. The flipping operation requires the quadrilateral to be convex, otherwise the underlying space changes. In this particular example, the triangles are Delaunay (cf. Section 2.3) after the flipping operation is applied.

3.2 Mesh Adaptation Algorithms

A mesh adaptation algorithm takes a multi-region mesh $(\mathcal{M}, \xi)$ as the input to create a resulting mesh $(\widetilde{\mathcal{M}}, \widetilde{\xi})$, for which the underlying space of all regions is the same: $\forall i = 1, \ldots, \mathrm{rc}((\mathcal{M}, \xi)) : \mathrm{us}(\mathrm{region}((\mathcal{M}, \xi), i)) = \mathrm{us}(\mathrm{region}((\mathcal{R}, \widetilde{\xi}), i))$. In other words, the input mesh and the resulting mesh conform to the same geometry.

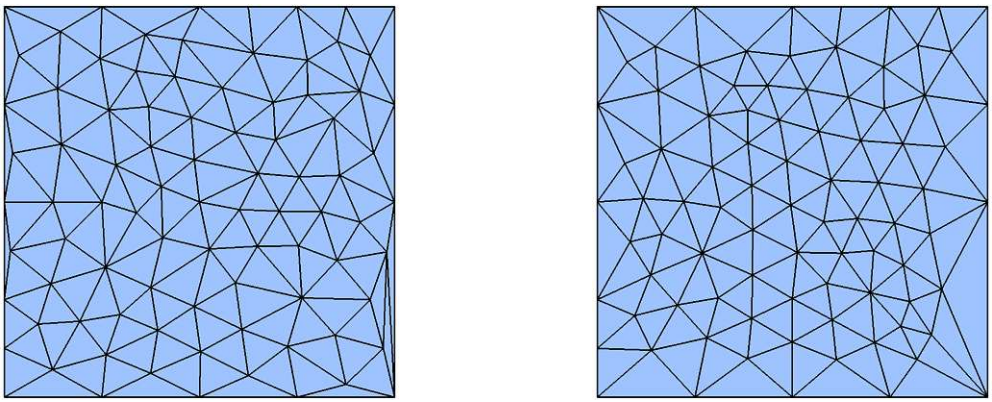
Mesh adaptation algorithms can roughly be split into three groups, those being topological operations (cf. Section 3.2.1), geometrical operations (cf. Section 3.2.2), and mixed operations (cf. Section 3.2.3). These operations are often used for mesh improvement, as discussed in Section 3.2.4. At last, Section 3.2.5 gives an overview of a selection of mesh adaptation software tools.

3.2.1 Topological Operations

Topological operations are operations which do not insert, delete, or move any vertices. The most popular topological operations are *flips*, where a number of neighboring cells are replaced by another configuration of cells. For example, an edge flip, visualized in Figure 3.4, is an operation, where two triangles sharing one edge are replaced by two different triangles. Flip operations for 2D triangle meshes have a special role as they can make any mesh Delaunay. It can be shown, that for a triangulation every edge E is either locally Delaunay or E is flippable and the flipped edge is locally Delaunay [127]. It can also be shown, that the algorithm which recursively flips every edge which is not locally Delaunay terminates and therefore results in a Delaunay mesh [127]. The 3D extension of the 2D flips are called *bistellar flips* [50]. However, bistellar flips are fairly complex and cannot be applied to all configurations of neighboring tetrahedrons. Additionally, recursively performing bistellar flips on tetrahedral meshes only results in a mesh which is Delaunay for edge-protected PLCs [67]. However, it has been proven, that there are algorithms using flip operations which result in a simplex Delaunay mesh for arbitrary dimensions n [136]. Flip operations can be defined for arbitrary element types by using the projections of an element of the same type with one dimension higher [83].

3.2.2 Geometrical Operations

Geometrical operations, or vertex smoothing algorithms, are operations which only move vertices but do not insert or delete any elements. Vertex smoothing algorithms are very popular algorithms for improving mesh element quality, because they do not alter the mesh topology. Since they do not delete or insert any new elements, they are also easy to handle from the data structure point of view.



(a) Before Laplacian smoothing

(b) After Laplacian smoothing

Figure 3.5: Laplacian smoothing

The triangles after the Laplacian smoothing operation, especially those near the boundary, are less *skinny* and have better quality (cf. Section 2.2.2).

Simple movement algorithms, like Laplacian smoothing visualized in Figure 3.5, have been shown to improve the overall mesh quality in many applications [41]. Laplacian smoothing also works well for non-simplex meshes [116]. However, to ensure geometry-conformity, the vertex smoothing algorithms have to be modified by restricting vertex movements of boundary vertices [124]. More complex smoothing algorithms, for example vertex smoothing based on the optimal Delaunay triangulation [78] or using local quality measure optimization [95], have also been proposed.

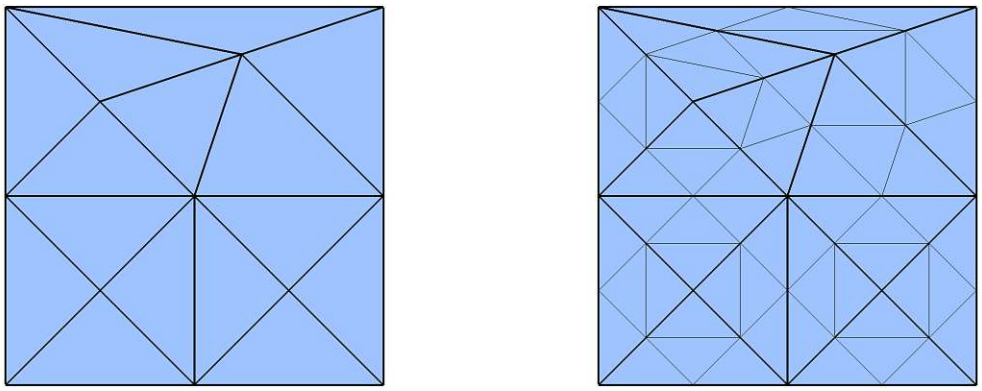
3.2.3 Mixed Operations

There are operations which can neither be classified as purely topological or purely geometrical. The most popular operations, being mesh refinement algorithms, are presented in this section.

Mesh refinement algorithms are similar to flips, where a number of cells is replaced by another configuration of cells. In contrast to flips, mesh refinement algorithms also require that every cell of the input mesh is a union of cells of the refined mesh. In other words, the refined mesh has to respect the original mesh.

Definition 3.1 (Refined mesh)**.** Let $(\mathcal{M}, \xi)$ and $(\mathcal{R}, \widetilde{\xi})$ be multi-region meshes. $(\mathcal{R}, \widetilde{\xi})$ is called a refinement of $(\mathcal{M}, \xi)$, or a refined mesh of $(\mathcal{M}, \xi)$, if every cell C in $(\mathcal{R}, \widetilde{\xi})$ can be represented by a union of cells $C_1, \ldots, C_k$ of $(\mathcal{M}, \xi)$: $C = \bigcup_{i=1}^{k} C_i$.

The main advantage of a refined mesh is that for every refined cell there is exactly one parent cell in the original mesh. This is particularly useful when transferring additional meta information stored on elements from the original mesh to the refined mesh or vice versa. Figure 3.6 shows an example of a mesh refinement. A popular type of refinement uses edge bi- or trisection operations, where an edge is marked for refinement and split into two or three new edges. For each element type there are rules on how to replace a cell with a specific configuration of marked edges by refined cells. To ensure element quality, additional edges can be marked for refinement. For example, a popular refinement strategy using this approach is *red-green-blue* refinement [34]. Such types of refinement algorithms have lower bounds for some element qualities, like the smallest angle [55]. Trisection of quadrilaterals and hexahedrons is frequently used in favor of bisection [114].



(a) Before refinement **(b)** After refinement

Figure 3.6: 2D simplex refinement

Every triangle of the initial mesh is refined into four new triangles.

Mesh refinement is often utilized in adaptive processes for boundary value problems, where error estimators are used to locally increase the accuracy of the solution. Error estimators detect mesh elements, where the difference of the numerical solution and the real solution is high, and mark those for refinement.

A related algorithm is Delaunay mesh refinement of simplex meshes [21]. Delaunay mesh refinement cannot be considered a classical refinement algorithm, because in general there are cells in the refined mesh which do not respect the original mesh. Delaunay mesh refinement incrementally inserts vertices in a mesh, which already is Delaunay. A popular Delaunay mesh refinement algorithm is the Bowyer-Watson algorithm [42]. Given a specific point $p \in \mathbb{R}^n$, all cells C of a mesh are removed where $p \in \overline{\mathcal{B}}^n(C)$. Every facet of these cells, which is also in the boundary of the union of these cells, is then connected to the point p to form a new simplex cell. It has been shown, that the new mesh is also Delaunay, if the original mesh is Delaunay. This statement also holds for constrained Delaunay with the modification that only cells which are visible to the newly inserted point are deleted. Optimal locations for Delaunay mesh refinement can be determined by using a conflict graph [73].

3.2.4 Mesh Quality Improvement

The main goal of many mesh adaptation algorithms is to improve the mesh element quality. Mesh element quality improvement algorithms are available for both simplex and all-quad/all-hex meshes. There are algorithms for simplex meshes, which generate meshes with guaranteed element quality. Usually, mesh quality improvement algorithms have a set of local operations which they *test* on a set of elements: If an operation locally increases the mesh element quality, it is applied. For simplex, all-quad, and all-hex meshes, it is often sufficient to perform a combination of mesh smoothing and flip operations to obtain meshes with a *good* quality [75][79]. However, for tetrahedral meshes, these operations might lead to poor local quality optima. More aggressive methods, like vertex insertion, are required to further increase the mesh element quality [32]. In contrast to triangular meshes, algorithms which optimize the radius-edge ratio of tetrahedral meshes still might generate slivers (cf. Section 2.2.2). Sliver exudation algorithms must be applied after mesh generation to remove this type of element [129]. Approaches for smoothing algorithms can be extended to various element types [76]. There are optimization algorithms for quadrilateral and hexahedral meshes operating on the dual mesh, which, however, might be non-local [108][115]. Many algorithms even consider complete mesh re-sampling to improve mesh element quality [139].

3.2.5 Mesh Adaptation Software

Similar to Section 3.1.3, a selection of software tools for mesh adaptation and quality improvement is presented here.

The commercial software tool **CUBIT** [133] offers, in addition to its mesh generation algorithms, a wide range of optimization methods for different element types using various metrics. Similar to its mesh generation infrastructure, the software focuses on quadrilateral and hexahedral meshes.

Mesquite [8][85] is an open source mesh optimization software with a rich set of different optimization algorithms, including support for mixed element meshes, anisotropic smoothing, and local size control.

The commercial software tool **Pointwise** [11] includes mesh optimizations methods with a focus on CFD applications.

Stellar [12] is an open source mesh optimization software which mainly improves dihedral angles of tetrahedral meshes. The software is based on the aggressive tetrahedral mesh improvement algorithm [32].

Verdict [18][109] is an open source software for measuring mesh element quality. However, the software does not provide any mesh adaptation and is therefore strictly speaking not a mesh adaptation software.

3.3 Symmetry and Similarity

The main goal of this work is to take advantage in storing and generating meshes of geometries with symmetries and similarities. The identification process of these properties is non-trivial but crucial. Two different geometrical properties are of interest: Symmetries and similarities. In the literature, similarities are often referred to as local symmetries or partial symmetries.

In general, the detection of symmetries and similarities is a very challenging task. One has to distinguish between exact and approximate symmetries or similarities. The first rarely occurs in practice due to numerical issues or inaccuracies in geometric modeling. The latter does not suffer from such issues. Section 3.3.1 and Section 3.3.2 covers detection of symmetries and similarities, respectively. Symmetry-aware mesh processing is discussed in Section 3.3.3.

3.3.1 Symmetry Detection

Detection of symmetries is easy for linear geometries in $\mathbb{R}^2$. The initially devised algorithms create string literals based on the points and lines of the linear geometries for elements with the same distance to the center of gravity and using circular rotations and reversal of these literals to detect rotational and reflective symmetries [94]. These approaches can be extended to linear 3D geometries, however, additional information, like the axis of symmetry, is required [58]. Another approach uses surface sampling and clustering to detect symmetries for 2D and 3D geometries [98]. This algorithm is also capable of detecting not only exact but also approximate symmetries. A technique called generalized moment has also proven to be robust for identifying symmetries of a geometry in $\mathbb{R}^3$ [26]. This approach uses a spherical harmonics representation to efficiently find extrema of the generalized moment, which indicate potential symmetry candidates. However, a post processing step has to validate these candidates to filter out false positives. A similar algorithm uses the planar-reflective symmetry transform for detecting reflective symmetries [63]. The latter three algorithms can also be used for approximate symmetry detection.

3.3.2 Similarity Detection

In contrast to symmetry detection, automatic similarity detection is much harder and more time-consuming. Some algorithms for symmetry detection also cover similarity detection [96][98]. A framework for identifying regular and similar shapes in 3D objects was proposed [91]. Other approaches even support transformation functions which are not required to be rigid [20]. The *symmetry factored embedding* and the *symmetry factored distance* can be used to analyze symmetries in points sets [140]. A hierarchical approach was proposed for building a graph of all subparts of an object [142]. Usually, all of these algorithms use the surface of the object to sample points and detect local features to find matching similarities [84]. A good overview of symmetry and similarity detection algorithms is given in [99].

3.3.3 Symmetry-Aware Mesh Processing

Symmetry-aware mesh processing is a popular area of research in the field of computer graphics, for example symmetrization [97][137]. Symmetries and similarities are used in re-construction and mesh healing processes after 3D scanning [30][68][123]. As mentioned in the introduction (cf. Chapter 1), one of the main approaches for this work is a technique in the area of computer graphics called *instancing*: With instancing, a high number of instances of a single object template is rendered into a scene at different locations [37][47][71][92].

However, most algorithms and concepts in the field of computer graphics only work for surface geometries or surface meshes. Some of the properties vital for simulation in science and engineering, like conformity, are, however, not important in the field of computer graphics. For example, in a rendering process, meshes are not required to be *watertight* or *intersection-free* [132].

A more general approach for symmetry-aware mesh processing, which also supports volumetric meshes, uses group theory and the generalized Fourier transform [72]. This work, however, only covers symmetries and lacks support of general similarities. Benchmarks for rotational symmetry-aware 3D volumetric mesh generation yield possible performance gains of factors of up to 50 [48]. Improvements of these concepts and approaches for supporting other types of symmetries and similarities are presented and discussed in Section 5.1.2.

A topic related to similarity in meshes concerns periodic meshes, where one part of the mesh is repeated in several directions. Generation of periodic meshes has been investigated and implemented in the CGAL software library [29][86].

As motivated in the introduction (cf. Chapter 1), mesh generation and mesh adaptation algorithms would benefit from a theoretical approach which considers symmetries and/or similarities. However, there is no literature for such an approach which at the same time takes symmetries and similarities in account and additionally supports multi-region geometries and meshes.

Chapter 4

Templated Geometries and Meshes

The main objective of this work is inspired by a concept in the field of computer graphics called *instancing*. In instancing approaches, a geometry template is defined which is then inserted into a scene multiple times at different locations, often also slightly modified. Figure 4.1 shows a simple example of instancing in computer graphics. The main idea of this work is to decompose geometries and meshes into smaller parts, called blocks, where as many blocks as possible are similar to each other to obtain high improvements in memory usage and algorithm runtime. In this chapter, geometry and mesh templates – and their instances – are defined and some of their properties are presented and discussed. General decompositions as well as the basics of templated structures are covered in Section 4.1. Section 4.2 discusses interfaces of templated structures as well as conformity issues. The boundary patch partition, a mechanism to identify dependencies and constraints, is presented in Section 4.3. The Delaunay property for templated meshes is discussed in Section 4.4. Finally, in Section 4.5, two different data structures for storing templated meshes are presented.

4.1 Decomposition of Geometries and Meshes, Templated Structures

First, decompositions of geometries and meshes are defined. Geometry and mesh decompositions are similar to manifold partitions and mesh partitions, respectively, with the difference that they support multiple regions.

Definition 4.1 (Geometry decomposition, geometry block, geometry template, geometry instance)**.** Let $(\mathcal{G}, \widetilde{\xi})$ be an n-dimensional multi-region geometry. $((\mathcal{G}_1, \widetilde{\xi}_1), \ldots, (\mathcal{G}_k, \widetilde{\xi}_k))$ is called a geometry decomposition of $(\mathcal{G}, \widetilde{\xi})$, if

(i) $(\mathcal{G}_1, \ldots, \mathcal{G}_k)$ is a manifold partition where all elements have the same dimension n and

(ii) all region indicator functions $\widetilde{\xi}_i$ are the restriction of $\widetilde{\xi}$ on the corresponding interior of the geometry partition element: $\widetilde{\xi}_i = \widetilde{\xi}|_{\mathrm{int}^\star(\mathcal{G}_i)}$.

A multi-region geometry $(\mathcal{T}, \widetilde{\zeta})$ is called a geometry template of $(\mathcal{G}_i, \widetilde{\xi}_i)$, if there is an invertible function $T : \mathbb{R}^n \to \mathbb{R}^n$, for which $(T(\mathcal{T}), \widetilde{\zeta} \circ T^{-1}) = (\mathcal{G}_i, \widetilde{\xi}_i)$. $(\mathcal{G}_i, \widetilde{\xi}_i)$ is called a geometry instance of $(\mathcal{T}, \widetilde{\zeta})$ with a transformation function T.

Figure 4.1: Instancing

The head of a monkey is rendered 20 times at different locations.

(a) Multi-region mesh **(b)** Decomposition

Figure 4.2: Decomposition of a multi-region mesh

Similar terms can be defined for meshes as well.

Definition 4.2 (Mesh decomposition, mesh block, mesh template, mesh instance)**.** Let $(\mathcal{M}, \xi)$ be an n-dimensional multi-region mesh: $((\mathcal{M}_1, \xi_1), \ldots, (\mathcal{M}_k, \xi_k))$ is called a mesh decomposition of $(\mathcal{M}, \xi)$, if

(i) $(\mathcal{M}_1, \ldots, \mathcal{M}_k)$ is a mesh partition with all elements having the same dimension and

(ii) all region indicator functions ξ_i are the restrictions of ξ on the corresponding *interior* mesh partition element: $\xi_i = \xi|_{\mathcal{M}_i \setminus \mathrm{bnd}(\mathcal{M}_i)}$.

A multi-region mesh $(\mathcal{T}, \zeta)$ is called a mesh template of $(\mathcal{M}_i, \xi_i)$, if there is an invertible function $T : \mathbb{R}^n \to \mathbb{R}^n$, for which $(T(\mathcal{M}_i), \zeta \circ T^{-1}) = (\mathcal{M}_i, \xi_i)$. $(\mathcal{M}_i, \xi_i)$ is called mesh instance of $(\mathcal{T}, \zeta)$ with transformation function T.

An example of a mesh decomposition is visualized in Figure 4.2. In general, arbitrary invertible transformation functions T can be used for instances, but it is beneficial to restrict the transformation functions to functions which are angle-preserving. The angle-preserving property is important, because most popular mesh element quality measures are invariant under such transformations. Therefore, only rigid transformations which are rotations, translations, reflections, and their combinations, are used.

On the contrary, a multi-region geometry (or multi-region mesh) can be defined using a set of templates, each together with a set of transformation functions. A naive approach is to define the templated structure of a multi-region geometry or mesh as a collection of multi-region templates each with a transformation function for each instance. However, it is also possible to use templates with only one region to represent a multi-region mesh or geometry. For each multi-region template in this naive approach, each region can be seen as a template on its own with the same transformation function and an additional region indicator. The resulting templated structure is defined as follows:

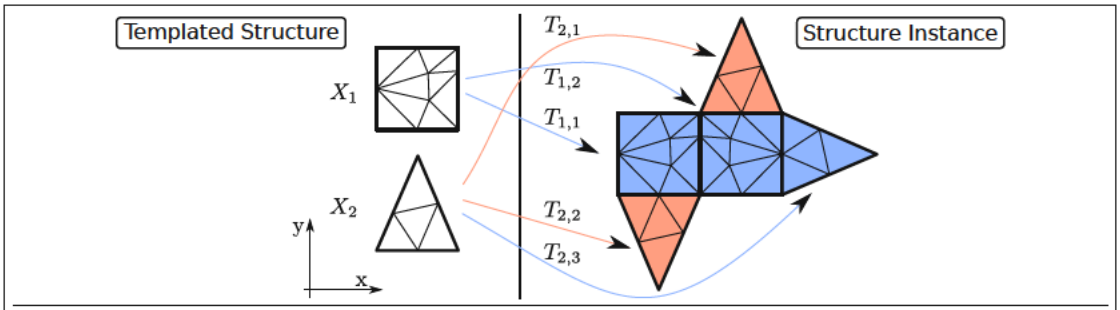


Figure 4.3: Templated mesh

The mesh template X_1 has two instances which are generated using the transformation functions $T_{1,1}$ (a translation) and $T_{1,2}$ (a combination of a reflection around the y-axis and a translation). The mesh template X_2 has three instances which are generated using the transformation functions $T_{2,1}$ (a translation), $T_{2,2}$ (a combination of a rotation by 180° and a translation), and $T_{2,3}$ (a combination of a clockwise rotation by 90° and a translation). The transformation functions and the region indicators of the instances are visualized by the corresponding colored arrows.

Definition 4.3 (Templated structure). Let $X_1, \ldots, X_k$ be either geometries or meshes having the same dimension, $T_{i,j}$ be transformation functions, and $r_{i,j} \in \mathbb{N}, j = 1, \ldots, m_i$ region indicators. The structure

$$\mathbb{X} = ((X_1, (T_{1,1}, \ldots, T_{1,m_1}), (r_{1,1}, \ldots, r_{1,m_1})), \ldots, (X_k, (T_{k,1}, \ldots, T_{k,m_k}), (r_{k,1}, \ldots, r_{k,m_k}))) \tag{4.1}$$

is called a templated structure, if the intersection of the interior of the two different instances is empty:

$$\forall T_{i,g}, T_{j,h}, T_{i,g} \neq T_{j,h} : \text{int}^\star(\text{us}(T_{i,g}(X_i))) \cap \text{int}^\star(\text{us}(T_{j,h}(X_j))) = \emptyset \tag{4.2}$$

A templated structure is a collection of templates X_i, which form multiple instances by using transformation functions $T_{i,j}$ and region indicators $r_{i,j}$. An instance is generated by applying the transformation function $T_{i,j}$ to the template and using a region indicator function which maps every element of that instance to the corresponding region indicator $r_{i,j}$. Therefore, the instance of the template X_i with a transformation function $T_{i,j}$ and a region indicator $r_{i,j}$ is equal to $(T_{i,j}(X_i), R)$ with $R(x) := \{r_{i,j}\}$. The advantage of this definition (in contrast to using multi-region templates) is the simplicity of the templates, which have just one region. The boundary patch partition, an instrument to identify and address conformities in templated meshes (see Section 4.3), is much easier to generate, when the templates have one single region.

The instance of a templated structure, in short called *structure instance*, is obtained by using the *apply-template* operator, which is defined in the following way:

$$\text{AT}(\mathbb{X}) := \left(\left\{ \bigcup_{i=1}^{k} \bigcup_{j=1}^{m_i} \text{inst}(\mathbb{X}, i, j) \right\}, R_{\mathbb{X}} \right) \text{ with } R_{\mathbb{X}}(x) := \{r_{i,j} | x \in \text{inst}(\mathbb{X}, i, j)\} \tag{4.3}$$

This operator generates a multi-region geometry or mesh based on the templated structure. A templated structure and its structure instance are visualized in Figure 4.3. A templated geometry and a templated mesh are templated structures, the structure instances of which result in a valid multi-region geometry or multi-region mesh, respectively.

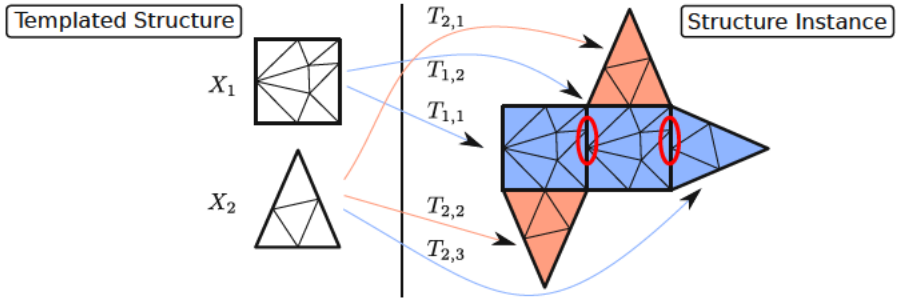


Figure 4.4: Templated structure with non-conforming structure instance

The templated structure is similar to the structure visualized in Figure 4.3 with the difference, that $T_{1,2}$ is just a translation (in contrast of being a combination of a reflection around the y-axis and a translation). This results in a structure instance which is not conforming as indicated by the highlighted red areas.

Definition 4.4 (Templated geometry, templated mesh)**.** A templated structure $\mathbb{X}$ with all templates being geometries is called a templated geometry. A templated structure $\mathbb{X}$ with all templates being meshes and $\text{AT}(\mathbb{X})$ is a valid multi-region mesh is called a templated mesh.

The structure instance of a templated structure with geometry templates is automatically a valid multi-region geometry. However, for a templated structure with mesh templates, its structure instance might not be a valid multi-region mesh as visualized in Figure 4.4. Thus, the additional requirement of $\text{AT}(\mathbb{X})$ being a valid multi-region mesh is necessary. The following two statements are equivalent for a templated structure with mesh templates:

(i) $\text{AT}(\mathbb{X})$ is a multi-region mesh.

(ii) $\text{AT}(\mathbb{X})$ is conforming.

The dimension of a templated structure $\text{DIM}(\mathbb{X})$ is equal to the dimension of all its templates $\text{DIM}(\mathbb{X}_i)$.

It is important for templated structures to find out, if a templated mesh is the mesh of a templated geometry. Therefore, the term geometry-conforming is also defined for templated structures.

Definition 4.5 (Geometry-conforming)**.** A templated mesh Γ is called geometry-conforming to a templated geometry Λ, if

(i) all mesh templates are geometry-conforming to its corresponding geometry templates ($\forall i : \text{templ}(\Gamma, i)$ is geometry-conforming to $\text{templ}(\Lambda, i)$),

(ii) all transformation functions are the same ($\forall i, j : \text{tf}(\Gamma, i, j) = \text{tf}(\Lambda, i, j)$), and

(iii) all region indicators are the same ($\forall i, j : \text{rid}(\Gamma, i, j) = \text{rid}(\Lambda, i, j)$).

The motivation for this definition of the term geometry-conforming for templated structures is the direct correlation between the geometry and mesh instances. For a templated mesh Γ, which geometry-conforms to a templated geometry Λ, all mesh instances are naturally geometry-conforming to the corresponding geometry instances. Therefore, $\text{AT}(\Gamma)$ automatically geometry-conforms to $\text{AT}(\Lambda)$. This definition is also beneficial when formulating mesh adaptation and mesh generation algorithms. Similar to the geometry of a mesh, the templated geometry of a templated mesh is defined as

$$\text{geo}(\Gamma) = ((\text{geo}(\Gamma_1), (T_{1,1}, \dots), (r_{1,1}, \dots)), \dots, (\text{geo}(\Gamma_k), (T_{k,1}, \dots), (r_{k,1}, \dots))) . \tag{4.4}$$

Every templated mesh is naturally geometry-conforming to its templated geometry.

4.2 Instance Interfaces and Conformity

Instances of templated structures, which are not disjoint, form interfaces. Due to Requirement 4.2, these interfaces have a dimension which is lower than the dimension of the templated structure. These interfaces between instances are especially important for templated meshes, where $\mathrm{AT}(\Gamma)$ has to be a valid conforming multi-region mesh: Issues might arise on interfaces between mesh instances, where the conformity can break. For example, if an algorithm performs an operation on a mesh template and ignores other mesh templates where at least two instances of each mesh template *touch* each other, a non-conformity will potentially be induced in the structure instance.

The main goal of this section is to identify which properties a templated structure with mesh templates has to fulfill to be a templated mesh. Identifying mesh instance interfaces and defining neighborhood terms is therefore crucial.

Definition 4.6 (Instance interface, geometry instance interface)**.** Let $\mathbb{X}$ be a templated structure. The interface of multiple instances with transformation functions $T_1, \dots, T_m \in \mathrm{tf}(\mathbb{X})$ is defined as the intersection of the instances:

$$\mathrm{interf}(\mathbb{X}, T_1, \dots, T_m) := \bigcap_{i=1}^{m} \mathrm{inst}(\mathbb{X}, T_i) \tag{4.5}$$

Additionally, the geometry instance interface is defined as the intersection of the underlying spaces of the instances:

$$\mathrm{interf}_{\mathrm{geo}}(\mathbb{X}, T_1, \dots, T_m) := \bigcap_{i=1}^{m} \mathrm{us}(\mathrm{inst}(\mathbb{X}, T_i)) \tag{4.6}$$

The motivation for the definition of $\mathrm{interf}_{\mathrm{geo}}$ lies in the fact that for templated structures with mesh templates, interfaces of two different instance might be empty even though the intersection of their underlying space is not. For example, the interface of the instances formed by the transformation function $T_{1,1}$ and $T_{1,2}$ of the templated structure visualized in Figure 4.4 only includes two vertices. However, the geometry instance interface of the same instances includes all vertices and edges of both instances which *touch* the other instance. The instance interfaces and the geometry instance interfaces are the same for templated geometries. For templated meshes, these two terms are related as indicated by Lemma 4.1.

Lemma 4.1 (Instance interface properties)**.** *Let $\mathbb{X}$ be a templated structure with mesh templates:*

(i) Any instance interface of $\mathbb{X}$ respects the corresponding geometry instance interface.

(ii) If $\mathbb{X}$ is a templated mesh, the underlying space of any instance interface is equal to the corresponding geometry instance interface.

(iii) If $\mathbb{X}$ is a templated mesh, any instance interface is equal to the set of all boundary elements of a mesh template, which are included in the inverse *geometry instance interface, transformed by the corresponding transformation function:*

$$\mathrm{interf}\left(\mathbb{X}, T_{i,j}, \dots\right) = T_{i,j}\left(\mathrm{bnd}\left(X_i\right)|_{T_{i,j}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X}, T_{i,j}, \dots))}\right) \tag{4.7}$$

Proof.

(i) Every element E in an instance interface $\mathrm{interf}(\mathbb{X}, T_1, \dots, T_k)$ is in all instances $\mathrm{inst}(\mathbb{X}, T_i)$ and therefore also in the underlying space of all instances. Consequently, $E \cap \mathrm{interf}_{\mathrm{geo}}(\mathbb{X}, T_1, \dots, T_m) = E$ and E respects the geometry instance interface.

(ii) follows from Lemma 2.1.

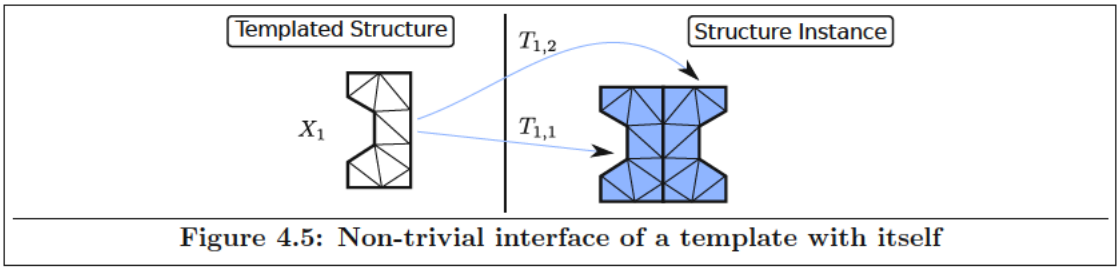


Figure 4.5: Non-trivial interface of a template with itself

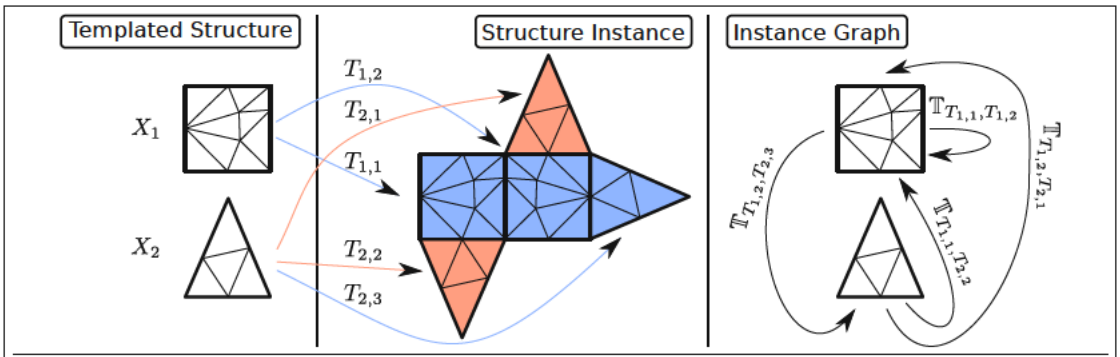


Figure 4.6: Instance graph

Every edge of the graph represents a non-empty instance interface and is indicated using an arrow pointing from and to the boundary sections of the instance interface.

(iii) From the definition of the apply-template operator follows, that $T_{i,j}\left(\mathrm{bnd}\,(X_i)\,|_{T_{i,j}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,j},\dots))}\right)$ is a subset of $\mathrm{interf}\,(\mathbb{X},T_{i,j},\dots)$. On the other hand, assume, that there is an element E in $\mathrm{interf}\,(\mathbb{X},T_{i,j},\dots)$, which is not in $T_{i,j}\left(\mathrm{bnd}\,(X_i)\,|_{T_{i,j}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,j},\dots))}\right)$: Due to (ii) and the conformity of $\mathrm{AT}(\mathbb{X})$, E has to be a face of an element in $T_{i,j}\left(\mathrm{bnd}\,(X_i)\,|_{T_{i,j}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,j},\dots))}\right)$. However, $T_{i,j}\left(\mathrm{bnd}\,(X_i)\,|_{T_{i,j}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,j},\dots))}\right)$ is face-complete which is a contradiction to the assumption.

□

The dimension of instance interface elements is always less than the dimension of the original structure. In particular, an element in the instance interface of k different instances has a dimension of at most $\mathrm{DIM}(\Lambda) - k + 1$. Every template has a trivial instance interface with itself: $\mathrm{interf}(\mathbb{X},T_{i,g},T_{i,g}) = \mathrm{inst}(\mathbb{X},i,g)$. A template might also have additional interfaces with itself using different transformation functions. For example, in a templated structure based on a reflection, the template has an instance interface with itself using different transformation functions (cf. Figure 4.5).

Two instances $\mathrm{inst}(\mathbb{X},i,g)$ and $\mathrm{inst}(\mathbb{X},j,h)$ are said to be neighbors, if their geometry instance interface is not empty: $\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,g},T_{j,h}) \neq \emptyset$. Two templates are said to be neighbors, if there are instances $\mathrm{inst}(\mathbb{X},i,g)$ and $\mathrm{inst}(\mathbb{X},j,h)$ which are neighbors.

Using this neighborhood definition, a graph can be created, where each node is a template and two nodes are connected, if they are neighbors. Because two templates can have multiple different neighboring instances, this graph is in fact a multi-graph, where each edge represents the combination of the transformation functions which form the neighboring instance interface. This graph, called the instance graph of the templated structure, is reflexive (every template is neighbor of itself) and undirected (if template $\mathbb{X}_i$ is a neighbor of $\mathbb{X}_j$, then $\mathbb{X}_j$ is also a neighbor of $\mathbb{X}_i$). Figure 4.6 shows an example of an instance graph of a templated structure.

The requirement for a templated mesh given in Definition 4.4, being $\mathrm{AT}(\mathbb{X})$ to be a valid conforming multi-region mesh, is hard to evaluate in practice. Therefore, more practicable lemmas and instruments are given in this section to verify, if a templated structure with mesh templates is a templated mesh.

Lemma 4.2 (Interface conformity Lemma)**.** *Let Γ be a templated structure with mesh templates. Γ is a templated mesh, i.e., $\mathrm{AT}(\Gamma)$ is conforming, if for all neighboring instances $\mathrm{inst}(\Gamma, i, g)$ and $\mathrm{inst}(\Gamma, j, h)$, the following holds:*

(i) The underlying space of the instance interface is equal to their geometry instance interface: $\mathrm{us}(\mathrm{interf}(\Gamma, T_{i,g}, T_{j,h})) = \mathrm{interf}_{\mathrm{geo}}(\Gamma, T_{i,g}, T_{j,h})$.

(ii) The boundary elements of both templates transformed to the instance interface are the same: $T_{i,g}(\mathbb{IP}_{T_{i,g},T_{j,h}}) = T_{j,h}(\mathbb{IP}_{T_{j,h},T_{i,g}})$*, where*

$$\mathbb{IP}_{T_{i,g},T_{j,h}} := \mathrm{bnd}(\mathrm{templ}(\Gamma, i))|_{T_{i,g}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\Gamma,T_{i,g},T_{j,h}))}. \tag{4.8}$$

Requirement (i) ensures that the transformed boundary mesh of a template respects the corresponding geometry instance interface; requirement (ii) states that for every two neighboring mesh templates, the mesh instance interface is conforming.

Proof. Every mesh template Γ_i is already conforming, so non-conformities can only occur between two elements $A, B \in \mathrm{AT}(\Gamma)$ which are in different mesh instances. Let A and B be any two elements in the structure instance $\mathrm{AT}(\Gamma)$ where A is in the instance of template X_i with transformation function $T_{i,g}$ and B is in the instance of template X_j with a transformation function $T_{j,h}$. Due to Requirement 4.2 of Definition 4.3, the intersection of A and B is either empty (if X_i and X_j are no neighbors) or included in the geometry instance interface of the instances. If their intersection is not empty, Requirement (i) ensures that both A and B are also included in the geometry instance interface. Due to (ii), A and B are in both interfaces and their intersection is a face of both. Therefore, $\mathrm{AT}(\Gamma)$ is conforming. □

4.3 Boundary Patch Partition

Let Γ be a templated structure with mesh templates. According to Lemma 4.2, Γ requires the conformity of two neighboring templates for Γ to be a templated mesh. However, as visualized in Figure 4.7, a templated structure might induce dependencies between two templates which are not neighboring. If there is such a connection between two templates, these templates are said to be indirect neighbors. Any algorithm which only relies on Lemma 4.2 might induce additional non-conformities at interfaces when trying to fix a non-conformity between two direct neighboring templates. Therefore, there is no guarantee that such an algorithm terminates. An approach to resolve this issue is the boundary patch approach presented in the following.

A mapping between the boundary of two neighboring templates X_i and X_j with transformation function $T_{i,g}$ and $T_{j,h}$ can be defined.

Definition 4.7 (Template boundary mapping)**.** Let $\mathbb{X}$ be a templated structure, X_i and X_j be two neighboring templates with transformation functions $T_{i,g}$ and $T_{j,h}$: The template boundary mapping from template X_j to template X_i is defined as the composition of $T_{j,h}$ and the inverse of $T_{i,g}$:

$$\mathbb{T}_{T_{i,g},T_{j,h}} := T_{i,g}^{-1} \circ T_{j,h}|_{T_{j,h}^{-1}(\mathrm{interf}_{\mathrm{geo}}(\mathbb{X},T_{i,g},T_{j,h}))} \tag{4.9}$$

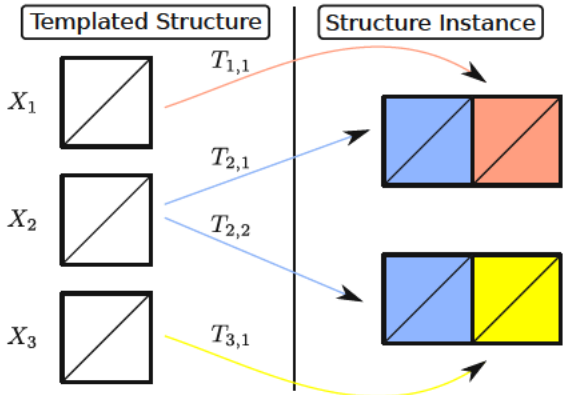


Figure 4.7: Indirect neighbors

Although template X_1 and X_3 are not neighboring, the templated structure induces a dependence between those two templates because of the two instances of template X_2. If, for example, a new vertex is inserted on the left edge of template X_1, a vertex has to be inserted on the right of template X_2, because the interface $\text{interf}(T_{1,2}, T_{2,1})$ has to be conforming. In turn, this vertex insertion in template X_2 requires a new vertex on the left side of template X_3 to ensure the conformity of interface $\text{interf}(T_{2,2}, T_{1,3})$. Therefore, template X_1 and X_3 are indirect neighbors.

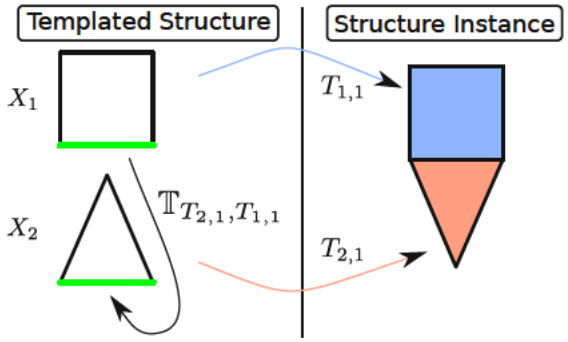


Figure 4.8: Template boundary mapping

The template boundary mapping $\mathbb{T}_{T_{2,1},T_{1,1}}$ maps from a subset of the boundary of template X_1 to a subset of the boundary of template X_2.

The template boundary mapping maps from $\text{dom}(\mathbb{T}_{T_{i,g},T_{j,h}}) = T_{j,h}^{-1}(\text{interf}_{\text{geo}}(\mathbb{X}, T_{i,g}, T_{j,h})) \subseteq \text{bnd}(\text{templ}(\mathbb{X}, j))$ to $\text{img}(\mathbb{T}_{T_{i,g},T_{j,h}}) = T_{i,g}^{-1}(\text{interf}_{\text{geo}}(\mathbb{X}, T_{i,g}, T_{j,h})) \subseteq \text{bnd}(\text{templ}(\mathbb{X}, i))$ and therefore maps boundary elements of one template to another neighboring template as visualized in Figure 4.8. For every template boundary mapping $\mathbb{T}_{T_{i,g},T_{j,h}}$ the inverse is also a template boundary mapping: $\mathbb{T}^{-1}_{T_{i,g},T_{j,h}} = \mathbb{T}_{T_{j,h},T_{i,g}}$. Template boundary mappings can also be composed to map from a template to any indirect neighbor template as shown in Figure 4.9. The set of all template boundary mappings and all valid compositions of template boundary mappings of a templated structure $\mathbb{X}$ is denoted as $\mathfrak{T}_{\mathbb{X}}$.

Using the template boundary mappings (and all compositions of template boundary mappings), a partition of the template boundary is generated. The goal of this partition is to reflect the dependencies of a template with all direct and indirect neighbor templates. The boundary patch partition, visualized in Figure 4.10, is the coarsest partition of the template boundary, where each element of the partition has the same direct and indirect neighbors.

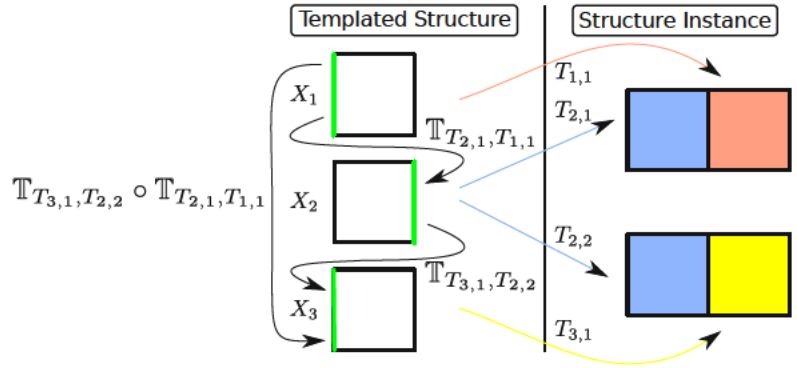


Figure 4.9: Composition of template boundary mappings

The composition of template boundary mappings $\mathbb{T}_{T_{3,1},T_{2,2}} \circ \mathbb{T}_{T_{2,1},T_{1,1}}$ directly maps from a subset of the boundary of template X_1 to a subset of the boundary of template X_3, although X_1 and X_3 do not share a common instance interface. If the templates X_1 and X_3 are both centered and oriented in the same direction, the composition $\mathbb{T}_{T_{3,1},T_{2,2}} \circ \mathbb{T}_{T_{2,1},T_{1,1}}$ is the identity.

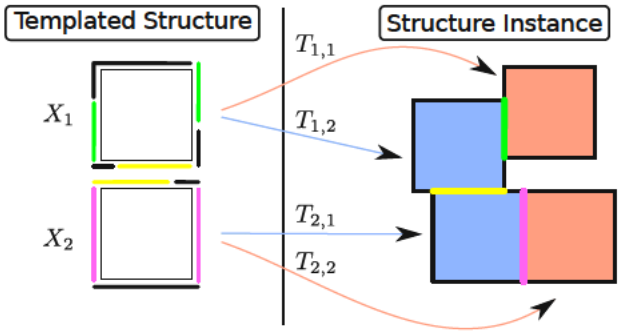


Figure 4.10: Boundary patch partition

Boundary patch partition elements, which are induced by an instance interface, are visualized in green, yellow, and purple. Boundary patch partition elements (black) are elements which are not in any instance interface. The boundary of template X_1 is decomposed into six elements, two induced by the instance interface $\text{interf}(T_{1,1}, T_{1,2})$ (green), one induced by the instance interface $\text{interf}(T_{1,2}, T_{2,1})$ (yellow), and three elements which cover the rest of the boundary. The boundary of template X_2 is decomposed into five elements, two induced by the instance interface $\text{interf}(T_{2,1}, T_{2,2})$ (purple), one induced by the instance interface $\text{interf}(T_{1,2}, T_{2,1})$, and two remaining elements which cover the rest of the boundary. All points within any boundary patch partition element have the same (direct and indirect) *neighbor configuration*. For example, for every point in the yellow boundary patch partition element, the template X_1 is on one side and the template X_2 is on the other side.

Algorithm 4.1 generates the boundary patch partition of a geometry template. At first (Line 3), all template boundary mappings and their compositions are obtained which map from the input template to any other template (including the input template). Then, the initial boundary patch partition of the template $\mathbb{B}_T$ is set to the boundary of the template (Line 4). For each template boundary mapping f obtained in Line 3, a trivial template boundary partition is generated by using the domain of f and its *boundary complement* (Line 6). This trivial partition is then refined with the boundary patch partition $\mathbb{B}_T$ (Line 7).

The boundary patch partition of a templated geometry Λ is defined as the union of the boundary patch partitions of all templates: $\text{bndpart}(\mathbb{X}) := \bigcup_{G \in \text{templ}(\mathbb{X})} \text{bndpart}(\mathbb{X}, G)$. For every template boundary mapping, its domain can be represented as a union of boundary patch partition elements. This is also true for all compositions of template boundary mappings. Using the template boundary mapping, a binary relation of two boundary patch partition elements can be defined in the following way:

```
1 Algorithm bndpart
  input  : templated geometry Λ, geometry template T
  output : boundary patch partition 𝔹_T of template T
2 begin
3 |  F_T ← {f ∈ 𝔉_Λ | dom(f) ⊆ T}
4 |  𝔹_T ← {bnd*(T)}
5 |  foreach f ∈ F_T do
6 |  |  B_f ← {dom(f), cl(bnd*(T) \ dom(f))}
7 |  |  𝔹_T ← refine(𝔹_T, b_f)
8 |  end
9 end
```

Algorithm 4.1: Generation of boundary patch partition

Definition 4.8 (Boundary Patch Relation)**.** Let $\mathbb{X}$ be a templated geometry, B_1 a boundary patch partition element of the template X_i, and B_2 a boundary patch partition element of the template X_j: B_1 is said to be related to B_2 (using a binary relation $R_{\mathbb{X}}$), if there is a template boundary mapping, which maps B_1 to B_2:

$$B_1 \sim_{\mathbb{X}} B_2 \Leftrightarrow \exists \mathbb{T}_{T_{i,g},T_{j,h}} : B_1 = T(B_2) \tag{4.10}$$

The boundary patch relation $\sim_{\mathbb{X}}$ is defined as the transitive completion of $R_{\mathbb{X}}$. The boundary patch relation of a templated mesh Γ is defined the same way except that template boundary elements are used instead of boundary patch partition elements.

Lemma 4.3 (Boundary patch relation is an equivalence relation)**.** *The boundary patch relation is an equivalence relation.*

Proof. $\sim_{\mathbb{X}}$ is reflexive because $\mathbb{T}_{T_{i,g},T_{i,g}} = \mathbb{I}$ is a valid template boundary mapping. $\sim_{\mathbb{X}}$ is symmetric, because $\mathbb{T}^{-1}_{T_{i,g},T_{j,h}} = \mathbb{T}_{T_{j,h},T_{i,g}}$. $\sim_{\mathbb{X}}$ is transitive by definition. Therefore, $\sim_{\mathbb{X}}$ is an equivalence relation. □

Because the boundary patch relation is an equivalence relation, it naturally has equivalence classes $[a]_{\sim_{\mathbb{X}}} := \{x \in \text{bndpart}(\mathbb{X}) | a \sim_{\mathbb{X}} x\}$ and a quotient set: $\text{bndpart}(\mathbb{X})/\sim_{\mathbb{X}} := \{[x] | x \in \text{bndpart}(\mathbb{X})\}$. In the example given in Figure 4.10, boundary patch partition elements which are related to each other are visualized in the same color. Black elements are just related to themselves. The quotient set of that boundary patch relation includes every black element and one representative for each color (green, yellow, and purple).

The boundary patch partition and the boundary patch relation have the following properties:

Lemma 4.4 (Template boundary mapping properties)**.** *Let Γ be a templated mesh:*

(i) For every two elements A and B of the same equivalence class there is a composition of template boundary mappings T, which maps A to B: $B = T(A)$.

(ii) Each template boundary mapping $\mathbb{T}_{T_{i,g},T_{j,h}}$ maps all boundary mesh template elements which are included in the domain of $\mathbb{T}_{T_{i,g},T_{j,h}}$ to boundary mesh template elements which are included in the image of $\mathbb{T}_{T_{i,g},T_{j,h}}$:

$$\mathbb{T}_{T_{i,g},T_{j,h}} \left(\text{bnd}\left(X_i\right) |_{\text{dom}\left(\mathbb{T}_{T_{i,g},T_{j,h}}\right)} \right) = \text{bnd}\left(X_j\right) |_{\text{img}\left(\mathbb{T}_{T_{i,g},T_{j,h}}\right)} \tag{4.11}$$

(iii) Every template boundary $\text{bnd}(X_i)$ of Γ respects the boundary patch partition of the corresponding geometry template $\text{geo}(X_i)$.

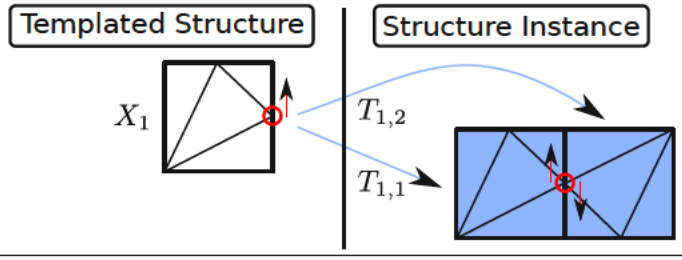


Figure 4.11: Issues due to boundary template mapping incompatibility

$T_{1,1}$ is a simple translation and $T_{1,2}$ is a combination of rotation and translation. If, for example, the template vertex highlighted by the red circle is moved up on the template boundary it will move up in the instance $T_{1,1}$ but down in the instance $T_{1,2}$ and therefore the structure instance is no longer conforming.

Proof.

(i) If A and B are in the same equivalence class, there are sets $C_1, \ldots, C_k$ for which there are boundary mappings which map A to C_1, C_1 to C_2, ..., and C_k to B. Therefore, their composition maps A to B.

(ii) Follows from Lemma 4.1 (iii).

(iii) For each composition of template boundary mappings T, which's domain is in template X_i, iteratively applying (ii) indicates that $\mathrm{bnd}(X_i)$ respects $\mathrm{dom}(T)$. From Lemma 2.3 follows, that $\mathrm{bnd}(X_i)$ also respects the boundary patch partition.

□

Due to these properties, the boundary patch partition of a templated geometry plays a central role in templated mesh generation and adaptation algorithms (cf. Chapter 5).

Two different compositions of template boundary mappings might be *incompatible*, as visualized in Figure 4.11. Formally, such an *incompatibility* occurs, when two valid compositions of template boundary mappings are different in an instance interface. In that case, the instance graph of the templated structure is called *irregular*.

Definition 4.9 (Regular/irregular instance graph)**.** Let $\mathbb{X}$ be a templated structure. The instance graph of $\mathbb{X}$ is called regular, if every two composition of template boundary mappings with a common domain are equal (restricted to the common domain):

$$\forall T, U \in \mathfrak{T}_{\mathbb{X}} \text{ with } \mathrm{dom}(T) \cap \mathrm{dom}(U) \neq \emptyset : T|_{T^{-1}(\mathrm{img}(U))} = U|_{U^{-1}(\mathrm{img}(T))} \tag{4.12}$$

Otherwise, the instance graph is called irregular.

In the example given in Figure 4.11, $\mathbb{T}_{T_{1,1},T_{1,2}}$ is not equal to $\mathbb{T}_{T_{1,1},T_{1,1}} = \mathbb{I}$ and therefore the instance graph is not regular. However, a non-trivial interface of two instances having the same template is not automatically problematic. For example, if $T_{1,2}$ is a combination of translation and reflection (similar to the example given in Figure 4.5), the instance graph is regular because $\mathbb{T}_{T_{1,1},T_{1,2}} = \mathbb{T}_{T_{1,1},T_{1,1}} = \mathbb{I}$.

From the definition above follows, that for every two representatives of an equivalence class (of the quotient set of the boundary patch relation of a templated structure with a regular instance graph) every composition of template boundary mappings is equal:

$$\forall [x]_{\sim_{\mathbb{X}}} \in \mathrm{bndpart}(\mathbb{X})/\sim_{\mathbb{X}} : \forall A, B \in [x]_{\sim_{\mathbb{X}}} : \forall F_1, F_2 \in \mathrm{tf}(\mathbb{X}, A, B) : F_1|_B = F_2|_B \tag{4.13}$$

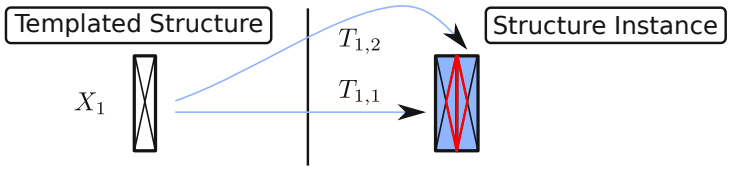


Figure 4.12: Issues with the Delaunay property of templated meshes

Although the mesh template X_1 is Delaunay, the structure instance on the right is not because of the two triangles visualized in red.

For every two elements of the boundary patch partition A, B, which are related ($A \sim_{\mathbb{X}} B$), all compositions of template boundary mappings $F \in \mathrm{tf}(\mathbb{X}, A, B)$ have to fulfill $A = F(B)$. However, for templated structures with an irregular instance graph, there are compositions of boundary template mappings F_1 and F_2, which are not equal: $F_1|_B \neq F_2|_B$. Because of the definition of the boundary patch partition, B is equal to $F_1^{-1}(F_2(B))$ but at the same time $F_1^{-1} \circ F_2$ is not the identity. Consequently, $F_1^{-1} \circ F_2$ has to be a symmetry for B. Although this is not an issue for templated geometries, it induces additional constraints for templated meshes. Algorithms for templated meshes with irregular instance graphs are therefore more challenging.

4.4 Template-Aware Delaunay

The Delaunay property (cf.Section 2.3) is a powerful property for simplex meshes. It is therefore of interest to investigate this property for templated meshes as well.

A simplex element S, which is Delaunay in the structure instance of a templated mesh, is also Delaunay in all of its mesh templates. However, the converse is generally not true. A simplex element $S \in \Gamma_i$, which is Delaunay in the mesh template $T_{i,j}(E)$, is not necessarily Delaunay in the structure instance as visualized in Figure 4.12. As shown in Lemma 2.5, global locally Delaunay is equivalent to global Delaunay. Therefore, elements, which are not locally Delaunay in the structure instance of a templated mesh with all mesh templates being Delaunay, have to be in an instance interface. To identify these elements, the locally Delaunay property is defined for templated meshes.

Definition 4.10 (Template-aware locally Delaunay)**.** Let Γ be a templated mesh: A simplex in a mesh template is called template-aware locally Delaunay, if $T_{i,j}(E)$ is locally Delaunay (in $\mathrm{AT}(\Gamma)$) for all transformation functions $T_{i,j}$ of that mesh template.

With this definition, the following Lemma can be formulated.

Lemma 4.5 (Template-aware Delaunay Lemma)**.** *Let Γ be a templated mesh. If all facets of all mesh templates of Γ are template-aware locally Delaunay, then $\mathrm{AT}(\Gamma)$ is Delaunay.*

Proof. Every facet in the structure instance is locally Delaunay because of the template-aware locally Delaunay property of every template facet. Therefore, the structure instance is Delaunay due to Lemma 2.5. □

In other words, if every mesh template of a templated mesh is Delaunay and every template boundary element is template-aware locally Delaunay, then $\mathrm{AT}(\Gamma)$ is Delaunay. An algorithm which adapts a templated mesh to be Delaunay is presented in Section 5.2.5.

4.5 Mesh Data Structures for Templated Meshes

Similar to Section 2.4, mesh data structures for templated meshes are presented in this section. Let Γ be a templated n-dimensional mesh. A trivial data structure approach is to simply store an array of meshes with each having a the set of transformation functions and a set of region IDs. However, this approach has the issue, that vertices, which are shared between two instances, cannot be detected in a stable way, because numerical tests are required. Keeping track of these vertex connections requires additional bookkeeping which in turn results in higher memory requirements. Therefore, two different data structures are presented, one using the simple approach and one with shared vertex bookkeeping.

Listing 4.1: Templated mesh data structure without shared vertex bookkeeping

```
struct TRANSFORMATION
{
  NUMERIC_TYPE          matrix[];
  NUMERIC_TYPE          translation[];
};

struct MESH_TEMPLATE
{
  MESH                  mesh;

  int                   instance_count;
  int                   transformation_indices[];
  REGION_ID_TYPE        region_ids[];
};

struct TEMPLATED_MESH
{
  int                   template_count;
  MESH_TEMPLATE         templates[];

  int                   transformation_count;
  TRANSFORMATION        transformations[];
};
```

The data structure in Listing 4.1 represents a templated mesh without shared vertex bookkeeping. The mesh data structure presented in Section 2.4 is reused for the mesh templates. Each rigid transformation function can be represented using an $n \times n$ matrix for the affine part of the transformation and an n-dimensional translation vector. To save additional memory, transformations in $\mathbb{R}^3$ can be represented using unit quaternions [69] instead of the transformation matrix and in $\mathbb{R}^2$, a single angle can be used. However, the data structure will be different for meshes in $\mathbb{R}^2$ and $\mathbb{R}^3$. The affine matrix and translation vector representation is used in this work. All transformation functions are stored in an array located in the `TEMPLATED_MESH` structure. For each mesh instance, the index of the corresponding transformation function is stored in the `transformation_indices` array and the region indicator is stored in the `region_ids` array. The total size of the templated mesh equals the sum of the size of all templates and all transformation functions. The additional memory space required for the pointers of transformation functions and all count variables is negligible.

Listing 4.2: Templated mesh data structure with shared vertex bookkeeping

```
1
2  struct MESH_TEMPLATE_SVB_INSTANCE
3  {
4    REGION_ID_TYPE                      region_id;
5    int                                 transformation_index;
6
7    int                                 local_to_global_vertex_mapping_count;
8    int                                 local_to_global_vertex_mapping[];
9  };
10
11 struct MESH_TEMPLATE_SVB
12 {
13   MESH                                mesh;
14
15   int                                 instance_count;
16   MESH_TEMPLATE_SVB_INSTANCE          instances[];
17 };
18
19 struct TEMPLATED_MESH_SVB
20 {
21   int                                 template_count;
22   MESH_TEMPLATE_SVB                   templates[];
23
24   int                                 transformation_count;
25   TRANSFORMATION                      transformations[];
26
27   int                                 global_vertex_count;
28   NUMERIC_TYPE                        global_vertices[];
29 };
```

The data structure in Listing 4.2 stores a templated mesh with additional shared vertex bookkeeping (SVB). Shared vertex connections are represented using an array of *global* vertices which are all vertices shared by at least two mesh instances. In other words, the `global_vertices` array holds all vertices which are in any mesh instance interface. The size of the `global_vertices` array therefore is equal to the number of *global* interface vertices times the geometric dimension n. For each mesh instance and for all vertices which are shared with any neighboring mesh instance, there is an additional mapping from the *local* mesh template vertices to the *global* vertices. If the i-th vertex of the template of a mesh instance is shared with another mesh, the pair (i, j) is inserted in the `local_to_global_vertex_mapping` array of the corresponding mesh instance, where j is the index of the global vertex in the `global_vertices` array. Using this mapping, any vertex which is shared can be uniquely identified without any numerical tests. For each local-to-global mapping, two integers, being the two vertex indices, are stored. The size of the `local_to_global_vertex_mapping` array therefore is equal to the number of shared local vertices times $2 \times \text{sizeof(int)}$. The size of a mesh instance does not only depend on n, but also on the number of shared vertices with other mesh instances. The transformation functions are stored similar to the data structure without SVB, i.e., in a global array in the `TEMPLATED_MESH_SVB` structure. Each mesh instance stores the array index of the corresponding transformation function in the `transformation_index` variable.

The total size of a templated mesh with SVB is the sum of all mesh templates, all instances, all transformation functions, and the size of the `global_vertices` array. All other data types are negligible for sufficiently large meshes. An example for a templated mesh stored with both data structures is visualized in Figure 4.13. It can be seen, that the data structure with SVB requires more memory than the data structure without SVB. A comparison of memory usage of templated meshes is given in Chapter 7.

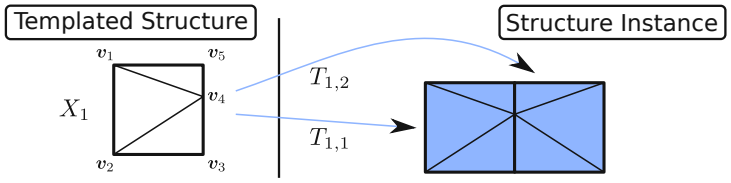


Figure 4.13: Templated mesh stored with and without SVB

The templated mesh has two transformation functions ($T_{1,1}$ being a translation and $T_{1,2}$ being a combination of reflection and translation) where the three vertices v_3, v_4, and v_5 are shared by an instance interface. When using the data structure without SVB (Listing 4.1), only the mesh template X_1 is stored as described in Section 2.4. `transformation_indices` would be $(0, 1)$ and `region_ids` would be equal to $(0, 0)$. The two transformation functions, $T_{1,1}$ and $T_{1,2}$, are stored in the `transformations` array. When using the data structure with SVB (Listing 4.2), the mesh template X_1 as well as the transformations functions are stored the same way. The `global_vertices` array would include $T_{1,1}(v_3)$, $T_{1,1}(v_4)$, and $T_{1,1}(v_5)$. Therefore, the `local_to_global_vertex_mapping` array of the instance indicated by $T_{1,1}$ is equal to $((3, 1), (4, 2), (5, 3))$, because the local vertex v_3 is associated with the global vertex $T_{1,1}(v_3)$ which is the first entry in the `global_vertices` array, v_4 with $T_{1,1}(v_4)$, and v_5 with $T_{1,1}(v_4)$. For similar reasons, the `local_to_global_vertex_mapping` array of the instance indicated by $T_{1,2}$ is also $((3, 1), (4, 2), (5, 3))$.

Chapter 5

Algorithms for Templated Geometries and Meshes

Mesh generation and adaptation algorithms for conventional meshes have been presented in Chapter 3. This chapter covers algorithms for generating and adapting templated meshes. Templated mesh generation based on templated geometries is covered in Section 5.1. Algorithms for templated mesh adaptation are discussed in Section 5.2.

5.1 Mesh Generation

In this section two different algorithms for generating a templated mesh based on a templated geometry are proposed. The first algorithm, presented in Section 5.1.1, individually generates a mesh for each geometry template and then eliminates potential non-conformities in instance interfaces. The second algorithm, presented in Section 5.1.2, utilizes the boundary patch partition to first generate conforming surface meshes for each template and then generating volumetric mesh templates from these surface meshes.

5.1.1 Independent Mesh Generation and Interface Merging

The algorithms presented in this section start with already generated mesh templates which do not necessarily yield conforming instance interfaces (cf. Figure 4.4). To obtain a valid templated mesh, potential non-conformities have to be fixed. The process of making these interfaces conforming is called *interface sewing* and consists of two steps: Vertex merging and refinement.

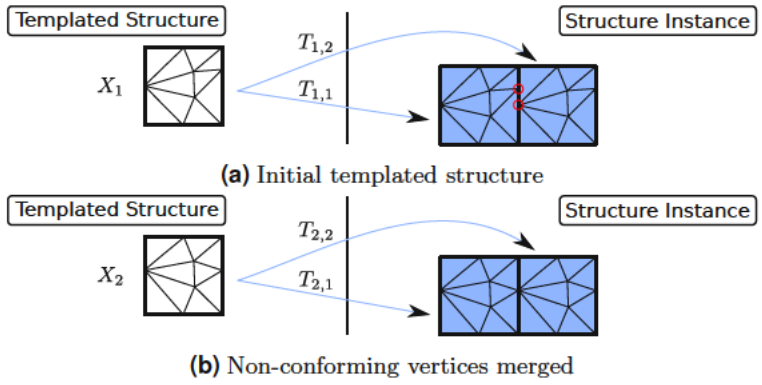
The first algorithm, being vertex merging, eliminates non-conformities in instance interfaces by identifying vertices which are close to each other and merging them. This vertex merging algorithm requires an auxiliary algorithm, Algorithm 5.1, which creates a tuple of template vertex pairs W which are not related to each other (Line 6), are close to each other (Line 7), and which are in a common geometry instance interface (Line 8). The relation used in Line 6 represents the boundary patch relation. Each element of the tuple is a tuple itself and includes the distance between the vertices, the template vertices, the template index, and the transformation functions (Line 9). In the end, the tuple W is sorted in ascending order by the first argument, i.e., the vertex distance (Line 13).

Algorithm 5.2 shows the vertex merging algorithm. The maximal distance between two vertices is specified with the parameter $d_{\max}$. At first, the helper variables V (the set of all vertices which are in any instance interface), $\sim_V$ (a relation on V where two vertices are related to each other, if there is a template boundary mapping which maps one to the other), and $\mathbb{T}_{\boldsymbol{x},\boldsymbol{y}}$ (the template boundary mappings which are used to create $\sim_V$) are set up. The set V represents all template vertices which are included in any geometry instance interface (Line 3).

Algorithm obtain_close_vertex_pairs
input : templated structure Γ with mesh templates, maximal vertex distance $d_{\max}$, template vertex relation $\sim_V$
output : sorted tuple W of vertices in instance interfaces which are close to each other

begin
  $W \leftarrow \emptyset$
  foreach $T_{i,g}, T_{j,h} \in \mathrm{tf}(\Gamma)$ **do**
    foreach $v \in \mathrm{elem}_0(\mathrm{templ}(\Gamma, i)), w \in \mathrm{elem}_0(\mathrm{templ}(\Gamma, j))$ **do**
      if $v \approx_V w$ **and**
        $\|T_{i,g}(v) - T_{j,h}(w)\|_2 \leq d_{\max}$ **and**
        $\{T_{i,g}(v), T_{j,h}(w)\} \subseteq \mathrm{interf}_{\mathrm{geo}}(\Gamma, T_{i,g}, T_{j,h})$ **then**
          $W \leftarrow W \cup (\|T_{i,g}(v) - T_{j,h}(w)\|_2, v, w, i, T_{i,g}, j, T_{j,h})$
      end
    end
  end
  sort W ascending by the first argument
end

Algorithm 5.1: Obtain close vertex pairs



(a) Initial templated structure

(b) Non-conforming vertices merged

Figure 5.1: Eliminating non-conformities by vertex merging

The initial structure instance of the templated structure has two non-conformities highlighted in red (a). These two vertices are merged to their center in the output templated structure on the bottom resulting in a conforming structure instance (b).

```
1  Algorithm fix_nonconformities_vertex_merging
   input  : templated structure Γ with mesh templates, maximal vertex distance d_max
   output : templated structure Γ
2  begin
3      V ← ⋃_i {v ∈ elem_0(templ(𝕏, i)) | v is in any geometry instance interface}
4      foreach v, w ∈ V do
5          if ∃T ∈ 𝔗_Γ : v = T(w) then
6              Set v ∼_V w and w ∼_V v
7              𝕋_{v,w} ← T, 𝕋_{w,v} ← T^{-1}
8          end
9      end
10     W_1 ← obtain_close_vertex_pairs(Γ, d_max, ∼_V)
11     n ← 1
12     repeat
13         for k ← 1 to |W_n| do
14             (d, v, w, i, T_{i,g}, j, T_{j,h}) ← W_k[k]
15             z ← (v+w)/2,   w_v ← T_{i,g}^{-1}(w),   w_w ← T_{j,h}^{-1}(w)
16             movement_valid ← true
17             if moving v to w_v is not valid or moving w to w_w is not valid then
18                 movement_valid ← false
19             end
20             foreach x ∈ V : x ∼_V v do
21                 if moving x to 𝕋_{x,v}(z_v) is not valid then
22                     movement_valid ← false
23                 end
24             end
25             foreach x ∈ V : x ∼_V w do
26                 if moving x to 𝕋_{x,w}(z_w) is not valid then
27                     movement_valid ← false
28                 end
29             end
30             if movement_valid then
31                 v ← z_v, w ← z_w
32                 foreach x ∈ V : x ∼_V v do
33                     x ← 𝕋_{x,v}(z_v)
34                 end
35                 foreach x ∈ V : x ∼_V w do
36                     x ← 𝕋_{x,w}(z_w)
37                 end
38                 foreach x ∈ V : x ∼_V v do
39                     foreach y ∈ V : y ∼_V w do
40                         Set x ∼_V y
41                         𝕋_{y,x} ← 𝕋_{y,w} ∘ 𝕋_{v,x}, 𝕋_{x,y} ← 𝕋_{x,v} ∘ 𝕋_{w,y}
42                     end
43                 end
44                 goto Line 47
45             end
46         end
47         W_n ← obtain_close_vertex_pairs(Γ, d_max, ∼_V)
48         n ← n + 1
49     until W_n = W_{n-1}
50 end
```

Algorithm 5.2: Fix non-conformities by vertex merging

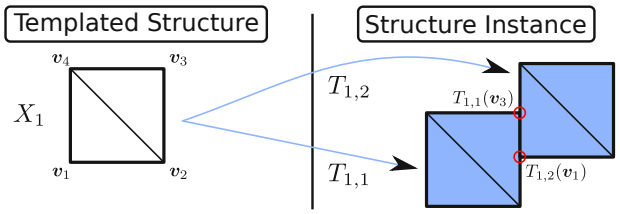


Figure 5.2: Vertex merging fails to fix all non-conformities

The structure instance has two non-conformities highlighted by the red circles. Although the vertices $T_{1,1}(v_3)$ and $T_{1,2}(v_1)$ can be considered close to each other, neither v_1 nor v_3 can be moved within template X_1 because it would break the geometry-conformity.

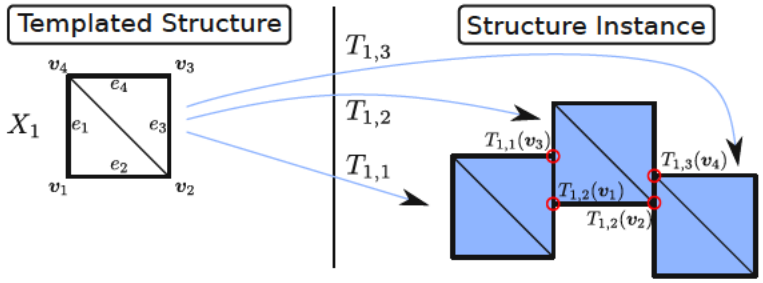
The binary relation $\sim_V$ represents the boundary patch relation, meaning that it links all template vertices which can be mapped to each other by a composition of template boundary mappings (Lines 6). The corresponding composition of template boundary mapping is stored in the $\mathbb{T}_{\boldsymbol{x},\boldsymbol{y}}$ variable sets (Line 7). The function obtain_close_vertex_pairs (cf. Algorithm 5.1) is used to obtain a sorted tuple W_1 of all template vertex pairs which are included in the same geometry instance interfaces and are close to each other (Line 10). The vertex merging algorithm iteratively tests template vertex pairs if they can be moved to their center (Lines 15-19). Additionally, it also evaluates if the corresponding movement of the linked template vertices is valid (Lines 20-29). A vertex movement is considered to be valid, if the movement in the mesh template does not break the validity of that mesh template (cf. Section 5.2.2). If all movements are valid, they are applied (Lines 31-37), the corresponding equivalence classes of relation $\sim_V$ are merged, and the $\mathbb{T}_{\boldsymbol{x},\boldsymbol{y}}$ variables are updated (Lines 38-43). If a template vertex pair with valid movements is found, the sorted tuple W_n is updated (Line 47) and the process starts again until the sorted tuple W_n does not change any more (Line 49). The algorithm is visualized in Figure 5.1.

Although Algorithm 5.2 reduces non-conformities, it is, in general, not able fix all non-conformities as shown in Figure 5.2. Therefore, two additional algorithms are proposed (cf. Algorithm 5.3 and Algorithm 5.4), which fix all remaining non-conformities for simplex meshes by using refinement and element splitting techniques. It is sufficient to only handle vertex-in-facet non-conformities for triangle meshes (Algorithm 5.3). However, for tetrahedron meshes, non-conformities induced by line-line-intersections have to be handled as well (Algorithm 5.4).

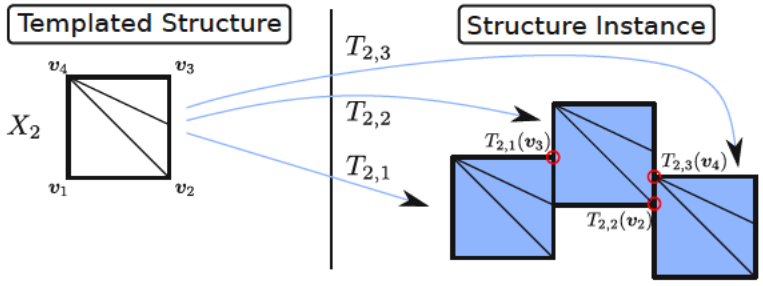
At first, Algorithm 5.3 generates the set of all vertex-face pairs $(\boldsymbol{v}, f)$ in the structure instance, where their intersection is not empty and the vertex $\boldsymbol{v}$ is not a face of f – and therefore creates a non-conformity (Line 5). As long as there are such pairs, there are non-conformities which must be fixed. For each vertex-face-pair $(\boldsymbol{v}, f)$ and each template face f_T of the structure instance face f, the vertex $\boldsymbol{v}_T = T_{i,j}^{-1}(\boldsymbol{v})$ is inserted into the corresponding mesh template (Line 11). For every template face f_T, every co-face cell c_T is split into new cells by connecting $\boldsymbol{v}_T$ to all vertices of c_T. These new cells are inserted into the mesh template (Line 13) and c_T is removed (Line 14). Depending on the position of $\boldsymbol{v}_T$ within c_T, a different number of new cells will arise. If $\boldsymbol{v}_T$ is in the interior of f_T, the number of new cells is equal to the cell dimension d_c. Otherwise, the number of new cells is higher, because there are additional co-face cells of f_T. Afterwards, newly introduced non-conformities are added to V (Lines 16-22). Finally, all vertex-face pairs $(\boldsymbol{v}, f)$ in V, where f is no longer present, are removed from V (Lines 24-29). This process is repeated, until $V = \emptyset$ and therefore all vertex-in-facet non-conformities have been eliminated. Although every iteration step inserts additional vertices and therefore potentially non-conformities, Algorithm 5.3 terminates: Vertices are only inserted at locations which are accessible from another vertex via compositions of boundary template mappings, which in turn is a finite number. The algorithm is visualized in Figure 5.3.

```
1  Algorithm fix_vertex_in_facet_nonconformities
   input  : templated structure Γ with simplex mesh templates
   output : templated simplex mesh Γ
2  begin
3      d_c ← DIM_cell(Γ)
4      d_f = d_c − 1
5      V ← {(v, f) ∈ elem_0(AT(Γ)) × elem_{d_f}(AT(Γ)) : v ⊆ f ∧ v ∉ faces_0(f)}
6      while V ≠ ∅ do
7          (v, f) ← V[0]
8          foreach f_T ∈ templ_elem(Γ, f) do
9              T ← templ(Γ, f_T)
10             T_{i,j} ← tf(Γ, f, f_T)
11             insert v_T = T_{i,j}^{-1}(v) into mesh template T
12             foreach c_T ∈ cofaces_{d_c}(f_T) do
13                 insert new simplices into mesh template T by split simplex cell c_T using v_T
14                 remove c_T from mesh template T
15             end
16             foreach F ∈ tf(Γ, T) do
17                 foreach g ∈ elem_{d_f}(AT(Γ)) : F(v_T) ⊆ g ∧ F(v_T) ∉ faces_0(g) do
18                     if (F(v_T), g) ≠ V then
19                         V ← V ∪ {(F(v_T), g)}
20                     end
21                 end
22             end
23         end
24         V ← V \ {(v, f)}
25         foreach (v, f) ∈ V do
26             if f ∉ AT(Γ) then
27                 V ← V \ {(v, f)}
28             end
29         end
30     end
31 end
```

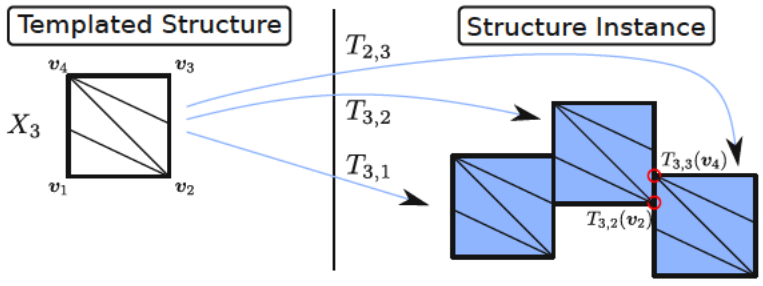
Algorithm 5.3: Fix vertex-in-facet non-conformities



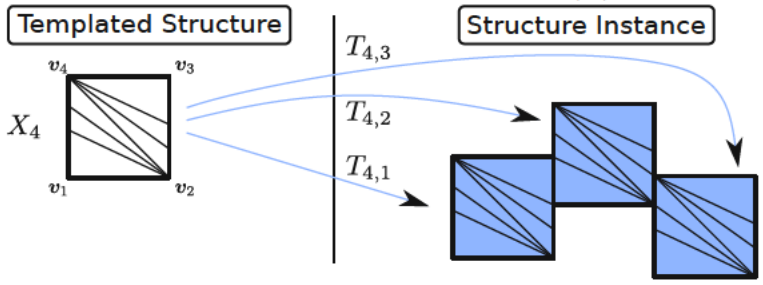
(a) Initial templated structure



(b) Fix non-conformity induced by $T_{1,2}(v_1)$



(c) Fix non-conformity induced by $T_{2,1}(v_3)$



(d) Final templated mesh

Figure 5.3: Eliminating vertex-in-facet non-conformities

The initial templated structure has four non-conformities (highlighted in red) in its structure instance which cannot be fixed using the vertex merging algorithm (a). At first, the vertex $T_{1,1}^{-1} \circ T_{1,2}(v_1)$ is inserted in the template edge e_3 resulting in the second templated structure (b). Then, $T_{2,2}^{-1} \circ T_{2,1}(v_3)$ is inserted in the template edge e_1 resulting in the third templated structure (c). Using the same approach, all other non-conformities are eliminated by transforming and inserting the vertices in the geometry instance interface to the template facet which it intersects. The structure instance of the final templated structure (d) does not have any non-conformities. However, it can be seen that the element quality is *poor*. This algorithm terminates, because vertices are only inserted at locations via a finite number of compositions of boundary template mappings.

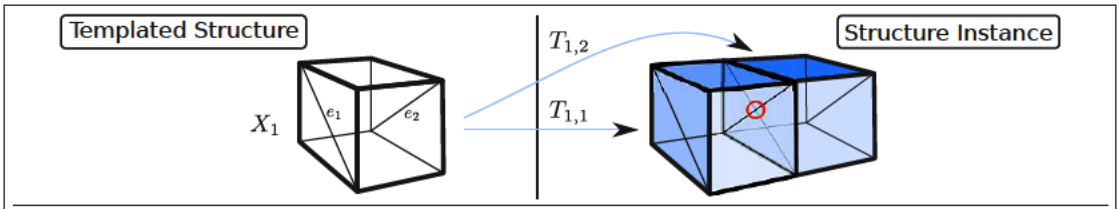


Figure 5.4: Non-conformity induced by a line-line intersection

Although every vertex in the instance interface interf$(T_{1,1}, T_{1,2})$ is conforming, the intersection of edges e_1 and e_2 induces a non-conformity.

Algorithm 5.2 is applicable to quadrilateral and hexahedral meshes without modification. However, this is not true for Algorithm 5.3. Splitting a quadrilateral or hexahedron using edge bisection requires column and sheet operations [88][89]. These operations are, in contrast to simplex splitting, non-local and therefore potentially introduce additional non-conformities which have to be taken care of. Therefore, Algorithm 5.3 has to be modified (Lines 16-22) to include these newly introduced non-conformities.

Algorithm 5.3 fixes all non-conformities of templated triangle meshes. However, for templated tetrahedron meshes, Algorithm 5.3 is not sufficient due to non-conformities induced by line-line-intersections at instance interfaces as visualized in Figure 5.4. Therefore, the algorithm for tetrahedrons requires a pre-processing step as presented in Algorithm 5.3.

```
1  Algorithm fix_line_line_nonconformities
   input  : templated structure Γ with tetrahedron mesh templates
   output : templated tetrahedron structure Γ
2  begin
3      L ← {(l1, l2) ∈ elem1(AT(Γ))^2 | l1 ∩ l2 = {x} ∧ {x} ∉ faces0(l1) ∧ {x} ∉ faces0(l2)}
4      foreach (l1, l2) ∈ L do
5          v ← l1 ∩ l2
6          foreach l ∈ {l1, l2} do
7              foreach l_T ∈ templ_elem(Γ, l) do
8                  T ← templ(Γ, l_T)
9                  T_{i,j} ← tf(Γ, l, l_T)
10                 insert v_T = T_{i,j}^{-1}(v) into mesh template T
11             end
12         end
13     end
14 end
```

Algorithm 5.4: Fix line-line-intersection non-conformities

At first, Algorithm 5.4 generates a set of line-line pairs (l_1, l_2) in the structure instance, which do intersect and where the intersection vertex is not a face vertex of both l_1 and l_2 (Line 3). For each of these pairs (l_1, l_2) the intersection vertex is calculated (line 5). This vertex is transformed to each template of l_1 and l_2 and inserted into these templates (Lines 6-12). As L is finite, this algorithm also terminates. To fix all non-conformities of a templated tetrahedron mesh, Algorithm 5.4 is applied first, followed by Algorithm 5.3.

Although Algorithm 5.3 and Algorithm 5.4 fix all non-conformities for triangle and tetrahedron meshes, they have the following issues: First, the algorithms potentially insert a lot of new cell elements at or near the instance interfaces. This might lead to element sizes which are smaller than desired. Additionally, due to the construction rule for simplices in Algorithm 5.3, the newly created cell elements have low quality. These effects can be reduced by applying Algorithm 5.2 right before Algorithm 5.3 and Algorithm 5.4, because the number of non-conformities will potentially be reduced. Another downside of the algorithms presented in this section are numerical issues with the inclusion tests in Algorithm 5.3, Line 5, or the three-dimensional line-line intersections of Algorithm 5.4, Line 3. However, these operations benefit from previous work on numerical stability for computational geometry [24][65][80][120].

For the algorithms presented in this section, arbitrary volumetric mesh generation algorithms can be used without any modification, increasing the flexibility in the templated mesh generation process. However, as mentioned, the element quality of the resulting templated meshes is potentially inferior.

5.1.2 Templated Mesh Generation Based on the Boundary Patch Partition

In contrast to the algorithms presented in the previous section, this section focuses on first generating surface mesh templates which already result in a conforming structure instance. Afterwards, a mesh generation algorithm generates the volumetric meshes of the geometry templates without touching the surface meshes. The basic idea is the following:

(i) The boundary patch partition, the boundary relation, and its quotient set of a templated geometry Λ are created.

(ii) All equivalence classes, being the elements of the quotient set, are sorted by their dimension.

(iii) Starting from the lowest dimension, surface meshes for all elements of the quotient set are generated considering already generated surface meshes.

Special care has to be taken in step (iii). Ideally, any representative of an equivalence class can be chosen for surface mesh generation and the resulting mesh can be copied to all other members of the equivalence class to obtain a conforming structure instance. However, this is not true for templated geometries with an irregular instance graph (cf. Figure 4.11).

```
1  Algorithm init_templated_mesh
   input  : templated structure Λ
   output : templated mesh Γ with empty mesh templates
2  begin
3      foreach T_Λ ∈ templ(Λ) do
4          T_Γ ← create an empty mesh template for geometry template T_Λ in Γ
5          set templ(Γ, T_Λ) := T_Γ
6          foreach F_Λ ∈ tf(Λ, T_Λ) do
7              F_Γ ← copy transformation function F_Λ to T_Γ
8              set tf(Γ, F_Λ) := F_Γ
9              copy region indicators
10         end
11     end
12 end
```

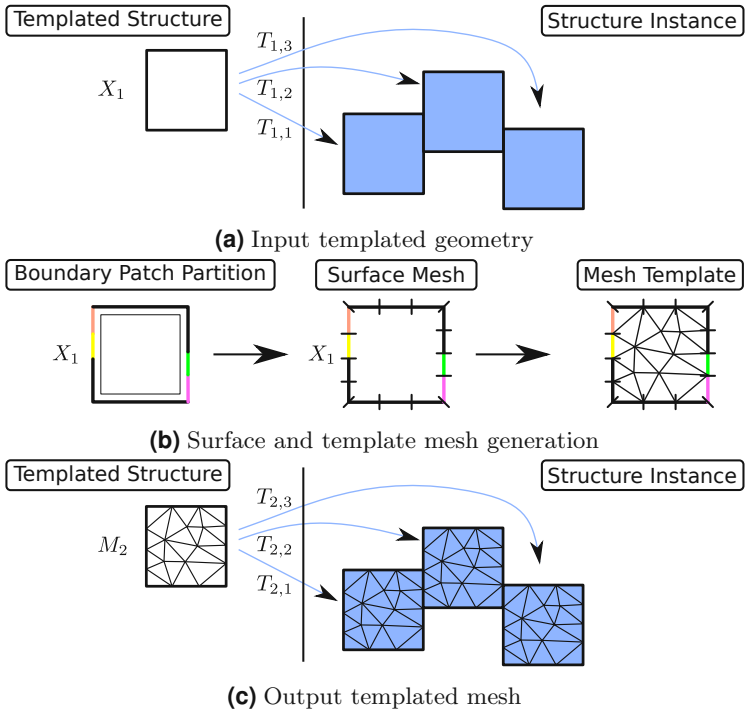
Algorithm 5.5: Initialize templated mesh

First, an auxiliary algorithm for initializing a templated mesh based on a templated geometry is presented, which initializes a templated mesh based on a templated geometry (Algorithm 5.5). For each geometry template an empty mesh template is created (Line 4) and all transformation functions and region indicators are copied (Lines 6-10).

```
1  Algorithm generate_templated_triangle_mesh
   input  : templated geometry Λ with DIM(Λ) = 2
   output : templated triangle mesh Γ which geometry-conforms to Λ
2  begin
3      Γ ← init_templated_mesh(Λ)
4      𝔅 ← bndpart(Λ)/∼_Λ
5      foreach [v] ∈ 𝔅, DIM([v]) = 0 do
6          foreach v_T ∈ templ_elem(Λ, [v]) do
7              T_Γ ← templ(Γ, templ(Λ, v_T))
8              insert v_T into T_Γ
9          end
10     end
11     foreach [l] ∈ 𝔅, DIM([l]) = 1 do
12         l_T ← any representative of [l]
13         T_Γ ← templ(Γ, templ(Λ, l_t))
14         V ← {v ∈ elem_0(T_Γ) | v ⊆ l_T}
15         create a symmetric line mesh L, where us(L) = l_T and V ⊆ elem_0(L)
16         foreach l̃_T ∈ [l] do
17             T̃_Γ ← templ(Γ, templ(Λ, l̃_T))
18             F_Γ ← tf(Γ, tf(Λ, l̃_T, l_T)[0])
19             insert F_Γ(L) into mesh template T̃_Γ
20         end
21     end
22     foreach T_Λ ∈ templ(Λ) do
23         T_Γ = templ(Γ, T_Λ)
24         create a triangle mesh M which geometry-conforms to T_Λ and bnd(M) = T_Γ
25         copy all triangles from T to T_Γ
26     end
27 end
```

Algorithm 5.6: Templated triangle mesh generation using the boundary patch partition

The templated mesh generation algorithm for 2D templated triangle meshes is presented in Algorithm 5.6. At first, the resulting templated mesh is initialized based on the templated geometry (Line 3). Afterwards, the quotient set of the boundary patch relation is calculated (Line 4). For each vertex in each vertex equivalence class, the vertex is inserted into the corresponding mesh template (Lines 5-10). Next, for each line equivalence class, a representative l_T is chosen (Line 12) and the vertices V in the corresponding mesh template, which are subsets of l_T, are determined (Line 14). In Line 15, a line mesh L, which respects V, is generated for l_T. To support templated geometries with irregular instance graphs, the line mesh L should have a reflective symmetry with a reflecting hyperplane which is orthogonal to the line l_T. The created line mesh is then inserted into the templates of the other elements of the line equivalence class (Lines 16-20). At last, a triangle mesh is generated for each mesh template using the created line meshes of the template surface (Line 24) and the resulting triangles are copied to the mesh template (Line 25). Any triangle mesh generation algorithm can be used for this step as long as it does not alter the surface. For example, the mesh generation tool Triangle offers an option which prohibits the insertion of additional vertices on the boundary [17].



(a) Input templated geometry

(b) Surface and template mesh generation

(c) Output templated mesh

Figure 5.5: Templated mesh generation using surface meshes

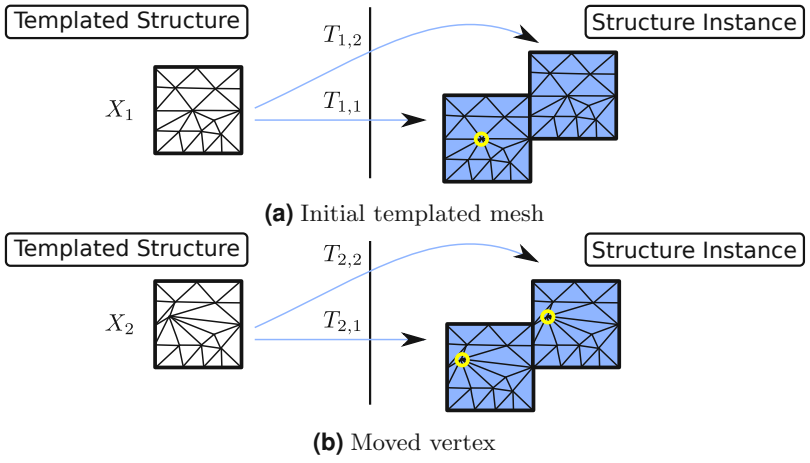
The initial templated structure consists of one template which is instanced three times (a). At first, the boundary patch partition is obtained where a surface mesh is generated for each boundary patch partition element (b). Using this surface mesh, a volumetric mesh is created using an algorithm which does not alter the surface mesh. The final templated mesh is visualized on the bottom (c).

The resulting templated output structure of this algorithm is a valid templated mesh which geometry-conforms to the templated input geometry (cf. Section 4.3). Algorithm 5.6 terminates for all valid inputs and even works for templated geometries with irregular instance graphs. In comparison to the algorithms presented in the previous section, Algorithm 5.6 has fewer issues with poor element quality for elements on or near instance interfaces, because more explicit control is given to the surface mesh generation process. Additionally, fewer numerical issues arise. For the inclusion tests, required in Line 14, information from the boundary patch generation process can be re-used to avoid explicit numerical inclusion tests. Figure 5.5 visualizes Algorithm 5.6 applied to the templated geometry presented in Figure 5.3.

Algorithm 5.7 generates 3D templated tetrahedral meshes based on templated geometries with regular instance graphs. This algorithm is similar to Algorithm 5.6. The major difference is in Line 15, where the line mesh is not required to be symmetric. In contrast to Algorithm 5.6, triangles have to be created for the boundary patches which is done in Lines 22-33. This process differs from the creation of line meshes in a way, that for the boundary patch triangle mesh generation previously created lines also have to be taken into account (Line 26). The triangle mesh generation (Line 27) is similar to the final triangle mesh generation in Algorithm 5.6 and therefore the same mesh generation algorithms and tools can be applied. Finally, tetrahedral meshes are generated for each template based on the surface meshes. Like in Algorithm 5.6, any tetrahedral mesh generation algorithm which does not alter the surface can be used here. Possible software choices are the advancing front mesh generation tool Netgen [9] or the Delaunay refinement mesh generation tool Tetgen [14] (with the input surface mesh preservation option enabled).

Algorithm generate_templated_tetrahedron_mesh
input : templated structure Λ with $\mathrm{DIM}(\Lambda) = 3$ with regular instance graph
output : templated tetrahedron mesh Γ which geometry-conforms to Λ
begin
  $\Gamma \leftarrow$ init_templated_mesh(Λ)
  $\mathfrak{B} \leftarrow \mathrm{bndpart}(\Lambda)/\sim_\Lambda$
  foreach $[\boldsymbol{v}] \in \mathfrak{B}, \mathrm{DIM}([\boldsymbol{v}]) = 0$ **do**
    foreach $\boldsymbol{v}_T \in \mathrm{templ\_elem}(\Lambda, [\boldsymbol{v}])$ **do**
      $T_\Gamma \leftarrow \mathrm{templ}(\Gamma, \mathrm{templ}(\Lambda, \boldsymbol{v}_T))$
      insert $\boldsymbol{v}_T$ into T_Γ
    end
  end
  foreach $[l] \in \mathfrak{B}, \mathrm{DIM}([l]) = 1$ **do**
    $l_T \leftarrow$ any representative of $[l]$
    $T_\Gamma \leftarrow \mathrm{templ}(\Gamma, \mathrm{templ}(\Lambda, l_T))$
    $V \leftarrow \{\boldsymbol{v} \in \mathrm{elem}_0(T_\Gamma) | \boldsymbol{v} \subseteq l_T\}$
    create a line mesh L, where $\mathrm{us}(L) = l_T$ and $V \subseteq \mathrm{elem}_0(L)$
    foreach $\widetilde{l_T} \in [l]$ **do**
      $\widetilde{T_\Gamma} \leftarrow \mathrm{templ}(\Gamma, \mathrm{templ}(\Lambda, \widetilde{l_T}))$
      $F_\Gamma \leftarrow \mathrm{tf}(\Gamma, \mathrm{tf}(\Lambda, \widetilde{l_T}, l_T)[0])$
      insert $F_\Gamma(L)$ into mesh template $\widetilde{T_\Gamma}$
    end
  end
  foreach $[p] \in \mathfrak{B}, \mathrm{DIM}([p]) = 2$ **do**
    $p_T \leftarrow$ any representative of $[p]$
    $T_\Gamma \leftarrow \mathrm{templ}(\Gamma, \mathrm{templ}(\Lambda, p_T))$
    $V \leftarrow \{\boldsymbol{v} \in \mathrm{elem}_0(T_\Gamma) | \boldsymbol{v} \subseteq p_T\}$
    $L \leftarrow \{l \in \mathrm{elem}_1(T_\Gamma) | l \subseteq p_T\}$
    create a triangle mesh M, where $\mathrm{us}(M) = p_T$, $V \subseteq \mathrm{elem}_0(M)$, and $L \subseteq \mathrm{elem}_1(M)$
    foreach $\widetilde{p_T} \in [p]$ **do**
      $\widetilde{T_\Gamma} \leftarrow \mathrm{templ}(\Gamma, \mathrm{templ}(\Lambda, \widetilde{p_T})$
      $F_\Gamma \leftarrow \mathrm{tf}(\Gamma, \mathrm{tf}(\Lambda, \widetilde{p_T}, p_T)[0])$
      insert $F_\Gamma(M)$ into mesh template $\widetilde{T_\Gamma}$
    end
  end
  foreach $T_\Lambda \in \mathrm{templ}(\Lambda)$ **do**
    $T_\Gamma = \mathrm{templ}(\Gamma, T_\Lambda)$
    create a tetrahedron mesh M which geometry-conforms to T_Λ and $\mathrm{bnd}(M) = T_\Gamma$
    copy all tetrahedrons from T to T_Γ
  end
end

Algorithm 5.7: Templated tetrahedron mesh generation using the boundary patch partition



(a) Initial templated mesh

(b) Moved vertex

Figure 5.6: Non-locality of operations in the structure instance

Because the operation is actually performed on the corresponding template vertex, the operation is not local in the structure instance.

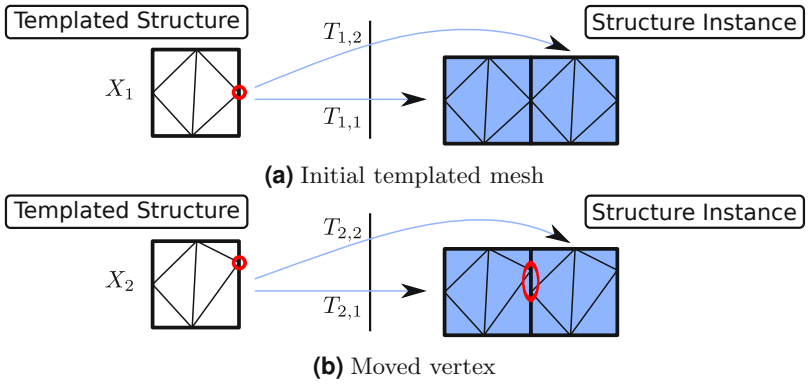
As mentioned above, Algorithm 5.7 does not support templated geometries with irregular instance graphs. To tackle this issue, the line surface mesh generated in Line 15 and the triangle surface mesh generated in Line 27 have to be symmetric. For the line surface mesh, the same approach as presented in Algorithm 5.6 can be used. However, for the triangle surface mesh, the generation process of a symmetric mesh is more complex as rotational symmetries can occur in addition to reflective symmetries. Algorithms for generating a symmetric mesh are presented in Section 6.2 and Section 6.3.

Algorithm 5.6 and Algorithm 5.7 can also be modified to generate all-quad or all-hex meshes. For all-quad meshes, the only difference in Algorithm 5.6 is the usage of the volumetric mesh generator in Line 24. For all-hex meshes, the triangle surface mesh generator and the tetrahedron volume mesh generator have to be exchanged with their all-quad and all-hex counterparts (Line 27 and Line 36). All-quad and all-hex approaches based on Paving or sweeping (cf. Section 3.1.2) are natural choices for surface and volumetric mesh generation (cf. Section 3.1.2).

5.2 Mesh Adaptation

In this section, selected algorithms presented in Section 3.2 are modified for templated meshes. These modifications are discussed and investigated. Algorithms presented in this section take a templated mesh Γ as the main input to generate an adapted templated mesh Ω where AT(Γ) and AT(Ω) geometry-conform to the same geometry. The input and the output of a mesh adaptation algorithm are even geometry-conforming to the same templated geometry in ideal situations.

Operations on elements in the structure instance of a templated mesh might affect other parts of the structure instance. Even if an algorithm performs an operation which would be local in AT(Γ), the effect is potentially non-local as due to the instance-template-relation (cf. Figure 5.6). For almost every operation performed on a templated mesh, all consequences for AT(Γ) have to be considered. In some cases, an operation which is beneficial in one area of the mesh potentially has negative impacts in other areas. For example, a refinement operation of one template element automatically refines all instances of that element potentially yielding non-desired element sizes in some ares of the structure instance. However, if such an operation is necessary or highly beneficial, like mesh refinement in an adaptive process, template splitting and cloning (cf. Section 5.2.1) is an option.



(a) Initial templated mesh

(b) Moved vertex

Figure 5.7: Conformity issues with operations performed on template boundaries

Moving the vertex (encircled in red) breaks the conformity of the structure instance.

For operations on or near template boundaries, which are part of any instance interface, Lemma 4.2 has to be fulfilled. This results in less freedom for operations, like vertex movement-based quality improvement algorithms, and is especially problematic for templated meshes with irregular instance graphs. However, valid operations in the *interior* of a template have no negative effects on the conformity of the structure instance. An example of algorithmic issues due to Lemma 4.2 is shown in Figure 5.7.

In the following, a selection of mesh adaptation algorithms for templated meshes are presented and discussed in detail.

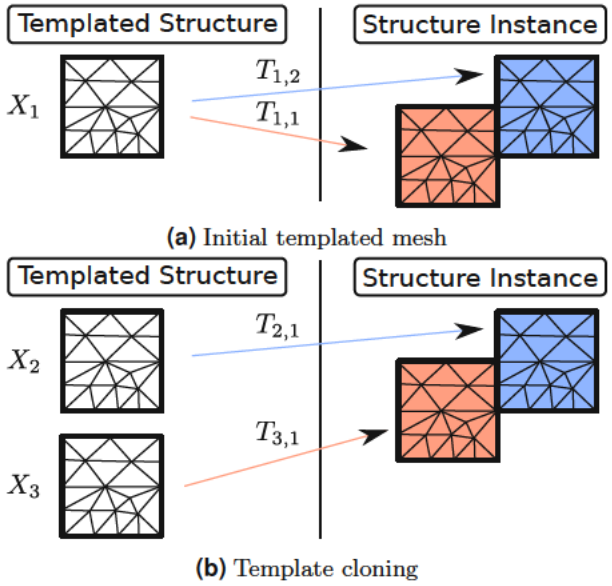
5.2.1 Template Cloning, Merging, and Splitting

Template cloning, merging, and splitting are three related algorithms for template management.

Template cloning (cf. Figure 5.8) is a technique where a mesh template Γ_i is cloned into two identical mesh templates. Each transformation function of the original mesh template is assigned to exactly one cloned mesh template. Cloning a template increases the memory consumption of the templated structure, because the mesh of the template has to be duplicated. If SVB (cf. Section 4.5) is used, the corresponding book-keeping has to be copied as well. No additional memory is required for the transformation functions, because each transformation occurs exactly once in the input and the output structure.

Template merging is the inverse operation to template cloning. If two different mesh templates Γ_i and Γ_j in a templated mesh are identical, these mesh templates can be merged into one single mesh template. The transformation functions of the resulting mesh template are the transformation functions of both source mesh templates Γ_i and Γ_j. As template merging is the inverse operation to template cloning, it reduces the memory consumption.

Template splitting (cf. Figure 5.9) splits a template Γ_i into two different *disjunct* templates using a mesh partition of Γ_i. To ensure that the resulting structure geometry-conforms to the same geometry, the transformation functions of Γ_i are copied to all new mesh templates. When storing the templated mesh without SVB, splitting a mesh template slightly increases the memory requirement because some vertices have to stored in both new templates and the transformation functions are copied. On the other hand, with SVB, there is an additional memory increase because the global vertices array and the local-to-global mapping array will grow due to additional shared vertices between the two new templates.

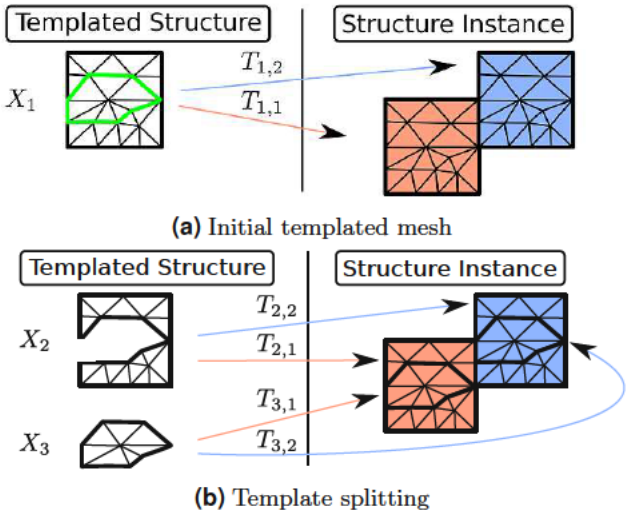


(a) Initial templated mesh

(b) Template cloning

Figure 5.8: The template cloning operation

The template X_1 (a) is cloned by copying the template mesh and assigning every instance to one of the new templates X_2 and X_3 (b).



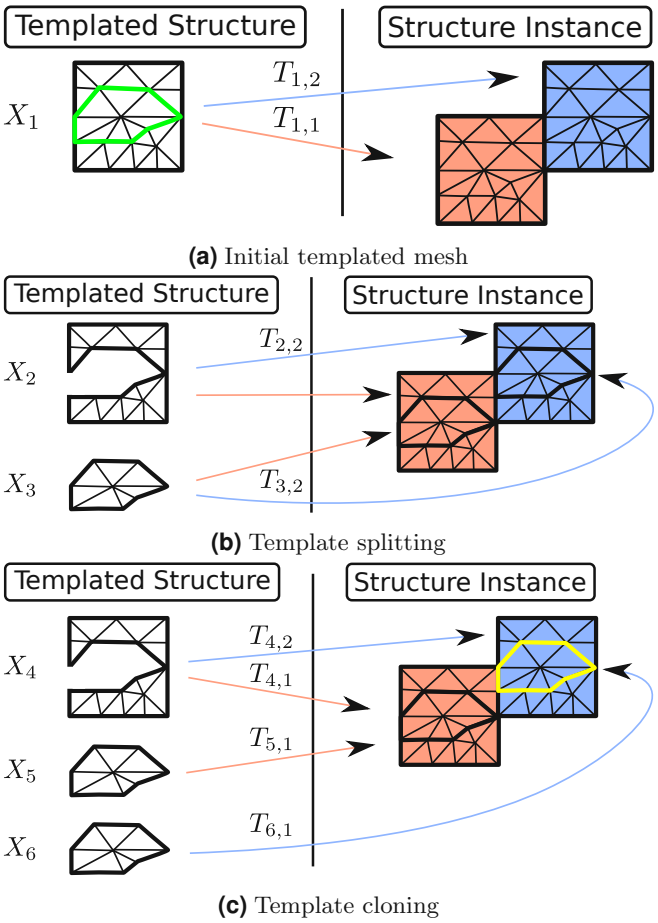
(a) Initial templated mesh

(b) Template splitting

Figure 5.9: The template splitting operation

The template X_1 is split into two parts based on the area highlighted in green (a). The transformation functions and region indicators for every instance are copied to both new mesh templates X_2 and X_3 (b). In particular, $T_{2,1} = T_{3,1} = T_{1,1}$ and $T_{2,2} = T_{3,2} = T_{1,2}$.

As mentioned above, mesh operations in $\mathrm{AT}(\Gamma)$ are not necessarily local and might affect other areas in $\mathrm{AT}(\Gamma)$. If an operation is mandatory or highly beneficial in a certain area A, but at the same time harmful to other areas, a combination of template splitting and cloning can be applied. At first, the mesh template is split into two templates, where the instance of one template covers A. Next, the template covering A is cloned into two templates, where one template only has the



(a) Initial templated mesh

(b) Template splitting

(c) Template cloning

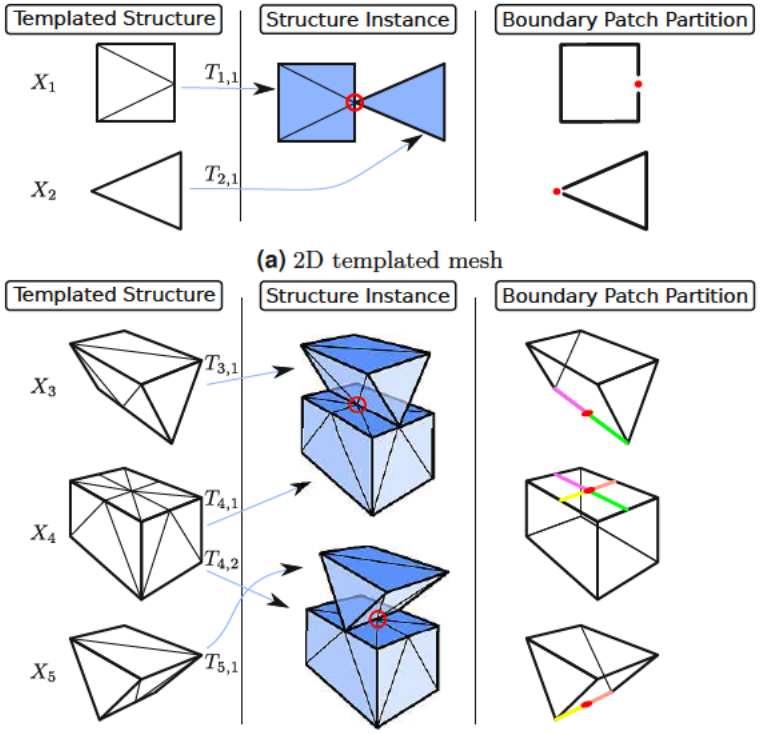
Figure 5.10: Combination of template splitting and cloning

First, a template splitting operation is performed to extract the area visualized in green (b). Then, the template X_3 in the middle templated mesh is cloned (c). For the resulting templated mesh on the bottom, any operation performed in the area highlighted in yellow is local in the structure instance (because it only affects template X_6).

transformation function leading to the instance which covers A. All other transformation functions are copied to the other new template. For the resulting templated mesh Ω, the operation is local in $\mathrm{AT}(\Omega)$ and no other ares of $\mathrm{AT}(\Omega)$ are affected. However, the resulting templated mesh Ω requires more memory, because one template is stored twice. This process is visualized in Figure 5.10.

5.2.2 Vertex Smoothing

As mentioned in Section 3.2.4, vertex smoothing is an important part in mesh quality improvement processes. Vertices, which are not part of the boundary of a template, can safely be moved as if they were in a non-templated structure. For all other vertices Lemma 4.2 has to hold to ensure the conformity of the output templated mesh. If a vertex is moved, all related templated vertices (using the boundary patch relation) have to be moved accordingly using a composition of template boundary mappings. If any movement of a related template vertex invalidates the corresponding mesh template, the movement is considered invalid.



(a) 2D templated mesh

(b) 3D templated mesh

Figure 5.11: Examples of vertices which cannot be moved

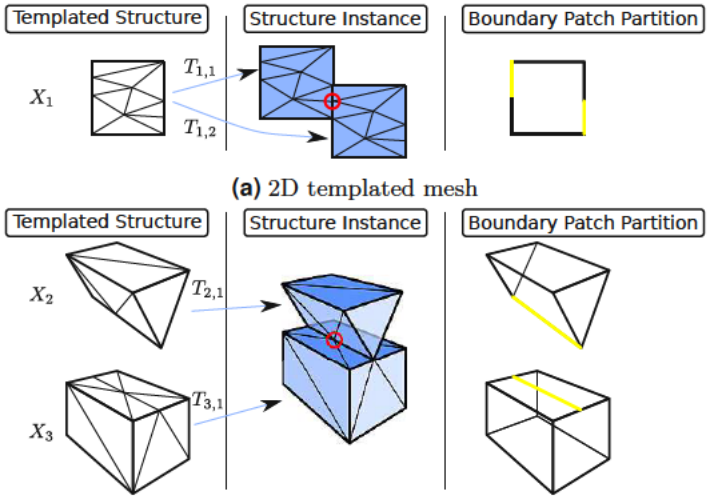
A simple 2D templated mesh where the vertex (encircled in red) cannot be moved because the intersection of all boundary patch elements is only one point (visualized in red) (a). A 3D templated geometry and its boundary patch partition is visualized in (b). The vertex highlighted in red could be moved in instance interface interf$(T_{3,1}, T_{4,1})$ and instance interface interf$(T_{4,2}, T_{5,1})$ when only considering one of these instance interfaces. However, this vertex cannot be moved when considering the complete templated mesh, because it is in the boundary patch element visualized in red.

The boundary patch partition of the geometry of the input templated mesh can be used to determine valid movements for a vertex in a template boundary. Every vertex in a mesh template $v \in \text{bnd}(\Gamma_i)$ can only be moved within a boundary patch in which the vertex is included. The valid positions for a vertex therefore are the intersection of all boundary patches in which the vertex is included. For example, if the intersection of the boundary patches in which the vertex is included is just one point, the vertex cannot be moved at all as shown in Figure 5.11. Examples for valid vertex movements are shown in Figure 5.12. Additionally, all vertices which are related in the boundary patch relation have to be moved accordingly using a corresponding composition of template boundary mappings.

$$w \sim_\Gamma v \Rightarrow w \mapsto \mathbb{T}_{T_{i_1,g_1},T_{i_2,g_2}} \circ \mathbb{T}_{T_{i_2,g_2},T_{i_3,g_3}} \circ \cdots \circ \mathbb{T}_{T_{i_{k-1},g_{k-1}},T_{i_k,g_k}}(F(v)) \tag{5.1}$$

F represents the movement of v. However, this is an issue for templated meshes with irregular instance graphs as depicted in Figure 4.11, where the movement of a vertex v induces a different movement for the same vertex, which is a contradiction and invalidates that particular movement.

To summarize, a vertex on the boundary of a template can only be moved if the instance graph is regular. Additionally, their movement is limited to the intersection of all boundary patch elements, in which the vertex is located.



(a) 2D templated mesh

(b) 3D templated mesh

Figure 5.12: Valid vertex movements visualized using the boundary patch partition

The vertices (highlighted in red) can safely be moved within the yellow boundary patch partition elements.

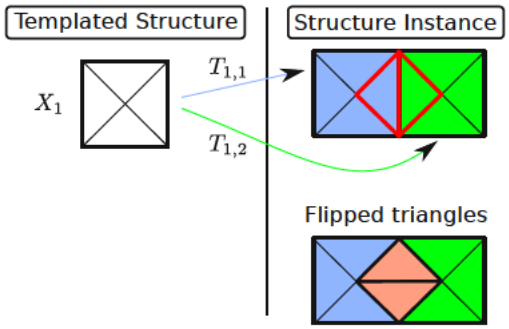


Figure 5.13: Issues with element flips across instances

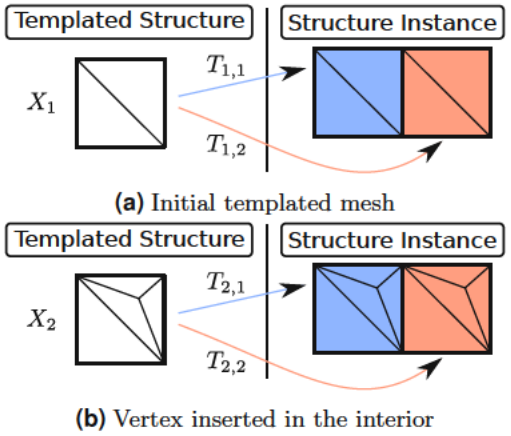
The triangles in the structure instance on the top highlighted in red cannot be flipped, because this would break the geometry-conformity of the template X_1.

5.2.3 Flips

Flipping operations (cf. Section 3.2.1) can safely be performed for all elements in a mesh template, because they do not alter the boundary. However, flipping elements in $\mathrm{AT}(\Gamma)$, which are in at least two different mesh instances, is generally not possible as visualized in Figure 5.13, because they break the conformity and/or the geometry-conformity of the templated structure.

5.2.4 Vertex Insertion and Refinement

When inserting a vertex into a mesh template, it has to be distinguished, if the vertex is to be inserted in the interior or on the boundary of the mesh template. Adding a vertex in the interior of the mesh template does not conflict with Lemma 4.2 and is a safe operation as visualized in Figure 5.14. On the other hand, inserting a vertex on the boundary potentially affects other mesh templates, if the newly inserted vertex is part of any geometry instance interface.



(a) Initial templated mesh

(b) Vertex inserted in the interior

Figure 5.14: Vertex insertion in the interior of a template

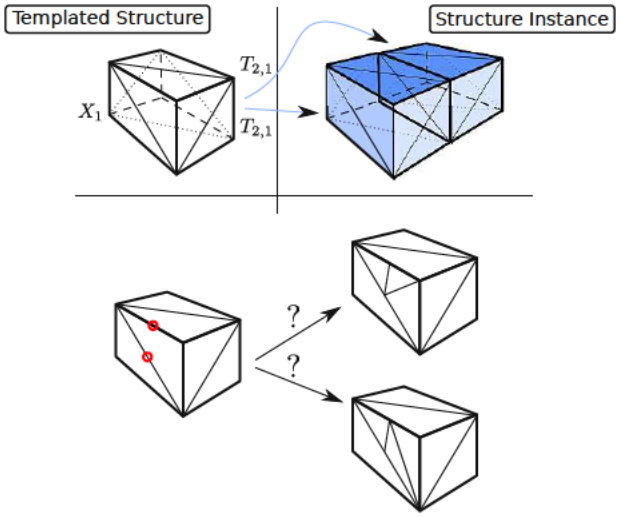


Figure 5.15: Conformity issues with insertion of multiple vertices

If two vertices (highlighted in red) are inserted in the template X_1, two different edge configurations are possible for the *front* face. The same edge configuration has to be applied to the *back* side to maintain the conformity of the structure instance.

A vertex inserted on a boundary which is part of any geometry instance interface induces the insertion of vertices in all direct and indirect neighboring mesh templates. This is always possible for 2D meshes, even for templated meshes with irregular instance graphs. However, for 3D meshes, not only the inserted vertex has an effect on the neighboring templates, but also all additionally inserted edges have to match at all instance interfaces. Especially, if multiple vertices are inserted at once, the edges have to be taken into account (cf. Figure 5.15). However, the edge configuration is unique when inserting a single vertex into the boundary of a template of a 3D templated simplex mesh.

As mention in Section 3.2.3, edge bi- or trisection is often used in element refinement algorithms, where one or two new vertices are inserted on every edge marked for refinement. The refinement algorithm remains unchanged for edges which are not part of the boundary. For all other edges, the refinement algorithm can be adapted using the vertex insertion described above.

5.2.5 Adapt to Delaunay Mesh

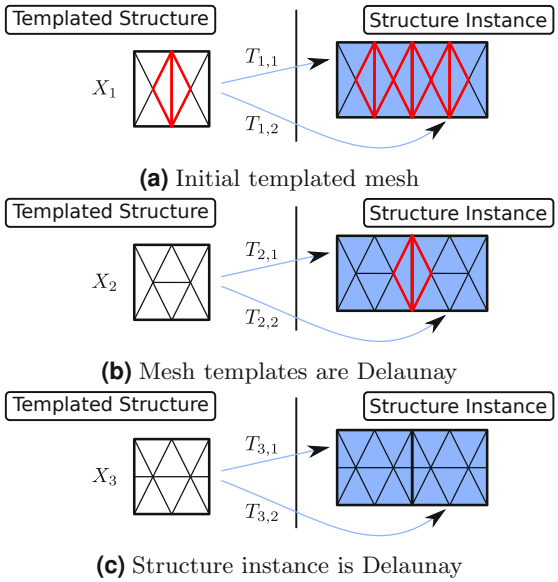
An algorithm is presented in this section, which adapts a templated mesh in a way that its structure instance is Delaunay. As shown in Section 4.4, it is not sufficient that all templates are Delaunay. Additionally, every facet in the boundary of a mesh template has to be template-aware locally Delaunay for $\mathrm{AT}(\Gamma)$ to be Delaunay (cf. Lemma 4.5).

```
1  Algorithm make_template_mesh_delaunay
   input  : templated simplex mesh Γ
   output : templated simplex mesh Γ which's structure instance is Delaunay
2  begin
3      foreach T_Γ ∈ templ(Γ) do
4          make T_Γ Delaunay
5      end
6      repeat
7          F ← ⋃_i {f ∈ facets_i(templ(Γ, i)) | f is not template-aware locally Delaunay}
8          foreach f ∈ F do
9              v ← centroid of f
10             Inserting the v in f using the Bowyer-Watson algorithm
11             foreach Template boundary face f̃: f̃ ∼_Γ f do
12                 Insert T_{f̃,f}(v) in f̃ using the Bowyer-Watson algorithm (cf. Section 3.2.3)
13                 if f̃ ∈ F then
14                     F ← F \ f̃
15                 end
16             end
17         end
18     until F ≠ ∅
19 end
```

Technique 5.8: Make templated mesh Delaunay

Technique 5.8 recursively inserts vertices in template boundary facets to make them template-aware locally Delaunay. At first, all mesh templates are made Delaunay using flip and Delaunay refinement operations (Line 4). Then, the set F of all template facets which are not template-aware locally Delaunay is determined (Line 7). For each template facet f in F, its centroid $\boldsymbol{v}$ is computed (Line 9). In Line 10 this centroid is inserted in the template facet f using the Bowyer-Watson algorithm (cf. Section 3.2.3). For every template boundary facet $\tilde{f}$, which is related to f, the vertex $\boldsymbol{v}$ is inserted using a template composition of boundary mappings $\mathbb{T}_{\tilde{f},f}$, which maps f to $\tilde{\boldsymbol{v}}$ (Lines 11-16). The Bowyer-Watson algorithm preserves the Delaunay property of the mesh templates and does not introduce new vertices (except the centroid which is handled separately) and therefore does not break the conformity of the structure instance. This process is repeated, until F is empty.

A visualization of Technique 5.8 is sketched in Figure 5.16. There is no formal guarantee that Technique 5.8 stops for arbitrary (valid) inputs. For geometries without acute angles, vertex insertion using the Bowyer-Watson algorithm does not influence other facets which are already template-aware locally Delaunay. However, for geometries with angles less or equal to 45°, a phenomenon called *ping-pong encroachment* potentially occurs as visualized in Figure 5.17. A technique called *protection balls* is used to avoid this issue for classical simplex meshes [90][119][128]. The same approach can also be applied to templated meshes, where the smallest protection ball of all instance vertices and edges is used to determine a *better* location for the vertex v (Line 9).



(a) Initial templated mesh

(b) Mesh templates are Delaunay

(c) Structure instance is Delaunay

Figure 5.16: A templated mesh is made template-aware Delaunay

A valid templated mesh with one mesh template is visualized (a). Neither the mesh template X_1 nor the structure instance is Delaunay (triangles which are not Delaunay are highlighted in red). At first, the mesh template is made Delaunay using algorithms presented in Section 3.2. The resulting mesh template is Delaunay but the structure instance is not (b). Inserting vertices at template faces which are not template-aware locally Delaunay yields the final templated mesh, the structure instance of which is Delaunay (c).

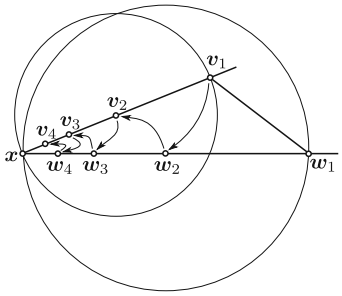


Figure 5.17: The *ping-pong encroachment* effect near small angles

Starting with vertices $\boldsymbol{x}$, $\boldsymbol{v}_1$ and $\boldsymbol{w}_1$, bisecting the facet edge simplex$(\boldsymbol{x}, \boldsymbol{w}_1)$ using its centroid $\boldsymbol{w}_2$ potentially results in an edge simplex$(\boldsymbol{x}, \boldsymbol{v}_1)$ not being locally Delaunay. When inserting vertex $\boldsymbol{v}_2$ into that edge, the edge simplex$(\boldsymbol{x}, \boldsymbol{w}_2)$ might not be locally Delaunay and so on. This iterative process potentially never stops.

5.2.6 Quality Improvement

Since most quality measures are invariant under rigid transformations, an operation which increases the quality of an element $E \in \mathrm{AT}(\Gamma)$ will have no negative effects on elements in other locations. A combined flipping and vertex smoothing approach is often sufficient to obtain a simplex mesh with *good* quality [32][75]. Both types of operations can safely be performed on the *inside* of mesh templates as mentioned in Section 5.2.2 and Section 5.2.3. However, for operations on the boundary of a mesh template, there are some restrictions. Flip operations cannot be performed, if at least two elements are in different mesh instances and for vertex smoothing, the search space for the local optimization of quality measures is restricted to the boundary patches. With these restrictions the mesh optimization algorithms can be modified for templated meshes.

Chapter 6

Decompositions and Symmetries

In this chapter, decomposition methods of geometries for templated structures are presented and discussed. A special focus is laid on geometries with symmetries. Taking care of issues and dependencies for templated structures in mesh generation and adaptation algorithms is complicated for objects with general similarities (cf. Chapter 5). However, for objects with symmetries, most of these issues and dependencies are much easier to handle.

Section 6.1 covers general identification methods for mesh and geometry decompositions. Special circumstances and algorithms for reflective symmetries and rotational symmetries are presented and discussed in Section 6.2 and Section 6.3, respectively. Finally, Section 6.4 covers combinations of symmetries.

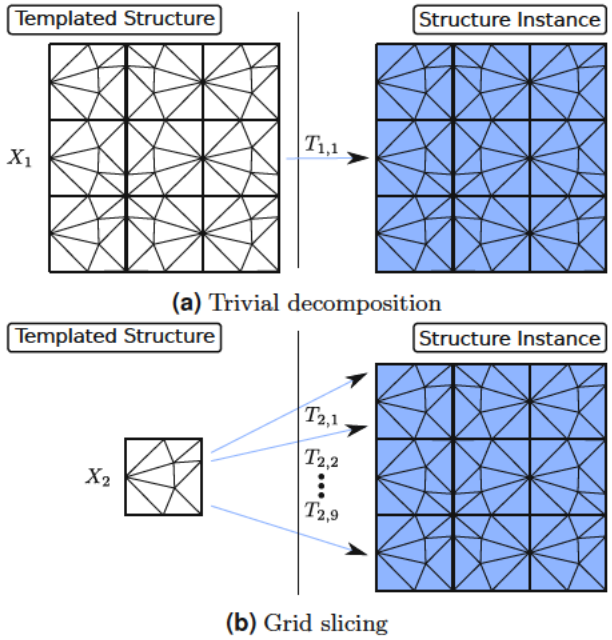
6.1 Decomposition Identification

Geometry decompositions and templated meshes and geometries have been introduced in Chapter 4. This section deals with the identification processes of geometry decomposition and how they are related to templated structures.

Let $(\mathcal{G}, \widetilde{\xi})$ be an n-dimensional multi-region geometry and $G = ((\mathcal{G}_1, \widetilde{\xi}|_{\mathcal{G}_1}), \ldots, (\mathcal{G}_k, \widetilde{\xi}|_{\mathcal{G}_k}))$ a decomposition of $(\mathcal{G}, \widetilde{\xi})$. The geometry decomposition G can be represented with a templated geometry, if the interior of every element of G has just one region. The interior is required, because points on block interfaces are allowed to be in multiple regions. If the requirement is fulfilled, a templated geometry Λ is said to be related to the decomposition G, if the structure instance of Λ is equal to $(\mathcal{G}, \widetilde{\xi})$ and for every decomposition element of G, there is an instance in Λ, which is equal to the decomposition element.

Any given geometry can be represented using a templated structure. For example, for every multi-region geometry, a trivial templated structure can be defined, which has one template for every region (being the region). Every template has only one instance indicated by the identity transformation function and the region indicator of the corresponding region. However, there is no benefit in using templated structures with this trivial approach. Except for very simple meshes, the memory consumption of a transformation function is far less than for a mesh. Therefore, it is advantageous to have a high number of instances per mesh template. Figure 6.1 visualizes the benefit of using multiple transformation functions with just one small template in contrast to the trivial decomposition.

On the other hand, according to Lemma 4.2, each instance interface potentially introduces new constraints and dependencies for the boundary mesh of the corresponding template, as shown in Figure 6.2. These constraints restrict several operations in algorithms presented in Section 5.2, which potentially lowers the effect of quality improvement algorithms.



(a) Trivial decomposition

(b) Grid slicing

Figure 6.1: Two different templated meshes with same structure instance

The trivial decomposition is used for (a). The templated mesh (b) is decomposed into nine mesh blocks all originating from the same mesh template. A total number of 72 vertices and 90 triangles must be stored for the mesh template X_1, while for mesh template X_2 only 10 vertices and 10 triangles have to be stored. Therefore, the templated mesh on the bottom requires less memory.

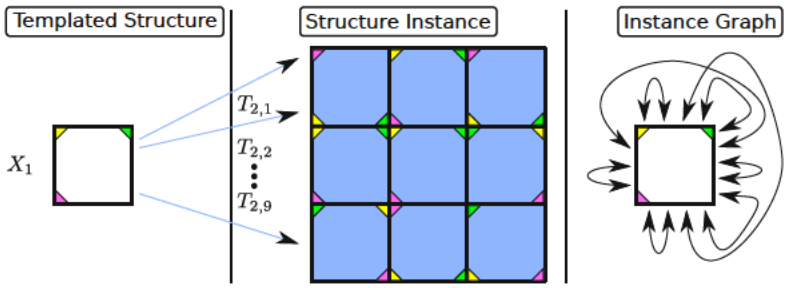


Figure 6.2: Boundary dependencies in a templated geometry

The green, yellow, and purple triangles are included in the drawing to indicate the orientation of the instances. Although there is only one template, all 12 instance interfaces of two pairwise different instances have different *configurations*, i.e., there are no two instances, the instance interface of which includes the same boundary parts. Therefore, there are many dependencies of the boundary elements of a corresponding mesh template for X_1 as indicated by the instance graph on the right.

It is therefore important to find a balance between memory savings and template dependencies by choosing a decomposition which has many similar blocks but as few dependencies between block templates as possible. In general, it is hard to find the *optimal* decomposition of a mesh or geometry, if there even is any. Some methods for finding a decomposition of meshes and/or geometries are presented in this section. All except graph-based decomposition methods work for geometries.

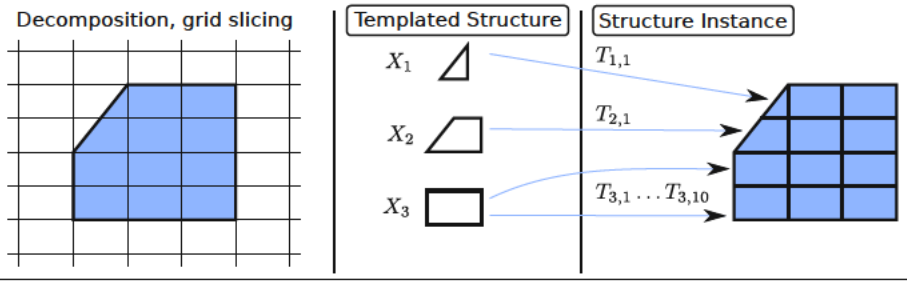


Figure 6.3: Grid slicing decomposition

A regular grid is laid over the geometry resulting in three different templates X_1, X_2, and X_3. The templates X_1 and X_2 only have one instance each, but the template X_3 has a total of 10 instances.

Trivial decomposition simply decomposes a geometry or mesh into only one decomposition element for each region, being the geometry or mesh region. The resulting templated structure has one template for every region, each with just one instance using the identity transformation function and the corresponding region indicator.

Grid slicing is a method, where a (regular) grid is laid over a geometry. The geometry is then sliced according to the overlay grid to obtain a decomposition of the geometry as visualized in Figure 6.3. Using a regular equidistant grid on a single-region geometry potentially results in many similar blocks which can originate from a small number of templates. However, due to the similarity to quad/octtree-based mesh generation algorithms, this method potentially creates blocks with bad shapes near the boundary of the geometry (cf. Figure 3.3).

Graph-based decompositions are methods based on graph partitioning algorithms [22]. Given a mesh or a multi-region mesh, the dual mesh or corresponding graph is generated by using each cell as a node. Two cells are connected, if they share a facet. Using a graph partitioning algorithm, the mesh can be decomposed according to the resulting graph partition. This method only works well for meshes but not for geometries. Additionally, it is very unlikely that the mesh decomposition created by this method results in mesh parts which are similar to each other and therefore can share a common template.

If available, **domain knowledge** can be used for decomposition methods. For example, if a copy operation of a part-geometry has been performed in the CAD process, this information could be re-used for the geometry decomposition. This approach only works in special cases where this domain knowledge is available and accessible. Also, the decomposition algorithm has to be formulated separately for each type of domain knowledge making these types of decomposition methods impractical.

In the field of automatic **quadrilateral and hexahedral** mesh generation, a lot of algorithms heavily rely on block decomposition methods [100][102][107], which can also be used for geometry decomposition. Mesh decomposition based on these methods is beneficial for quadrilateral or hexahedral mesh generation, but they do not offer guarantees for finding similar blocks. However, in CAE applications, decompositions with similar blocks are likely.

There are also algorithms which automatically detect **general similarities** in objects [61][99]. However, many of these algorithms are computationally intensive. Additionally, they work with points sampling and therefore are not stable. Methods and algorithms for automatic similarity detection are covered in Section 3.3.2.

A very promising decomposition method relies on **symmetries** of the geometry. A set $A \subseteq \mathbb{R}^n$ is said to have a symmetry, if there is a non-trivial rigid function T under which A is invariant, i.e., $A = T(A)$. There are two types of symmetries, which are of primary interest, being reflective symmetries and rotational symmetries (methods and algorithms for automatic detection of these types of symmetries are presented in Section 3.3.1). These two types of symmetries are covered in detail in Section 6.2 and Section 6.3. Combinations of these symmetries, like multiple point symmetries or a combination of reflective and rotational symmetry, are discussed in Section 6.4.

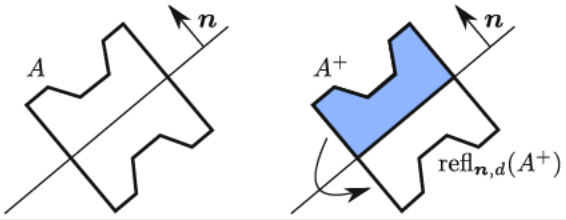


Figure 6.4: Reflective symmetry

The set A has a reflective symmetry indicated by the hyperplane with the normal vector $\boldsymbol{n}$ and the scalar d. It can be reconstructed by using the subset on the *positive* side A^+ and the reflection function $\text{refl}_{\boldsymbol{n},d}$.

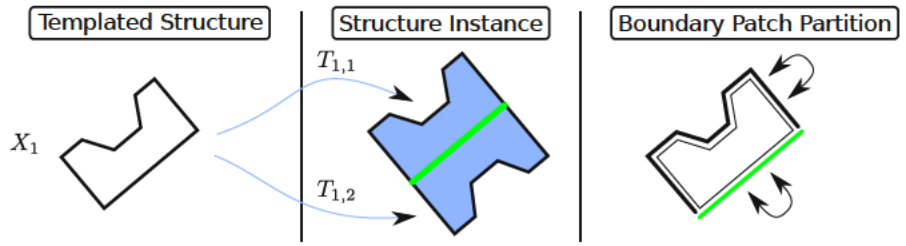


Figure 6.5: Boundary patch partition of a set with reflective symmetry

The arrows on the right visualize the boundary patch relation. Each boundary patch partition element is only related to itself. The green line indicates the reflecting hyperplane.

6.2 Reflective Symmetries

Reflective symmetries are (beside the identity) the simplest types of symmetries.

Definition 6.1 (Reflective symmetry). Let $A \subseteq \mathbb{R}^k$ be a set. A is said to have a reflective symmetry, if there is a normalized vector $\boldsymbol{n} \in \mathbb{R}^k$ and a scalar $d \in \mathbb{R}$, for which $A = \text{refl}_{\boldsymbol{n},d}(A)$, with

$$\text{refl}_{\boldsymbol{n},d}(\boldsymbol{x}) := \boldsymbol{x} - 2(\boldsymbol{x} \cdot \boldsymbol{n} - d)\boldsymbol{n} \tag{6.1}$$

The hyperplane $\mathbb{H}_{\boldsymbol{n},d}$ is called the reflecting hyperplane and $\text{refl}_{\boldsymbol{n},d}$ is called the reflection function.

In other words, a set has a reflective symmetry, if there exists an affine coordinate system with an orthogonal matrix, where one coordinate is negated. The reflecting hyperplane always passes through the center of gravity of the set. For a reflective symmetric set A with the reflecting hyperplane $\mathbb{H}_{n,d}$, the set A can be reconstructed by using the subset on the *positive* side of the hyperplane $A^+ := \{\boldsymbol{x} \in A | \boldsymbol{n} \cdot \boldsymbol{x} \geq d\}$:

$$A = A^+ \cup \text{refl}_{\boldsymbol{n},d}(A^+) \tag{6.2}$$

In other words, the subset on one side of the reflecting hyperplane can be re-constructed by the subset on the other side using the reflection function $\text{refl}_{\boldsymbol{n},d}$ as visualized in Figure 6.4. Additionally, every point on the reflecting plane is invariant, i.e., is a fixed point, under the reflection function: $\text{refl}_{\boldsymbol{n},d}|_{A^0} = \mathbb{I}$ with $A^0 := \{\boldsymbol{x} \in A | \boldsymbol{n} \cdot \boldsymbol{x} = d\}$.

A geometry $\mathcal{G}$ having a reflective symmetry can be represented by a templated geometry with just one template being $\mathcal{G}^+$ (or $\mathcal{G}^-$) and two instances with the transformation functions $\mathbb{I}$ and $\text{refl}_{\boldsymbol{n},d}$. The same approach can also be applied to meshes having reflective symmetries. This templated structure has only one instance interface which is included in the reflection hyperplane.

The boundary patch partition of the sole template is fairly simple and consists of only two elements, being the boundary which is included in the reflecting hyperplane and the rest of the boundary. Both boundary patches are only related to itself in the boundary patch relation and the instance graph is regular. The boundary patch partition and boundary patch relation of a templated geometry with a reflective symmetry is visualized in Figure 6.5.

A multi-region geometry or a multi-region mesh is said to have a reflective symmetry with the reflecting hyperplane $\mathbb{H}_{\boldsymbol{n},d}$, if all regions have the same reflective symmetry. The corresponding multi-region templated structure has a template for each region and therefore additional instance interfaces are possible. However, no new transformation functions (besides $\mathbb{I}$ and $\mathrm{refl}_{\boldsymbol{n},d}$) are introduced for multi-region structures and the instance graph is still regular.

Using the property above, the following important Lemma for templated structures can be formulated:

Lemma 6.1 (Reflective symmetry conformity)**.** *Let $\boldsymbol{n} \in \mathbb{R}^n$ with $\|\boldsymbol{n}\|_2 = 1$, $d \in \mathbb{R}$, and $\mathcal{G}$ be a geometry with $\boldsymbol{n} \cdot \boldsymbol{x} \geq d$ for all $\boldsymbol{x} \in \mathcal{G}$. Then, for any region indicators $r_1, r_2 \in \{1, 2\}$, $\Lambda = ((\mathcal{G}, (\mathbb{I}, \mathrm{refl}_{\boldsymbol{n},d}), (r_1, r_2)))$ is a templated geometry.*

Similarly, for a mesh $\mathcal{M}$ with $\boldsymbol{n} \cdot \boldsymbol{x} \geq d$ for all $\boldsymbol{x} \in \mathrm{us}(\mathcal{M})$, the templated structure $\Gamma = ((\mathcal{M}, (\mathbb{I}, \mathrm{refl}_{\boldsymbol{n},d}), (r_1, r_2)))$ is a templated mesh.

Proof. Λ is a templated structure, because $\mathrm{int}^\star(\mathbb{I}(\mathcal{G})) \cap \mathrm{int}^\star(\mathrm{refl}_{\boldsymbol{n},d}(\mathcal{G})) = \emptyset$ and $\mathrm{AT}(\Lambda) = \mathbb{I}(\mathcal{G}) \cup \mathrm{refl}_{\boldsymbol{n},d}(\mathcal{G})$ is a geometry (by definition of a geometry). Therefore, Λ is a templated geometry.

Γ is a templated structure, because $\mathrm{int}^\star(\mathrm{us}(\mathbb{I}(\mathcal{G}))) \cap \mathrm{int}^\star(\mathrm{us}(\mathrm{refl}_{\boldsymbol{n},d}(\mathcal{G}))) = \emptyset$. If $\mathrm{AT}(\Gamma) = \mathbb{I}(\mathcal{M}) \cup \mathrm{refl}_{\boldsymbol{n},d}(\mathcal{M})$ is not conforming, then non-conformities can only occur on the instance interface which is included in the reflecting hyperplane. However, the reflecting hyperplane is invariant under the reflection function and therefore all elements are conforming. Thus, Γ is a templated mesh. □

Lemma 6.1 can also be formulated and proven for multi-region geometries and multi-region meshes. This implies that any mesh $\mathcal{M}^+$ which geometry-conforms to a geometry $\mathcal{G}^+$ can be used as a mesh template and it automatically results in a conforming templated mesh. This drastically simplifies the process of generating templated meshes based on geometries with reflective symmetry. Theoretically, any algorithm presented in Section 5.1 can be used for templated mesh generation. However, due to Lemma 6.1, Algorithm 6.1 presents a much simpler alternative: At first, the *positive* multi-region sub geometry $(\mathcal{G}, \widetilde{\xi})^+$ is extracted (Line 3). Then, a conventional mesh generation algorithm is used to create a multi-region mesh $(\mathcal{M}, \xi)^+$ which geometry-conforms $(\mathcal{G}, \widetilde{\xi})^+$ (Line 5). Finally, the resulting templated mesh is obtained by using every region of $(\mathcal{M}, \xi)^+$ as a mesh template with each two instances indicated by the transformation functions $\mathbb{I}$ and $\mathrm{refl}_{\boldsymbol{n},d}$ and the corresponding region indicator (Line 6).

1 **Algorithm** generate_templated_mesh_reflective_symmetry
  input : multi-region geometry $(\mathcal{G}, \widetilde{\xi})$ having a reflective symmetry with the reflecting hyperplane $\mathbb{H}_{n,d}$
  output : templated mesh Γ
2 **begin**
3 $(\mathcal{G}, \widetilde{\xi})^+ \leftarrow (\{\boldsymbol{x} \in \mathcal{G} | \boldsymbol{n} \cdot \boldsymbol{x} \geq d\}, \widetilde{\xi})$
4 $k \leftarrow \mathrm{rc}((\mathcal{G}, \widetilde{\xi})^+)$
5 $(\mathcal{M}, \xi)^+ \leftarrow$ generate_multi_region_mesh$((\mathcal{G}, \widetilde{\xi})^+)$
6 $\Gamma \leftarrow ((\mathrm{region}((\mathcal{M}, \xi)^+, 1), (\mathbb{I}, \mathrm{refl}_{\boldsymbol{n},d}), (1, 1)), \ldots, (\mathrm{region}((\mathcal{M}, \xi)^+, k), (\mathbb{I}, \mathrm{refl}_{\boldsymbol{n},d}), (k, k)))$
7 **end**

Algorithm 6.1: Templated mesh generation for geometries with reflecting symmetries

Even though reflective symmetries are easy to detect automatically (cf. Section 3.3.1) and the mesh generation process has no conformity and dependence issues, there is a theoretical upper bound on the improvements in memory and mesh generation runtime of a factor of two. Benchmark results for the improvements are given in Section 7.2. The approaches presented in this section can also by applied iteratively, if the geometry or mesh template itself again has a reflective symmetry.

6.3 Rotational Symmetries

The second main type of symmetries is the rotational symmetry.

Definition 6.2 (Rotational symmetry)**.** A set $A \subseteq \mathbb{R}^2$ is said to have a rotational symmetry, if there exists an angle $\alpha \in (0°, 180°]$ for which $A = \text{rot}_{\boldsymbol{c},\alpha}(A)$, with $\boldsymbol{c}$ being the center of gravity of A and

$$\text{rot}_{\boldsymbol{c},\alpha}(x) := \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} (\boldsymbol{x} - \boldsymbol{c}) + \boldsymbol{c} \tag{6.3}$$

$\boldsymbol{c}$ is also called the rotation center and α the rotation angle.

A set $B \subseteq \mathbb{R}^3$ is said to have a rotational symmetry, if there exists a normalized vector $\boldsymbol{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3$ and an angle $\alpha \in (0°, 180°]$ for which $B = \text{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha}(B)$, with $\boldsymbol{c}$ being the center of gravity of B and

$$\text{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha}(\boldsymbol{x}) := \left(\begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \sin(\alpha) + \left(\mathbb{I} - \boldsymbol{v}\boldsymbol{v}^T\right) \cos(\alpha) + \boldsymbol{v}\boldsymbol{v}^T \right) (\boldsymbol{x} - \boldsymbol{c}) + \boldsymbol{c} \tag{6.4}$$

Again, $\boldsymbol{c}$ is also called the rotation center, $\boldsymbol{v}$ is called the rotation axis, and α is called the rotation angle.

A general rotation in $\mathbb{R}^2$ $\text{rot}_{\boldsymbol{c},\alpha}$ is given by a rotation center $\boldsymbol{c} \in \mathbb{R}^2$ and a rotation angle $\alpha \neq 0 \mod 360°$. In $\mathbb{R}^3$, an additional axis $\boldsymbol{v} \in \mathbb{R}^3$ is required to specify a rotation. Similar to reflections, a rotation has also fixed points being the rotation center $\boldsymbol{c}$ for rotations in $\mathbb{R}^2$ and every point on the rotation axis for rotations in $\mathbb{R}^3$. The general rotation in $\mathbb{R}^3$ can also be written in the following way [104]:

$$\text{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha} = R \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} R^{-1}(\boldsymbol{x} - \boldsymbol{c}) + \boldsymbol{c} \tag{6.5}$$

with

$$R = \mathbb{I} + G\sin(\theta) + G^2(1 - \cos(\theta)) \tag{6.6}$$

$$\boldsymbol{t} = \boldsymbol{v} \times (0, 0, 1)^T \tag{6.7}$$

$$\theta = \cos^{-1}\left(\boldsymbol{v} \cdot (0, 0, 1)^T\right) \tag{6.8}$$

$$G = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} \tag{6.9}$$

In this formulation of $\text{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha}$, R specifies a coordinate system change, where the rotation is performed around the z-axis. A set is said to have a rotational symmetry of order n, if $n = 360°/\alpha$. Every set with a rotational symmetry of order n has also a rotational symmetry of order k, if k is a factor of n. Usually, a set with a rotational symmetry is specified using its largest symmetry order.

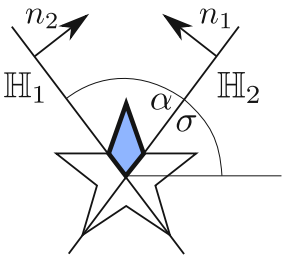


Figure 6.6: A slice of a set with a rotational symmetry

The slice starting angle is depicted as σ. The slice with a spanning angle of α is embedded between two hyperplanes with the normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$.

In contrast to reflective symmetries, there are infinitely many possible constructing subsets of a set having a rotational symmetry. Let A be a set with a rotational symmetry of order n. Then any *slice* subset $S \subseteq A$ with an angle of $360°/n$ can be used to reconstruct A. A slice is defined as follows:

Definition 6.3 (Slice)**.** Let $A \subseteq \mathbb{R}^2$, $\boldsymbol{c} \in \mathbb{R}^2$, $\alpha \in [0°, 360°]$, and $\sigma \in [0°, 360°)$. The slice with center $\boldsymbol{c}$, starting angle σ, and spanning angle α of A is defined as:

$$\mathcal{S}_{\boldsymbol{c},\sigma,\alpha}(A) := \{\boldsymbol{x} \in A | (\boldsymbol{x} - \boldsymbol{c}) \cdot \boldsymbol{n}_1 \geq 0 \wedge (\boldsymbol{x} - \boldsymbol{c}) \cdot \boldsymbol{n}_2 \geq 0\} \tag{6.10}$$

with

$$\boldsymbol{n}_1 = \begin{pmatrix} -\sin(\sigma) \\ \cos(\sigma) \end{pmatrix} \tag{6.11}$$

$$\boldsymbol{n}_2 = \begin{pmatrix} \sin(\sigma + \alpha) \\ -\cos(\sigma + \alpha) \end{pmatrix} \tag{6.12}$$

For $B \subseteq \mathbb{R}^3$ and $\boldsymbol{c}, \boldsymbol{v} \in \mathbb{R}^3$, the slice of B is given by:

$$\mathcal{S}_{\boldsymbol{c},\boldsymbol{v},\sigma,\alpha}(B) := \{\boldsymbol{x} \in B | (\boldsymbol{x} - \boldsymbol{c}) \cdot \boldsymbol{n}_1 \geq 0 \wedge (\boldsymbol{x} - \boldsymbol{c}) \cdot \boldsymbol{n}_2 \geq 0\} \tag{6.13}$$

with

$$\boldsymbol{n}_1 = R \begin{pmatrix} -\sin(\sigma) \\ \cos(\sigma) \\ 0 \end{pmatrix} \tag{6.14}$$

$$\boldsymbol{n}_2 = R \begin{pmatrix} \sin(\sigma + \alpha) \\ -\cos(\sigma + \alpha) \\ 0 \end{pmatrix} \tag{6.15}$$

R is the coordinate system matrix defined in Equation 6.6.

In other words, a slice is defined as all points which are on the positive side of two hyperplanes specified by the normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. α is the spanning angle of the slice, i.e., the angle between the two hyperplanes, and σ is the starting angle. A slice of an object in $\mathbb{R}^2$ is visualized in Figure 6.6.

Let $A \subseteq \mathbb{R}^2$ be a set with a rotational symmetry of order n and center $\boldsymbol{c}$. A can be reconstructed using a slice with any starting angle σ in the following way:

$$A = \bigcup_{k=0}^{n-1} \text{rot}_{\boldsymbol{c},k\frac{360°}{n}} \left(\mathcal{S}_{\boldsymbol{c},\sigma,\alpha}(A)\right) \tag{6.16}$$

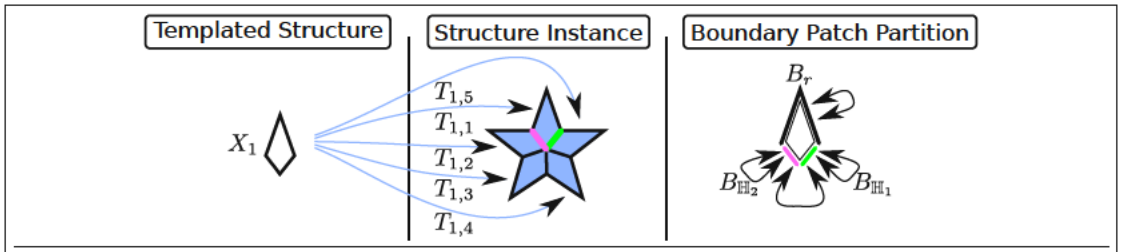


Figure 6.7: Boundary patch partition of a set with rotational symmetry

The boundary partition consists of three elements, being $B_{\mathbb{H}_1}$, $B_{\mathbb{H}_2}$, and B_r where $B_{\mathbb{H}_1}$ and $B_{\mathbb{H}_2}$ are the only two different elements which are related to each other by the boundary patch relation.

The same also works for $B \subseteq \mathbb{R}^3$ with a rotational symmetry of order n, center $\boldsymbol{c}$, and rotational axis $\boldsymbol{v}$:

$$B = \bigcup_{k=0}^{n-1} \text{rot}_{\boldsymbol{c},\boldsymbol{v},k\frac{360°}{n}} \left(\mathcal{S}_{\boldsymbol{c},\boldsymbol{v},\sigma,\alpha}(B)\right) \tag{6.17}$$

Theoretically, the choice of the starting angle σ is arbitrary. However, certain starting angles potentially have a bad impact on the element quality, like the smallest angle, of the resulting templated mesh. Therefore, the geometry has to be analyzed to determine good choices for σ. An algorithm which tackles this issue for linear geometries is presented in Section 6.3.1.

Using the relation above, a geometry $\mathcal{G}$ with a rotational symmetry of order n, rotation center $\boldsymbol{c}$, and rotation axis $\boldsymbol{v}$ can be represented by a templated geometry with just one template being a slice $\mathcal{S}_{\boldsymbol{c},\boldsymbol{v},\sigma,\frac{360°}{n}}(\mathcal{G})$ and n instances using the transformation functions $\text{rot}_{\boldsymbol{c},\boldsymbol{v},k\frac{360°}{n}}$, $k = 0, \ldots, n-1$. The boundary patch partition of the sole template, as visualized in Figure 6.7, is straightforward and consists of three boundary patches: The boundary of the template, which is included in the first hyperplane of the slice $B_{\mathbb{H}_1}$, the boundary of the template, which is included in the second hyperplane of the slice $B_{\mathbb{H}_2}$, and the remaining B_r. The boundary patch partition elements $B_{\mathbb{H}_1}$, $B_{\mathbb{H}_2}$, and B_r are related to each other by the boundary patch relation as follows:

$$B_{\mathbb{H}_1} \sim B_{\mathbb{H}_1} \tag{6.18}$$
$$B_{\mathbb{H}_2} \sim B_{\mathbb{H}_2} \tag{6.19}$$
$$B_{\mathbb{H}_1} \sim B_{\mathbb{H}_2} \tag{6.20}$$
$$B_r \sim B_r \tag{6.21}$$

Therefore, $B_{\mathbb{H}_1}$ and $B_{\mathbb{H}_2}$ must have the same mesh to ensure conformity. Note, that $B_{\mathbb{H}_1}$ and/or $B_{\mathbb{H}_2}$ can theoretically be empty for geometries which are not connected. The instance graph is, similar to reflective symmetries, regular.

Like with reflective symmetries, a multi-region geometry or multi-region mesh is said to have a rotational symmetry, if every region has the same rotational symmetry. For a multi-region geometry (or multi-region mesh) with a rotational symmetry of order n, a region might also have a rotational symmetry with the same rotation center and rotation axis but rotational symmetry order of k, where n is a factor of k. As for reflective meshes, the templated structure of a multi-region geometry or mesh has one template for each region. However, the boundary patch partition for each template (which represents a region) might be more complex than described above (cf. Figure 6.8). The instance graph, however, is still regular, because no new transformation functions are introduced.

Any mesh generation algorithm presented in Section 5.1 can be used to create a templated mesh based on a multi-region geometry with a rotational symmetry. Due to the simple structure of a rotationally symmetric object, the calculation of the boundary patch partition can be simplified as described in Algorithm 6.2.

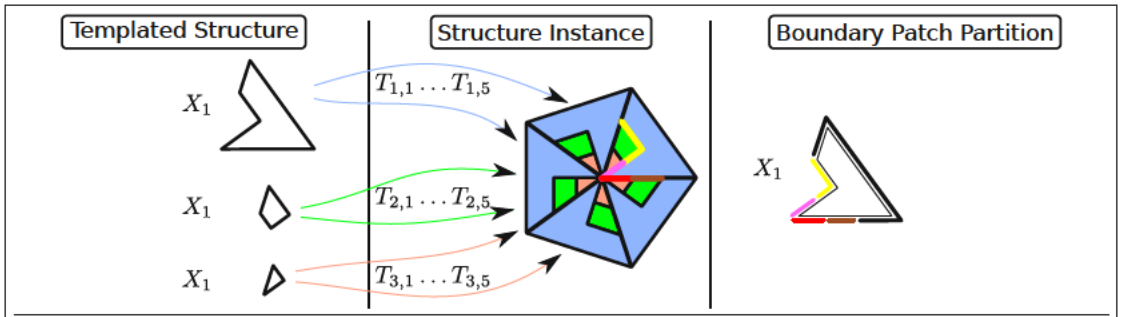


Figure 6.8: Boundary patch partition of a multi-region geometry with rotational symmetry

On the right the boundary patch partition of the template X_1 is visualized. The boundary patch partition elements colored in red and brown are not obvious when viewing only a single slice. These boundary patch partition elements are indicated by the neighboring instances in other slices.

At first, the slice with a starting angle of σ is calculated (Line 5). One half of the *first* hyperplane of this slice (with starting angle σ) is explicitly obtained (Line 6-7). Using this half hyperplane, a manifold partition for all regions is obtained, where the region is intersected with the half hyperplane (Line 10). These manifold partitions are refined (Line 11) and rotated (Line 13) to the other hyperplane of the slice to obtain a manifold partition of the slice hyperplanes B_S. Then, for each region, the section which lies within the slice is obtained (Line 15). The boundary patch partition of the region slice is calculated by using all elements of B_S, which are included in the region (Line 16) and the complement of the region slice boundary (Line 17). Although Algorithm 6.2 is formulated for 3D multi-region geometries, it can also be applied to 2D geometries. In that case, the rotation axis v is not needed and the hyperplane and slice creation as well as the rotation functions are less complicated to calculate.

In contrast to reflective symmetries, rotational symmetries promise higher improvements in memory and mesh generation runtime. Assuming no overhead, a theoretical improvement of a factor of n could be achieved for geometries with rotational symmetry of order n (in contrast to an improvement of two for reflective symmetries).

However, one issue with rotational symmetries is the appearance of small angles for objects with a high rotational symmetry order. In particular, a slice of a set A with a rotational symmetry of order n and the rotation center (as well as a neighborhood) included in A has an internal angle equal to $\alpha = 360°/n$ at the rotation center. For large n, this angle can be very small and therefore an issue for mesh generation processes, especially for algorithms with guaranteed mesh element qualities, like mesh generation algorithms based on Delaunay refinement (cf. Section 3.1.1). A possible solution for issues with small angles is given in Section 6.3.2.

6.3.1 Finding Optimal Slicing Positions

The choice of the starting angle σ of a slice has a potential impact on the mesh element quality as some slice positions might introduce small inner angles as visualized in Figure 6.9. It is therefore of interest to find an *optimal* starting angle for the slice. Because of the periodicity with respect to the angles, it is sufficient to only check and analyze starting angles σ in the range of $[0, \alpha)$.

For linear geometries (using boundary representation), Algorithm 6.3 evaluates all possible slice positions and returns a slice position which maximizes the smallest, newly introduced inner angle. For each facet which is intersected by a slice hyperplane, a new angle θ between that facet and the hyperplane is introduced. This angle θ is (piecewise) linearly dependent on the starting angle σ as shown in Figure 6.10. The algorithm partitions the slice starting angle interval $[0, \alpha)$ into sections where for every facet the angle between a potential slice hyperplane and facet is linear.

Algorithm generate_bndpart_rotational_symmetry

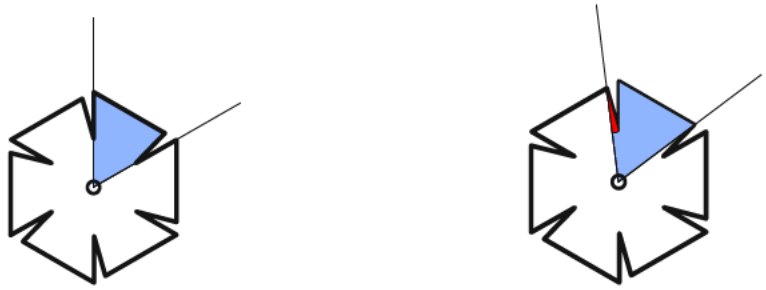
input : multi-region geometry $(\mathcal{G}, \widetilde{\xi})$ with a rotational symmetry of order n, rotation center $\boldsymbol{c}$, and rotation axis $\boldsymbol{v}$
Starting angle for slice σ

output : templated geometry of slice Λ and boundary patch partitions B_i for each geometry template

begin
- $r \leftarrow \mathrm{rc}((\mathcal{G}, \widetilde{\xi}))$
- $\alpha \leftarrow \frac{360^\circ}{n}$
- $S \leftarrow \mathcal{S}_{\boldsymbol{c},\boldsymbol{v},\sigma,\alpha}(\mathcal{G})$
- $\mathbb{H} \leftarrow$ starting hyperplane of slice S with starting angle of σ
- $\mathbb{H} \leftarrow \mathbb{H} \cap S$
- $B_{\mathbb{H}} \leftarrow \{\mathbb{H}\}$
- **for** $i \leftarrow 1$ **to** r **do**
  - $P_i \leftarrow \mathrm{ip}(\mathbb{H}, \mathrm{region}((\mathcal{G}, \widetilde{\xi}), i)) \cup \{\mathrm{cl}(\mathbb{H} \setminus \mathrm{region}((\mathcal{G}, \widetilde{\xi}), i))\}$
  - $B_{\mathbb{H}} \leftarrow \mathrm{refine}(B_{\mathbb{H}}, P_i)$
- **end**
- $B_S \leftarrow B_{\mathbb{H}} \cup \mathrm{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha}(B_{\mathbb{H}})$
- **for** $i \leftarrow 1$ **to** r **do**
  - $R_i \leftarrow \mathrm{region}((\mathcal{G}, \widetilde{\xi}), 1) \cap S$
  - $B_i \leftarrow \{b \in B_S | b \subseteq R_i\}$
  - $B_i \leftarrow B_i \cup \{\mathrm{bnd}R_i \setminus \mathrm{cl}(\bigcup_{b \in B_i} b)\}$
- **end**

end

Algorithm 6.2: Boundary patch partition generation of geometries with rotational symmetries



(a) *Good* position **(b)** *Bad* position

Figure 6.9: Examples for possible slice positions

While the slice position shown in (b) introduces a small inner angle (visualized in red) which potentially affects the mesh element quality in a negative way, the slice position shown in (a) is *optimal*, meaning that the newly introduced angles are minimized.

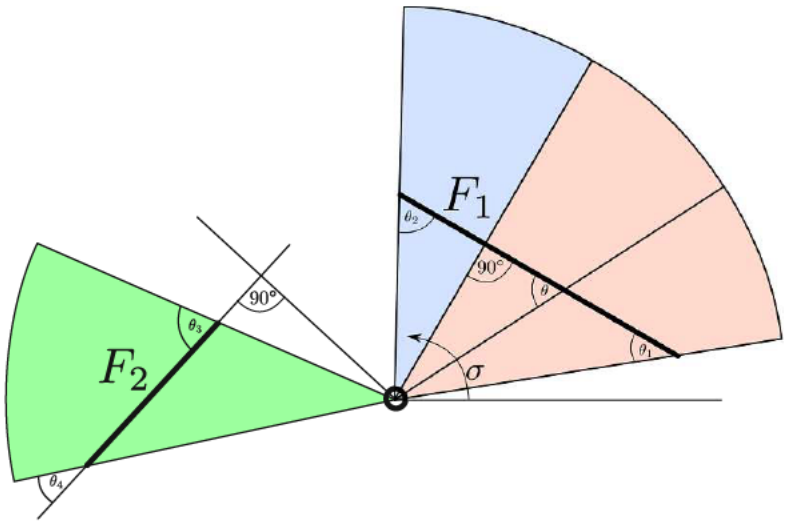


Figure 6.10: Piecewise linear slice starting angle

The newly introduced angle θ between F_1 and F_2 and a potential slice hyperplane is piecewise linearly dependent on the slice hyperplane angle σ. In the red section, θ starts at the angle θ_1 and increases linearly to a maximum of 90° and in the blue section, the angle θ starts at a maximum of 90° and decreases linearly to θ_2. Because the closest point of the affine hull of facet F_2 is not included in F_2, there is only one section (colored in green), where θ starts at a maximum angle of θ_3 and decreases linearly to θ_4.

Then, for each of these sections a linear optimization is used to maximize the smallest newly introduced angle. The section with the largest newly introduced angle is chosen as the *optimal* slice position.

Algorithm 6.3 is formulated for 3D linear geometries but also works for 2D linear geometries with some minor modifications. For simplicity, the input geometry is assumed to have its center located at the origin and the z-axis is the rotation axis. At first, all facets which are (partially) included in a slice with a starting angle of zero and spanning angle of α are obtained (Line 5). The section, for which the newly introduced angles for every facet are linear, is stored in P which is initially set to the angle interval $[0, \alpha)$ (Line 6). For each vertex of a facet, the angle of that vertex (in relation to the center and the x-axis) is calculated and sorted in ascending order (Lines 8-12).

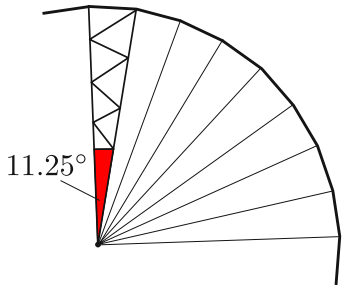


Figure 6.11: Issues with small angles

For sets with a high rotational symmetry order, small angles occur near the rotation center. The smallest angle of the triangle colored in red is 22.5°.

If the first and the last angle are equal, than the facet is radial and no additional angles are introduced between that facet and any potential slice hyperplane (Line 13). The angle of the first vertex $\theta[1]$ and the angle of the *last* vertex $\theta[k]$ are new separation points for the sections in P and therefore added to P (Line 16). In Lines 17 to 21, the angle (to the x-axis) of the closest point of the facet f (to the origin) is also added to P, if that angle is in the interval$[\theta[1], \theta[k]]$ (cf. Figure 6.10 for an explanation why this is required). The section separation points in P are then sorted in ascending order (Line 24) and for every two consecutive elements of P, an interval $[P[i], P[i+1]]$ is formed. The angle of a potential slice hyperplane and every facet is linear in that interval. Therefore, a linear optimization is used to find the slice hyperplane angle $\tilde{\sigma}$ which maximizes the smallest newly introduced angle (Line 29). If this maximal smallest angle $\tilde{\sigma}$ is larger than any previous best choice σ, $\tilde{\sigma}$ is used as a new best choice (lines 30-34).

6.3.2 Handling Small Angles

Extracting slices of rotationally symmetric objects with a high symmetry order n might lead to small angles near the rotation center as visualized in Figure 6.11. In that case, a regular n-polygon $P \subseteq A$ with its center being the rotation center can be extruded from A to form a separate block. To minimize newly introduced angles, P should be rotationally aligned in a way that the slice starting angle σ of A passes through a vertex of P. The remaining set $A \setminus P$ is still rotationally symmetric with order n, but the small angle in the center is eliminated. Using this approach, the geometry can be represented by a templated geometry having two templates, being the polygon P around the rotation center and a slice of $A \setminus P$.

This process is visualized in Figure 6.12. The structure instance of a templated mesh, which is generated based on that templated geometry, is potentially not rotationally symmetric around its center any more. However, no angle of $360°/n$ is introduced at the center of the slice but two angles equal to $(180° + \alpha)/2 = (n+2)/(2n)180°$ are introduced (cf. Figure 6.13). These angle are larger than the slice angle of $360°/n$ which would have been introduced in the center. This potentially leads to a better quality of the mesh elements in the slice instances. The mesh generation of the center template is not required to be rotationally symmetric and the mesh generation algorithm therefore has more freedom when creating volumetric elements hence leading to better overall mesh element quality. The approach also works for 3D sets and geometries. However, the set P, which is cut out around the axis, is not a regular n-prism in general.

Algorithm optimal_slicing_angle

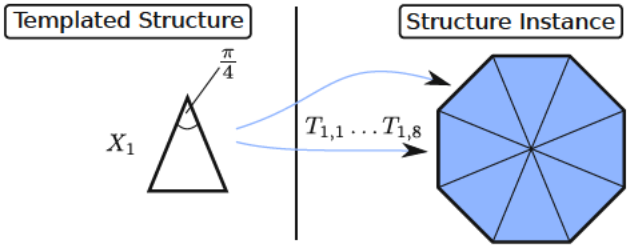
input : multi-region geometry $(\mathcal{G}, \widetilde{\xi}) \subseteq \mathbb{R}^3$ having a rotational symmetry of order k with rotation center $\boldsymbol{c} = \mathbf{0}$ and rotation axis $\boldsymbol{v} = (0,0,1)^T$

output : σ

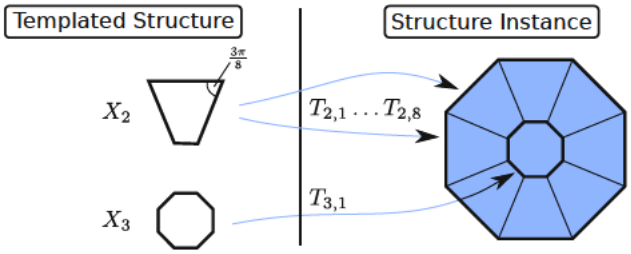
begin
- $\tilde{\boldsymbol{x}} = (1,0,0)^T$
- $\alpha \leftarrow \frac{360^\circ}{k}$
- $F \leftarrow \{f \text{ is a facet of } (\mathcal{G}, \widetilde{\xi}) | f \cap \text{int}(\mathcal{S}_{\boldsymbol{c},\boldsymbol{v},0,\alpha}(\mathbb{R}^n)) \neq \emptyset\}$
- $P \leftarrow \{0, \alpha\}$
- **foreach** $f \in F$ **do**
  - $\boldsymbol{n} \leftarrow$ normalized normal vector of f
  - $V \leftarrow \text{elem}_0(f)$
  - $k \leftarrow |V|$
  - $\theta \leftarrow \left(\arccos\left(\frac{V[1]\cdot\tilde{\boldsymbol{x}}}{\|V[1]\|_2}\right), \ldots, \arccos\left(\frac{V[k]\cdot\tilde{\boldsymbol{x}}}{\|V[k]\|_2}\right)\right)$
  - Sort θ ascending
  - **if** $\theta[1] = \theta[k]$ **then**
    - continue
  - **end**
  - $P \leftarrow P \cup \{\theta[1], \theta[k]\}$
  - $\tilde{\boldsymbol{v}} \leftarrow \boldsymbol{n}(\boldsymbol{n} \cdot V[1])$
  - $\tilde{\theta} \leftarrow \arccos\left(\frac{\tilde{\boldsymbol{v}}\cdot\tilde{\boldsymbol{x}}}{\|\tilde{\boldsymbol{v}}\|_2}\right)$
  - **if** $\tilde{\theta} \in [\theta[1], \theta[k]]$ **then**
    - $P \leftarrow P \cup \{\tilde{\theta}\}$
  - **end**
- **end**
- $\delta_{\max} \leftarrow 0$
- Sort P ascending
- **for** $i \leftarrow 1$ **to** $|P| - 1$ **do**
  - $\sigma_1 \leftarrow P[i]$
  - $\sigma_2 \leftarrow P[i+1]$
  - $F_i \leftarrow \{f \in F | f \cap \mathcal{S}_{\boldsymbol{c},\boldsymbol{v},\sigma_1,\sigma_2}(\mathbb{R}^n) \neq \emptyset\}$
  - Use linear optimization to find $\tilde{\sigma} \in [\sigma_1, \sigma_2]$ for which the smallest newly introduced angle with all facets in F_i is maximal
  - $\delta \leftarrow$ smallest angle which is newly introduced with hyperplane at angle $\tilde{\sigma}$ and every facet
  - **if** $\delta > \delta_{\max}$ **then**
    - $\delta_{\max} = \delta$
    - $\sigma \leftarrow \tilde{\sigma}$
  - **end**
- **end**

end

Algorithm 6.3: Finding *optimal* slicing angle



(a) One template with a normal slice.



(b) Two templates being the center and a slice without the center.

Figure 6.12: Small angle optimization for rotational symmetries

A regular eight-polygon is decomposed in two different ways. The smallest inner angle of template X_1 (a) is 45°, while the smallest inner angle of template X_2 and X_3 is 67.5° which is significantly larger and therefore enables better element qualities (b).

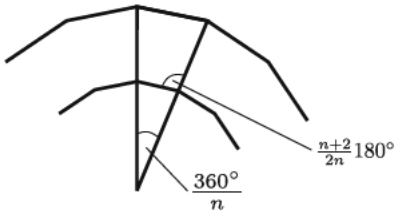


Figure 6.13: Small angle optimization for high rotational symmetry orders n.

6.4 Combined Symmetries

Objects with a combination of symmetries are discussed in this section. Two particular types of combinations are of interest: Combinations of multiple reflective symmetries (Section 6.4.1) and combinations of a reflective symmetry and a rotational symmetry (Section 6.4.2).

6.4.1 Combination of Reflective Symmetries

An object with multiple reflective symmetries, where the reflecting hyperplanes are pairwise orthogonal to each other, can benefit from each reflective symmetry by recursively applying the approaches proposed in Section 6.2 as visualized in Figure 6.14. This is ensured by Lemma 6.2.

Lemma 6.2 (Combination of reflective symmetries). *Let $A \subseteq \mathbb{R}^n$ be a set with two reflective symmetries, each represented by its reflecting hyperplane $\mathbb{H}_{\boldsymbol{n}_1,d_1}, \mathbb{H}_{\boldsymbol{n}_2,d_2}$. If the hyperplanes are orthogonal to each other, i.e., their normal vectors are orthogonal to each other, then $A^{+,\boldsymbol{n}_1,d_1}$ has a reflective symmetry with hyperplane $\mathbb{H}_{\boldsymbol{n}_2,d_2}$ with*

$$A^{+,\boldsymbol{n}_1,d_1} := \{x \in A | n_1 \cdot x \geq d_1\} \tag{6.22}$$

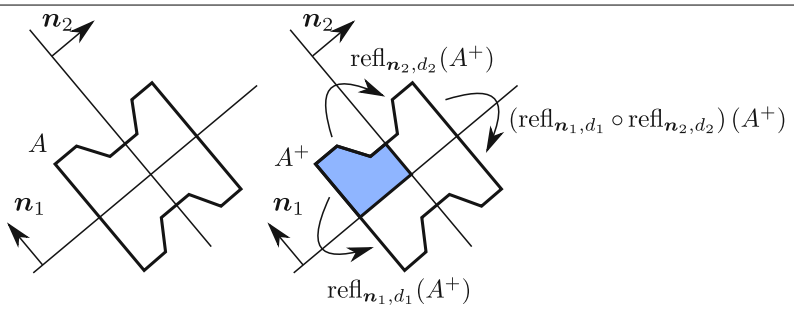


Figure 6.14: Combination of reflective symmetries

The set A has two reflective symmetries indicated by the hyperplanes with the (orthogonal) normal vectors $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. The reconstruction process uses the subset which is on the positive side of both hyperplanes A^+ and the reflection functions $\text{refl}_{\boldsymbol{n}_1,d_1}$ and $\text{refl}_{\boldsymbol{n}_2,d_2}$. The missing *diagonal* set is obtained by applying the combination of these reflection functions $\text{refl}_{\boldsymbol{n}_1,d_1} \circ \text{refl}_{\boldsymbol{n}_2,d_2}$ on A^+.

Proof. For $x \in A^{+,\boldsymbol{n}_1,d_1}$, the following holds:

$$\boldsymbol{n}_1 \cdot \text{refl}_{\boldsymbol{n}_2,d_2}(\boldsymbol{x}) = \boldsymbol{n}_1 \cdot (\boldsymbol{x} - 2(\boldsymbol{x} \cdot \boldsymbol{n}_2 - d_2)\boldsymbol{n}_2) = \boldsymbol{n}_1 \cdot \boldsymbol{x} - 2(\boldsymbol{x} \cdot \boldsymbol{n}_2 - d_2) \underbrace{\boldsymbol{n}_1 \cdot \boldsymbol{n}_2}_{=0} = \boldsymbol{n}_1 \cdot \boldsymbol{x} \geq d_1 \quad (6.23)$$

Therefore, for every $\boldsymbol{x} \in \text{refl}_{\boldsymbol{n}_2,d_2}(A^{+,\boldsymbol{n}_1,d_1})$, the inner product of $\boldsymbol{x}$ with $\boldsymbol{n}_1$ is larger or equal to d_1 and $\text{refl}_{\boldsymbol{n}_2,d_2}(A^{+,\boldsymbol{n}_1,d_1}) = A^{+,\boldsymbol{n}_1,d_1}$. □

This lemma also works for sets which have multiple reflective symmetries. Note, that for $A \subseteq \mathbb{R}^n$, A can have a maximum number of n reflective symmetries which are pairwise orthogonal to each other. Let $A \subseteq \mathbb{R}^n$ be a set with two reflective orthogonal symmetries with their reflecting hyperplanes $\mathbb{H}_{\boldsymbol{n}_1,d_1}, \mathbb{H}_{\boldsymbol{n}_2,d_2}$. The resulting templated geometry is described by using one geometry template being

$$A^+ := \{\boldsymbol{x} \in A | \boldsymbol{x} \cdot \boldsymbol{n}_1 \geq d_1 \wedge \boldsymbol{x} \cdot \boldsymbol{n}_2 \geq d_2\} \quad (6.24)$$

and all possible compositions of the reflection functions $\mathbb{I}, \text{refl}_{\boldsymbol{n}_1,d_1}, \text{refl}_{\boldsymbol{n}_2,d_2}, \text{refl}_{\boldsymbol{n}_1,d_1} \circ \text{refl}_{\boldsymbol{n}_2,d_2}$. The resulting templated geometry therefore is

$$\Lambda = ((A^+, (\mathbb{I}, \text{refl}_{\boldsymbol{n}_1,d_1}, \text{refl}_{\boldsymbol{n}_2,d_2}, \text{refl}_{\boldsymbol{n}_1,d_1} \circ \text{refl}_{\boldsymbol{n}_2,d_2}), (1,1,1,1))). \quad (6.25)$$

The same approach can also be applied to objects which have more than two (pairwise orthogonal) reflective symmetries (cf. Figure 6.14).

The approaches and algorithms presented in Section 6.2 can be modified to work with multiple reflections as well. The property that for reflective symmetries the instance interfaces are always conforming, also holds for multiple reflective symmetries. Therefore, Algorithm 6.1 can easily be adapted to scenarios with multiple reflective symmetries. The modified algorithm is presented in Algorithm 6.4. Instead of using the *positive* side of just one hyperplane, the positive side of all hyperplanes is used as the geometry template (Line 3). Additionally, multiple transformation functions have to be created for every composition of reflections and identities (Lines 6-8). The algorithm will generate a total number of 2^k transformation functions. Therefore, a theoretical improvement in memory usage and algorithm runtime of a factor of 2^k can be achieved in memory and runtime optimization.

Algorithm generate_templated_mesh_multiple_reflective_symmetries

input : multi-region geometry $(\mathcal{G}, \widetilde{\xi})$ having k (pairwise orthogonal) reflective symmetries with the reflecting hyperplanes $\mathbb{H}_{\boldsymbol{n}_1,d_1}, \ldots, \mathbb{H}_{\boldsymbol{n}_k,d_k}$

output : templated mesh Γ

begin

- $(\mathcal{G}, \widetilde{\xi})^+ \leftarrow (\{\boldsymbol{x} \in \mathcal{G} | \boldsymbol{n}_1 \cdot \boldsymbol{x} \geq d_1 \wedge \cdots \wedge \boldsymbol{n}_k \cdot \boldsymbol{x} \geq d_k\}, \widetilde{\xi})$
- $r \leftarrow \mathrm{rc}((\mathcal{G}, \widetilde{\xi})^+)$
- $(\mathcal{M}, \xi)^+ \leftarrow$ generate_multi_region_mesh$((\mathcal{G}, \widetilde{\xi})^+)$
- $T \leftarrow (\mathbb{I}, \mathrm{refl}_{\boldsymbol{n}_1,d_1})$
- **for** $i \leftarrow 2$ **to** r **do**
  - $T \leftarrow (T_1, \ldots, T_{|T|}, T_1 \circ \mathrm{refl}_{\boldsymbol{n}_i,d_i}, \ldots, T_{|T|} \circ \mathrm{refl}_{\boldsymbol{n}_i,d_i})$
- **end**
- $\Gamma \leftarrow ((\mathrm{region}((\mathcal{M}, \xi)^+, 1), T, (1, \ldots, 1)), \ldots, (\mathrm{region}((\mathcal{M}, \xi)^+, r), T, (r, \ldots, r)))$

end

Algorithm 6.4: Templated mesh generation for geometries with multiple reflective symmetries

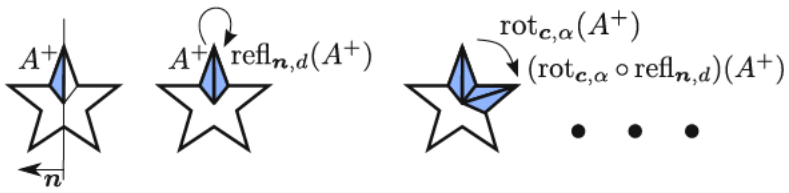


Figure 6.15: Combination of reflective and rotational symmetries

The set A has a rotational symmetry of order five and a reflective symmetry indicated by the hyperplane with the normal vector $\boldsymbol{n}$. The subset for the reconstruction process is obtained by using one half of a slice which also has a reflective symmetry. The resulting subset A^+ is then reflected around the reflecting hyperplane and in turn A^+ and $\mathrm{refl}_{\boldsymbol{n},d}(A^+)$ are rotated as described in Section 6.3.

6.4.2 Combination of Reflective Symmetry and Rotational Symmetry

The approaches introduced in Section 6.2 and Section 6.3 can be combined for sets which have a rotational symmetry and a reflective symmetry. This type of combination requires the axis of rotational symmetry to be included in the reflecting hyperplane for 3D sets. This requirement is not needed for 2D sets, because the reflective hyperplane always passes through the rotation center.

Let $A \subseteq \mathbb{R}^3$ be a set with a reflective symmetry depicted by $\mathbb{H}_{\boldsymbol{n},d}$ and a rotational symmetry of order k with rotation center $\boldsymbol{c}$ and rotation axis $\boldsymbol{v}$, where the rotation axis is a subset of the reflecting hyperplane. Then, A has a total of k reflective symmetries, each defined by rotating $\mathbb{H}_{\boldsymbol{n},d}$ around the axis of rotational symmetry by an angle of $i\frac{360°}{k}, i = 0, \ldots, k$. Additionally, there always exists a rotational symmetry slice which also has a reflective symmetry. Due to that reflective symmetry, this slice can be cut in half and used as a geometry template together with the transformation functions being the compositions of the reflection and all rotation transformation as visualized in Figure 6.15. The resulting templated structure is a templated geometry with the same property as the templated structure for reflective symmetries: The instance interfaces will always be conforming. Therefore, no boundary patch partition is needed during the mesh generation process and the templated mesh can easily be generated by creating any valid mesh.

The mesh generation process of a combined reflective-rotationally symmetric multi-region mesh is presented in Algorithm 6.5. Instead of using a normal slice as presented in Section 6.3, a half-slice is calculated using the reflecting hyperplane and the reflecting hyperplane rotated around the axis of rotational symmetry by an angle of $\alpha/2$ (Lines 4-5). A multi-region mesh is generated for this half-slice using an arbitrary volumetric mesh generation algorithm with multi-region support (Line 7). The transformation functions for the templated mesh are obtained by composing the identity and the symmetry reflection with all rotation angles around the axis of rotational symmetry (Line 8). The final templated mesh is generated by using the regions of the generated mesh $(\mathcal{M}, \xi)^+$ as mesh templates, each together with all transformation functions T (Line 9).

The theoretical improvement in memory and runtime is a factor of $2n$ for objects with a reflective symmetry as well as a rotational symmetry of order n. Because the boundary patch partition is not required, Algorithm 6.5 has much less overhead compared to the mesh generation process described in Section 6.3. Therefore, performance increases are expected and evaluated in Chapter 7.

1 **Algorithm** generate_templated_mesh_rotational_reflective_symmetries
  input : multi-region geometry $(\mathcal{G}, \widetilde{\xi})$ having a rotational symmetry of order k with rotation center $\boldsymbol{c}$ and rotation axis $\boldsymbol{v}$ and also having a reflective symmetry represented with the reflecting hyperplane $\mathbb{H}_{\boldsymbol{n},d}$. The rotation axis $\boldsymbol{v}$ has to be included in the reflecting hyperplane.
  output : templated mesh Γ
2 **begin**
3 $\quad \alpha \leftarrow \frac{360°}{k}$
4 $\quad \tilde{\boldsymbol{n}} = -\operatorname{rot}_{\boldsymbol{c},\boldsymbol{v},\frac{\alpha}{2}}(\boldsymbol{n})$
5 $\quad (\mathcal{G}, \widetilde{\xi})^+ \leftarrow (\{\boldsymbol{x} \in \mathcal{G} | (\boldsymbol{x} - \boldsymbol{c}) \cdot \boldsymbol{n} \geq 0 \wedge (\boldsymbol{x} - \boldsymbol{c}) \cdot \tilde{\boldsymbol{n}} \geq 0\}, \widetilde{\xi})$
6 $\quad r \leftarrow \operatorname{rc}((\mathcal{G}, \widetilde{\xi})^+)$
7 $\quad (\mathcal{M}, \xi)^+ \leftarrow$ generate_multi_region_mesh$((\mathcal{G}, \widetilde{\xi})^+)$
8 $\quad T \leftarrow (\mathbb{I}, \operatorname{refl}_{\boldsymbol{n},d}, \operatorname{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha}, \operatorname{rot}_{\boldsymbol{c},\boldsymbol{v},\alpha} \circ \operatorname{refl}_{\boldsymbol{n},d}, \ldots, \operatorname{rot}_{\boldsymbol{c},\boldsymbol{v},(k-1)\alpha}, \operatorname{rot}_{\boldsymbol{c},\boldsymbol{v},(k-1)\alpha} \circ \operatorname{refl}_{\boldsymbol{n},d})$
9 $\quad \Gamma \leftarrow ((\text{region}((\mathcal{M}, \xi)^+, 1), T, (1, \ldots, 1)), \ldots, (\text{region}((\mathcal{M}, \xi)^+, r), T, (r, \ldots, r)))$
10 **end**

Algorithm 6.5: Templated mesh generation for geometries with reflective and rotational symmetries

Chapter 7

Results and Applications

In this chapter, the techniques presented in the previous chapters are used for templated mesh generation benchmarks of synthetic as well as *real-world* objects are presented. The benchmark setup is presented in Section 7.1. Objects with reflective symmetries and rotational symmetries are covered in Section 7.2 and Section 7.3, respectively. Results for objects with general similarities are covered in Section 7.4. Section 7.5 presents an investigation of a FEM-based symmetry analysis of the five-pointed star, which will be referred to as *Gummel Star*. Memory and runtime benefits in FEM applications are discussed in Section 7.6.

7.1 Benchmark Setup

The benchmarks in this chapter show the benefits of the templated approaches presented in this work while at the same time preserving or even improving element qualities. To evaluate the memory savings, the required memory using the data structure `MESH` (cf. Section 2.4) – or for multi-region inputs `MRMESH` – are compared to the templated data structures `TEMPLATED_MESH` and `TEMPLATED_MESH_SVB` (cf. Section 4.5). For runtime analysis, the mesh generation time using a conventional approach (cf. Section 3.1) is compared to the runtime of a templated mesh generation process (cf. Section 5 and 6). To obtain better element qualities, mesh generation algorithms based on Algorithm 5.1.2 (and the specializations presented in Chapter 6) are utilized.

In typical mesh generation use cases an input geometry is meshed using a set of meshing parameters, e.g., local mesh element size or desired mesh element quality. To reflect these use cases in the benchmarks, a conventional mesh generation algorithm and a templated mesh generation algorithm are compared for generating meshes for the same geometry utilizing the same meshing parameters. However, the resulting mesh of the conventional mesh generation algorithm and the resulting structure instance (cf. Section 4.1) of the templated mesh differ in general. As presented later in this chapter, a high correlation between the memory savings and runtime speedups and cell counts is observed for most benchmarks. To associate the memory saving and the runtime speedup with one single cell count value, a common ground for the cell count of the conventionally generated mesh and the templated mesh is required. The average of the cell count of the conventionally generated mesh and the structure instance of the templated mesh is used in all cases for which two different meshes are compared. For example, the memory saving of two meshes with different cell counts will be associated with the average of the cell counts.

To ensure good mesh element quality, mesh element quality statistics of the resulting meshes and structure instances are generated. This statistic is used to evaluate, if the desired quality has been achieved. For 2D and 3D meshes, the smallest angle (for 3D meshes the smallest dihedral angle, cf. Definition 2.16) is used as mesh element quality measure. Benchmarks for 3D meshes are also performed using the radius-edge ratio quality measure parameter (cf. Definition 2.17).

In addition to mesh element quality measures, the benchmarks were carried out with different mesh element sizes (i.e. the element volume). As symmetry and similarity identification is not the focus of this work, it is assumed that the input geometry of each benchmark is already available in a templated form and symmetry or similarity detections are not required. Including automatic symmetry and similarity detection in the runtime benchmarks reduces the savings of the templated approaches. However, it has been shown, that the losses are moderate [48].

For the benchmarks in this chapter, a set of benchmark tools has been implemented utilizing the ViennaMesh software framework [19]. These implementations are published as free open source software [13]. Triangle and Tetgen are used for 2D and 3D mesh generation, respectively (cf. Section 3.1.3). All benchmarks have been carried out on an Intel Core i7-3770 workstation with 16GB of DDR3-1600 memory. The executables have been generated with the clang++ 3.1 compiler [3]. Runtime values are averaged over 100 benchmark executions.

7.2 Reflective Symmetries

The templated mesh generation of objects with reflective symmetries utilizes an implementation based on the algorithms presented in Section 6.2 and Section 6.4.1. In particular, the mesh generation software tools Triangle and Tetgen are used to generate a multi-region triangular and tetrahedral mesh from which the mesh templates are extracted; no special care has to be taken to ensure interface conformity of the structure instance (cf. Lemma 6.1 and Lemma 6.2).

Benchmarks of three example objects with reflective symmetries are presented:

(i) A 2D aircraft with one reflective symmetry (Figure 7.1a).

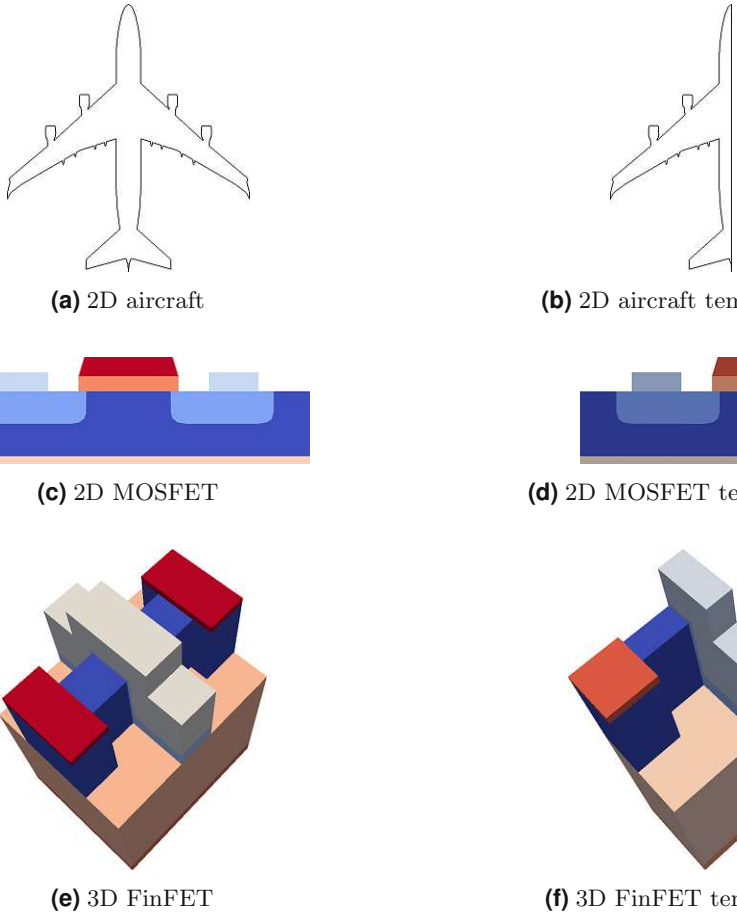
(ii) A 2D MOSFET [62] with one reflective symmetry (Figure 7.1c).

(iii) A 3D FinFET [43] with two reflective symmetries (Figure 7.1e).

The benchmark results for these three structures are shown in Figure 7.2. It can be seen that the runtime speedups for small cell counts are at most moderate. For the aircraft and the MOSFET, a cell count of at least 10^4 is required for the templated approach to be as fast as the conventional approach. It can also be seen that low cell counts result in a quite unstable runtime speedup behavior for the aircraft due to the fast underlying Triangle software for low cell counts. However, for high cell counts, the runtime speedups converge to the expected savings (cf. Section 6.2 and Section 6.4.1), which is a factor of two for the 2D aircraft and the 2D MOSFET and a factor of four for the 3D FinFET.

Regarding the memory savings, the behavior is similar. The memory savings are moderate for small cell counts. However, for large cell counts, a convergence to the expected memory savings is observed. Memory savings of nearly two (1.9 and higher) are achieved for aircraft meshes with cell counts larger than 10^4 using the templated data structure without SVB (cf. Section 4.5) and for cell counts larger than 10^5 using the templated data structure with SVB. Memory savings larger than the expected memory savings can be observed for the 2D MOSFET and the 3D FinFET. The reason for these high memory savings is the conventional multi-region mesh data structure `MRMESH`: The region identifier is stored for each cell while in the templated data structures, only one region identifier is required for a mesh template rather than for each cell. Therefore, for the 2D MOSFET, cell counts larger than 4×10^4 and 4×10^3 results in memory savings larger than two using the templated data structure with and without SVB, respectively. For the 3D FinFET, cell counts larger than 5×10^6 and 4×10^5 result in memory savings larger than four using the templated data structure with and without SVB, respectively.

Until now, memory savings are calculated using a conventionally generated mesh and a templated mesh, both generated with the same parameters. Memory savings of a templated mesh and its structure instance are visualized in Figure 7.3. It can be seen, that the qualitative behavior is the same as the memory savings of a templated mesh and a conventionally generated mesh with the same parameters (cf. Figure 7.2).



(a) 2D aircraft **(b)** 2D aircraft template

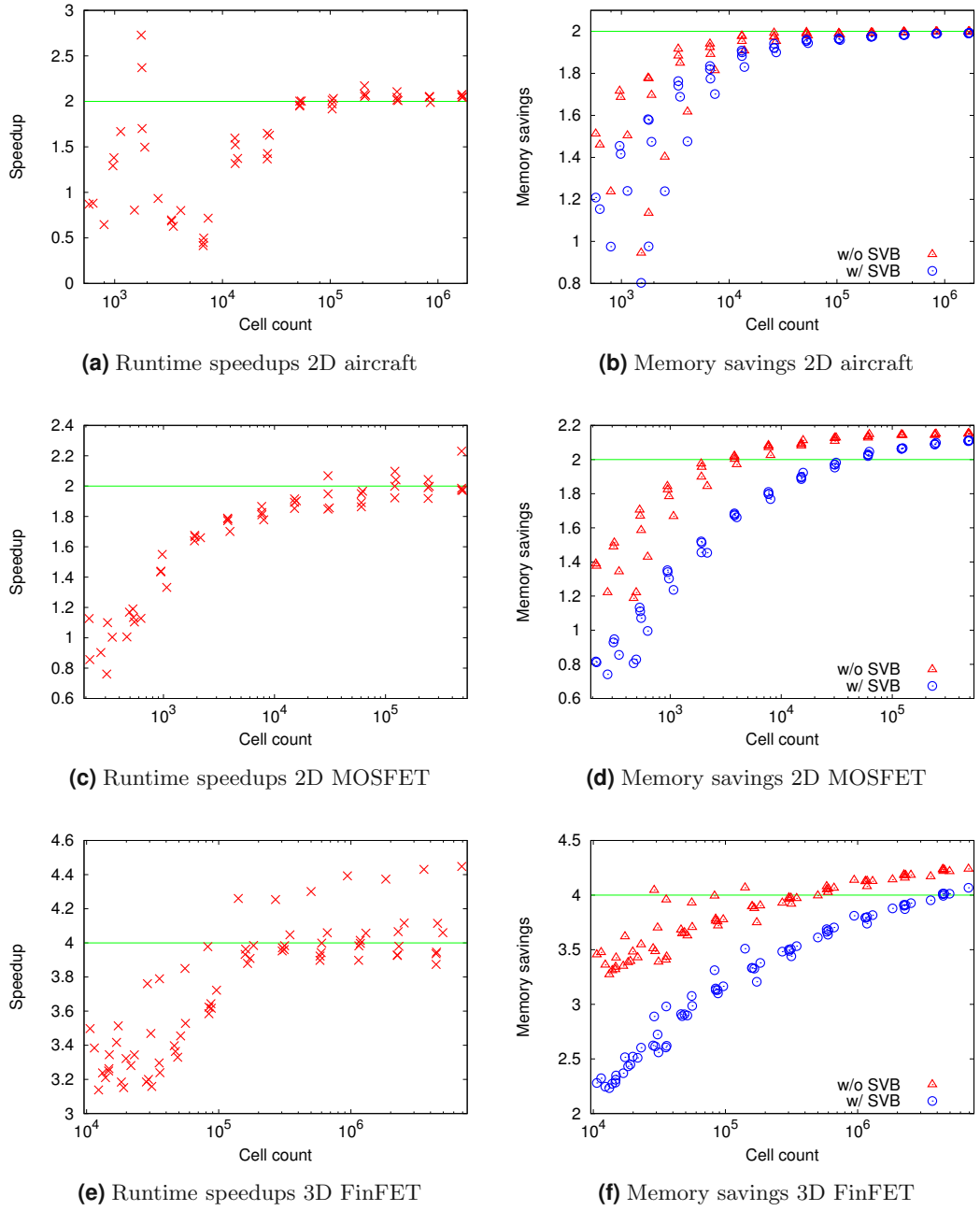
(c) 2D MOSFET **(d)** 2D MOSFET template

(e) 3D FinFET **(f)** 3D FinFET template

Figure 7.1: Benchmark objects with reflective symmetries and their templates

The aircraft (a) and the 2D MOSFET (c) have one reflective symmetry with a vertical reflecting hyperplane. The 3D FinFET (e) has two reflective symmetries with reflecting hyperplanes which are orthogonal to each other. While the aircraft geometry does not have any regions, the 2D MOSFET and the 3D FinFET geometry have multiple regions (indicated by different colors) which reflect different material properties.

As discussed in Section 5.1, a potential issue regarding templated mesh generation is the resulting mesh quality. To obtain *high quality* meshes, the corresponding mesh quality parameters of the mesh generation software are used to obtain a mesh with a specific minimal quality. For the 2D objects, being the aircraft and the MOSFET, smallest angles of $5°, 10°, 20°$, and $30°$ have been used as quality meshing parameters. All elements of the resulting conventionally generated meshes as well as the corresponding structure instances of all 2D MOSFET benchmarks have smallest angles larger or equal than the configured value and therefore meet the desired element quality constraints. However, the mesh generation algorithm was unable to meet the desired overall element quality for the 2D aircraft due to sharp angles in the input geometry leading to particularly bad elements (cg. Figure 7.4).



(a) Runtime speedups 2D aircraft

(b) Memory savings 2D aircraft

(c) Runtime speedups 2D MOSFET

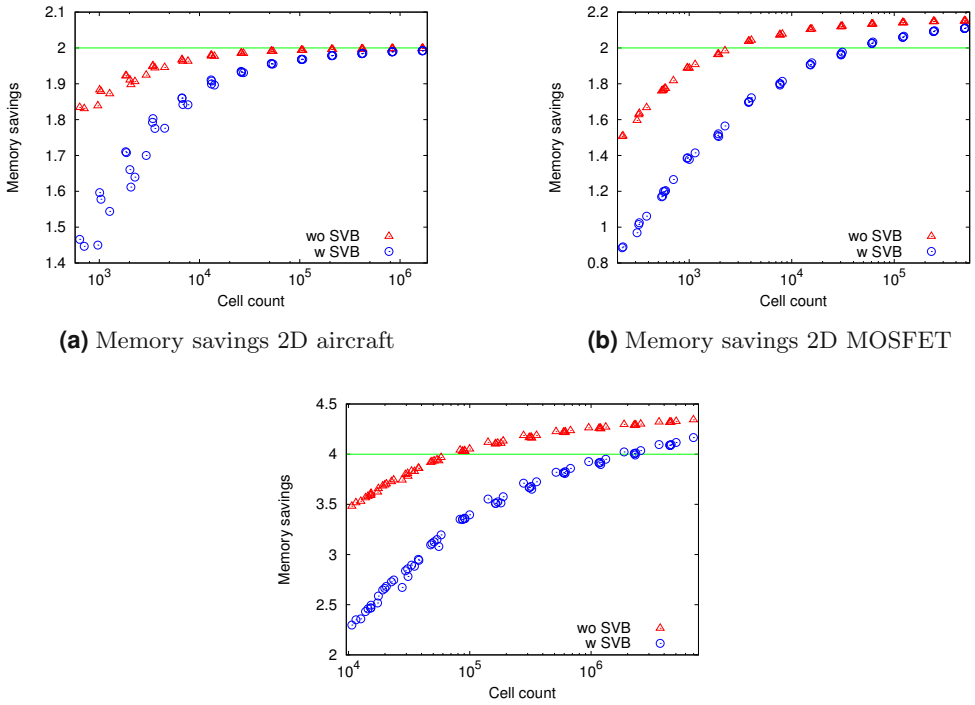
(d) Memory savings 2D MOSFET

(e) Runtime speedups 3D FinFET

(f) Memory savings 3D FinFET

Figure 7.2: Benchmark results for objects with reflective symmetries

The left and right column shows runtime and memory benchmarks, respectively, for different objects and for varying cell counts. Expected savings (visualized by the green lines) are a factor two for the 2D aircraft and 3D MOSFET (both having one reflective symmetry) and a factor of four for the 3D FinFET (having two reflective symmetries).



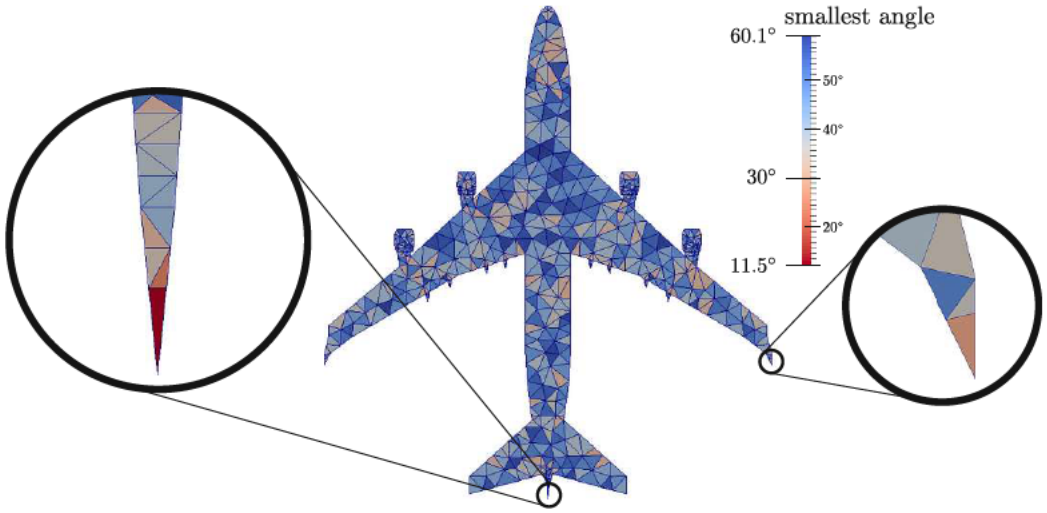
(a) Memory savings 2D aircraft

(b) Memory savings 2D MOSFET

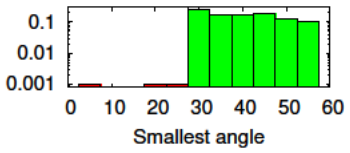
(c) Memory savings 3D FinFET

Figure 7.3: Memory benchmark results of structure instance

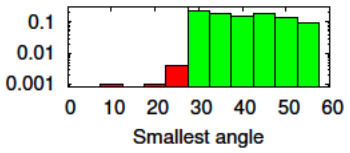
In contrast to Figure 7.2 memory savings of the templated mesh are calculated based on the memory requirement of its structure instance rather than a (non-symmetric) conventionally generated mesh of the same geometry. The expected savings are visualized by green lines.



(a) Element qualities



(b) Quality histogram, symmetric mesh



(c) Quality histogram, non-symmetric mesh

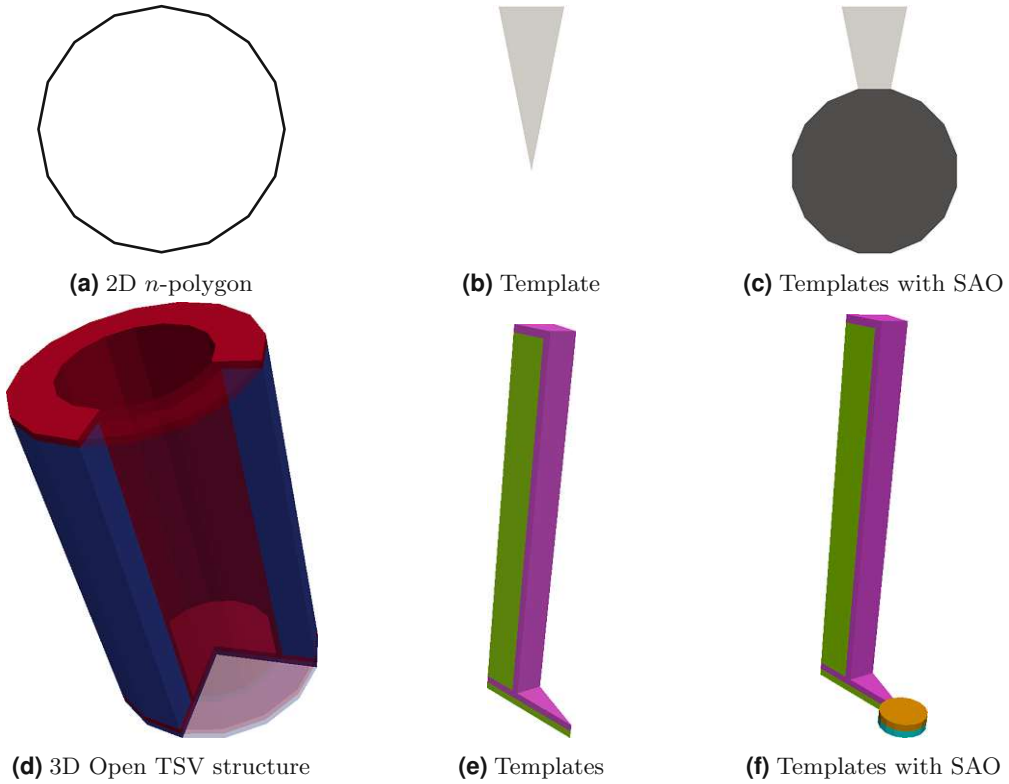
Figure 7.4: Element quality issues due to sharp angles in the 2D aircraft geometry

The mesh has been generated with a desired smallest angle of 30°. However at the wing and the end of the aircraft this element quality constraint cannot be achieved due to sharp angles in the input geometry (a). All other elements fulfill the desire element quality. The smallest angle quality histograms of the symmetric mesh (b) and non-symmetric mesh (c) show, that for the non-symmetric mesh slightly more elements have a quality worse than configured (visualized by the red bins). The histogram bins are given in log scale for better visualization of the *bad quality* bins.

Quality measure	Cell size	Configured quality	Relative count of elements with quality worse than configured (conventionally generated mesh)	Relative count of elements with quality worse than configured (structure instance)	Worst quality (conventionally generated mesh)	Worst quality (structure instance)	Relative worst value (structure instance)
A	0.5	6.0°	0.0	8.75×10^{-7}	6.0 °	5.48°	0.91
R	1024.0	1.5	2.82×10^{-4}	5.21×10^{-4}	1.53	1.57	1.05
R	16.0	1.5	3.04×10^{-5}	1.11×10^{-4}	1.56	1.73	1.16
R	8.0	1.5	1.97×10^{-5}	4.93×10^{-5}	1.59	1.58	1.05
A	2.0	12.0°	0.0	3.03×10^{-6}	12.0 °	11.58°	0.97
R	1.0	2.0	3.16×10^{-6}	5.27×10^{-6}	2.13	2.02	1.01
R	1.0	1.5	1.65×10^{-5}	1.90×10^{-5}	1.76	1.74	1.16
R	32.0	1.5	4.75×10^{-5}	1.31×10^{-4}	1.65	1.67	1.11
R	4.0	1.5	2.92×10^{-5}	4.59×10^{-5}	1.59	1.55	1.03
R	256.0	2.5	0.0	2.09×10^{-4}	2.5	2.51	1
R	1.0	2.5	1.35×10^{-6}	5.29×10^{-6}	2.87	3.68	1.47
R	64.0	1.5	1.22×10^{-4}	1.49×10^{-4}	1.73	1.54	1.03
R	128.0	1.5	1.21×10^{-4}	2.12×10^{-4}	1.56	1.68	1.12
R	4.0	2.0	3.49×10^{-6}	6.7×10^{-6}	2.08	2.03	1.01

Table 7.1: Element quality analysis of 3D FinFET benchmarks. The quality measure is given in the first column. A minimal dihedral angle quality measure is indicated by A and a radius-edge ratio quality measure is indicated by R. The relative worst value colored in red indicates elements with poor element quality in the structure instance. One single cell (located in the thin channel area) in the mesh template of that particular benchmark has a radius-edge ratio larger than 2.87 (the worst radius-edge ratio of the conventionally generated mesh).

Smallest dihedral angles of $6°, 12°$, and $18°$ and radius-edge ratios of $2.5, 2$, and 1.5 are used as quality meshing parameters for the 3D FinFET. However, the mesh generation algorithm is not able to fulfill these required mesh quality settings for all benchmarks. Out of 72 different configurations of mesh generation parameters, 23 conventionally generated meshes and 14 templated mesh structure instances did not fulfill the quality parameters. The quality benchmark results of all benchmarks, where the quality of the structure instance is worse than the quality of the conventionally generated mesh, are given in Table 7.1. It can be seen, that the worst element quality value for most benchmarks is at most 16% worse than the configured value. Only for one benchmark (indicated in red), the worst element quality value is off by 47%. However, in this extreme case, the number of elements with worse quality than configured is less than 5×10^{-6}; considering all benchmarks it is even less than 0.5% elements. Therefore, the mesh quality of the templated approach is considered as good as the quality of conventionally generated meshes for the most cases and minimally worse otherwise.



(a) 2D n-polygon **(b)** Template **(c)** Templates with SAO

(d) 3D Open TSV structure **(e)** Templates **(f)** Templates with SAO

Figure 7.5: Benchmark objects with rotational symmetries

Both objects have a rotational symmetry order of 16. On the left, the object geometry is visualized. In the middle, templates for the classical templated approach with one slice are shown. On the right, the small angle optimization (SAO) templates are given with one slice and the middle part which is a regular 16-polygon or regular 16-prism. The templates are visualized in different colors.

7.3 Rotational Symmetries

Algorithm 5.6 and Algorithm 5.7 are used to generate the templated meshes for objects with rotational symmetries. The implementation first creates a line mesh of the geometry using the requested cell size. For 2D objects, the Triangle software is used to generate a volumetric triangle mesh using the line mesh as a surface mesh. The surface mesh is not altered. For 3D objects, Triangle is used to generate triangle surfaces with triangle sizes suitable for the desired volumetric cell size. These triangle surfaces are used to generate the template surface meshes based on the boundary patch partition. For each template a volumetric mesh is generated using Tetgen without inserting Steiner points on any surface mesh.

Two cases with rotational symmetries are investigated:

(i) A 2D n-polygon (Figure 7.5a)

(ii) A 3D open through-silicon via (TSV) structure [28] (Figure 7.5f)

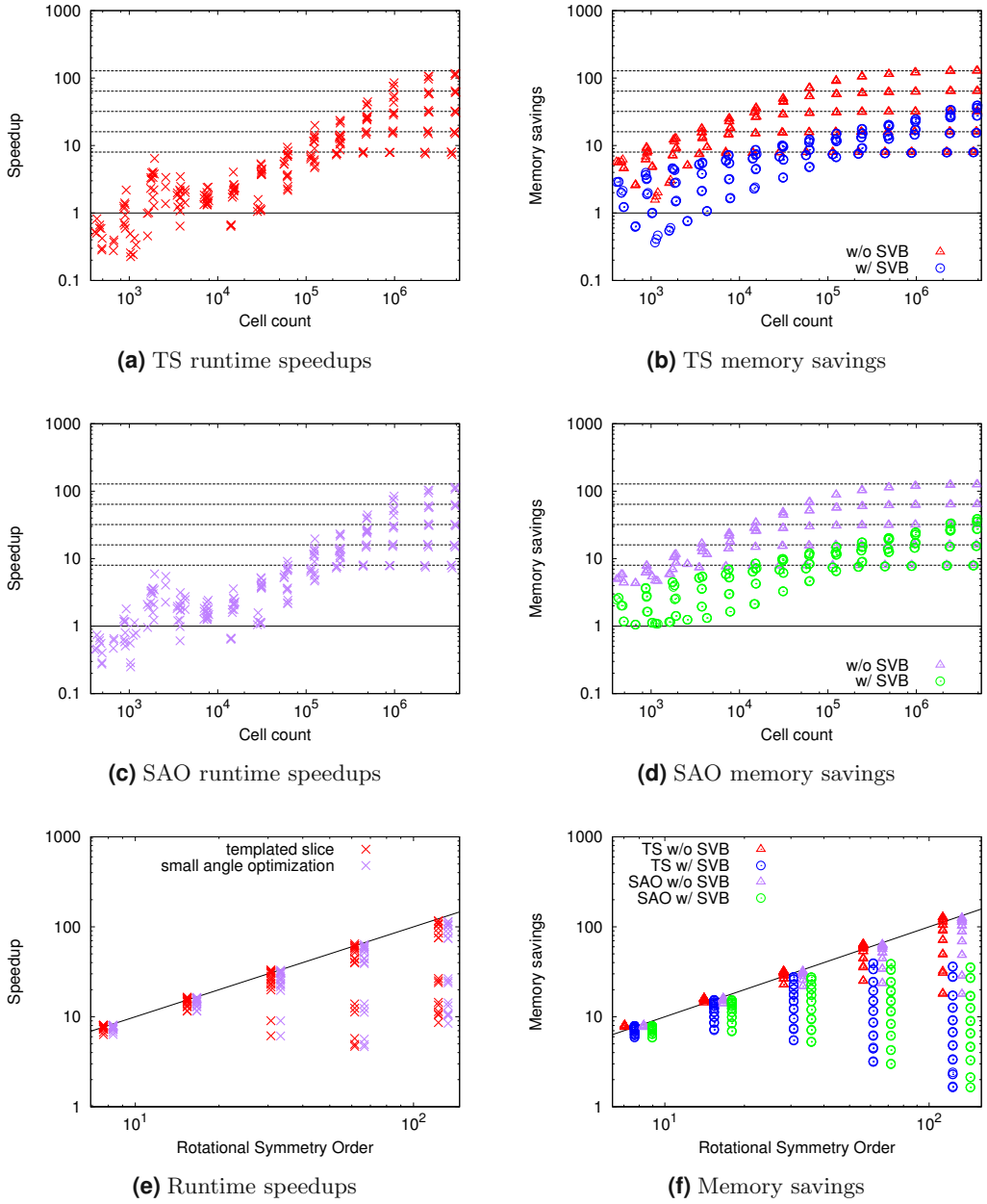
The objects visualized in Figure 7.5 have been synthetically generated. While the first one (the n-polygon) is merely used for a proof-of-concept, the open TSV structure represents an object used in microelectronic applications. Geometries with rotational symmetry orders of $8, 16, 32, 64$, and 128 have been generated for both objects. The concepts presented in Section 6.3 (in this section referred to as slice template or ST) have been applied to all rotationally symmetric objects.

Additionally, the small angle optimization (SAO) approach (cf. Section 6.3.2) has also been used in the benchmarks. Therefore, the conventional mesh generation has been benchmarked against a templated version with and without small angle optimization.

Benchmark results for the 2D n-polygon are given in Figure 7.6. The results are similar to the ones from the previous section: The expected savings in runtime and memory usage are achieved for meshes with a high number of cells. The runtime speedups are larger than a factor of one for cell counts larger than 10^3 and larger than a factor of five for cell counts larger than 5×10^4. The behavior of the memory savings is more stable. Memory savings larger than one are achieved for almost all benchmarks. Memory savings larger than a factor of five are obtained for both approaches (ST and SAO) for cell counts larger than 5×10^3 using the data structure without SVB. The data structure with SVB performs worse. However, memory savings larger than a factor of ten are achieved for large cell count values. The runtime speedups scale well with the rotational symmetry order for cell counts larger than 10^5 (cf. Figure 7.6e). Memory savings also scale well with the rotational symmetry order for meshes with more than 5×10^3 cells (cf. Figure 7.6f). However, memory savings with SVB (visualized using $\circ$) are smaller. For high rotational symmetry orders, the memory savings using a data structure with SVB even starts to decrease (cf. Figure 7.6f) due to the large number of vertices on instance interfaces which increase the bookkeeping effort.

The smallest angle quality parameters $10°, 20°$, and $30°$ are used for the rotational symmetry benchmarks. While the conventional mesh generation algorithm was always able to fulfill the desired quality parameters, the TS approach has issues due to small angles near the center of rotation for high rotational symmetry orders (cf. Section 6.3.2). However, the SAO approach was able to generate meshes with smallest element angles larger than the configured values. A selection of quality histograms is given in Figure 7.7, example meshes of a selected benchmark are visualized in Figure 7.8.

Benchmark results for the 3D open TSV structure are given in Figure 7.9. Runtime speedups from 5 up to 100 can be achieved. However, no correlation to the cell counts of the meshes is observed for the 3D structure. Memory savings without SVB ranging from 10 to 100 are observed (cf. Figure 7.9b). However, with SVB, the memory savings are much smaller ranging from a factor of two to a factor of ten. Similar to the runtime speedup behavior, the memory savings are not correlated to the cell counts. The memory savings, however, are slightly better as depicted by Figure 7.9d. The runtime speedups also scale well with the rotational symmetry order of the object (cf. Figure 7.9e). This is also true for the memory savings when using the data structure without SVB. The behavior when using the data structure with SVB, however, is completely different (cf. Figure 7.9f). The memory savings start to decrease for high rotational symmetry orders. This behavior can be explained by the high vertex count on the instance interfaces for high rotational symmetry orders. A high number of vertices on instance interfaces increases the memory usage of the shared vertex bookkeeping in turn resulting in reduced memory savings.



(a) TS runtime speedups

(b) TS memory savings

(c) SAO runtime speedups

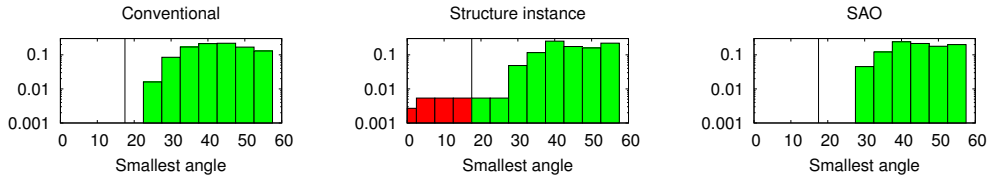
(d) SAO memory savings

(e) Runtime speedups

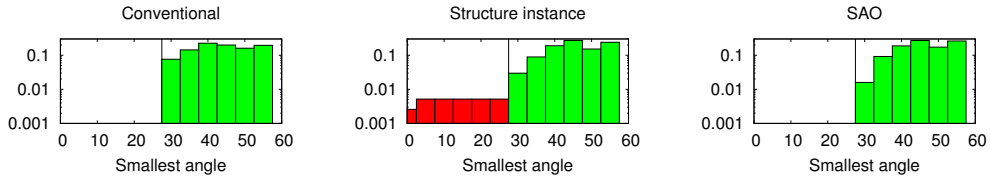
(f) Memory savings

Figure 7.6: Benchmark results for the 2D n-polygon

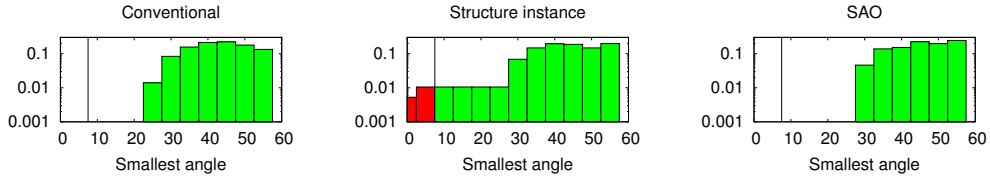
The top row (a)-(b) and middle row (c)-(d) visualizes benchmark results for different cell count values for the TS and SAO approach, respectively. The dashed black lines indicate the expected runtime and memory savings of the different rotational symmetry orders $8, 16, 32, 64$, and 128. Benchmark results for different rotational symmetry orders are visualized in the bottom row (e)-(f). Only meshes with a minimal cell count of 10^5 are considered for the rotational symmetry plots.



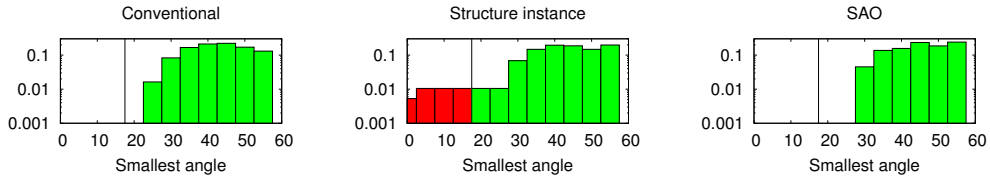
(a) 128-polygon, smallest angle = 20.0°, cell size = 5×10^{-5}



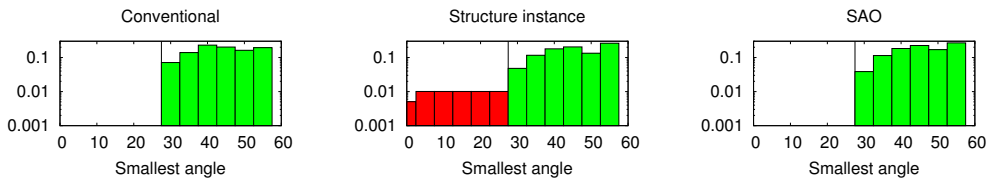
(b) 128-polygon, smallest angle = 30.0°, cell size = 5×10^{-5}



(c) 128-polygon, smallest angle = 10.0°, cell size = 0.0001



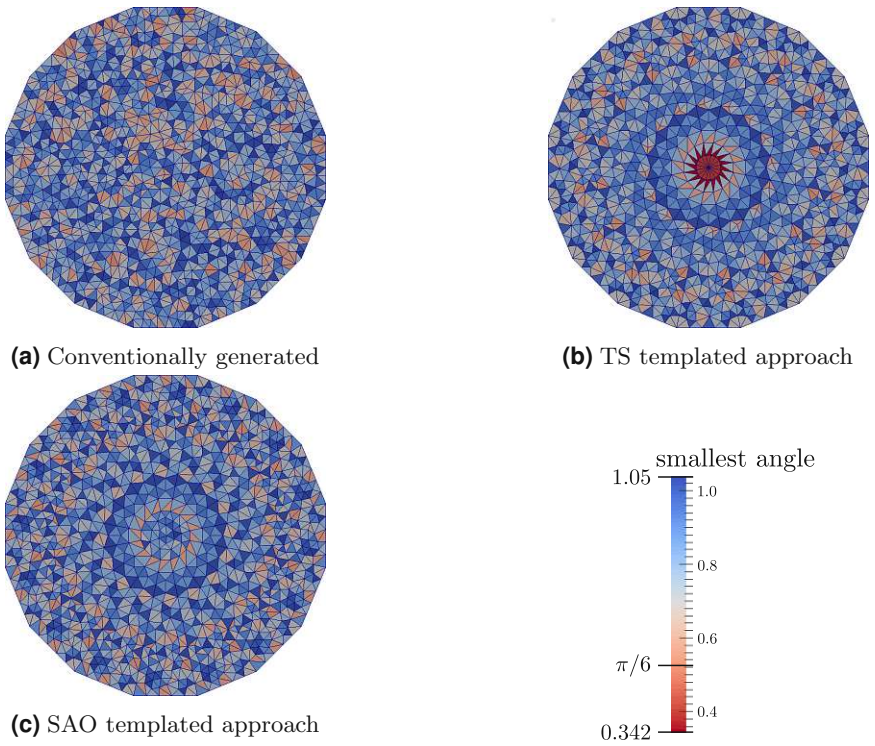
(d) 128-polygon, smallest angle = 20.0°, cell size = 0.0001



(e) 128-polygon, smallest angle = 30.0°, cell size = 0.0001

Figure 7.7: Smallest angle quality histograms for n-polygon benchmarks

A selection of smallest angle quality histograms of meshes of n-polygons. The histogram bins are given in log scale for better visualization of the *bad quality* bins. The quality histograms of the conventionally generated meshes are visualized in the left column. The quality histogram of the structure instance of the TS is given in the middle column. The right column shows quality histograms of the SAO approach. Bins colored in red indicate cells with quality worse than the configured value. The smallest angle quality value constraint is visualized using the black vertical line. The smallest angle of the TS approach structure instance of every presented benchmark does not fulfill the configured values as indicated by the red histogram bins. Additionally, the mesh quality of the structure instances of the SAO approach is even better then the quality of the corresponding conventionally generated mesh as indicated by a slightly higher dominance of larger smallest angles in the quality histograms of the SAO approach.



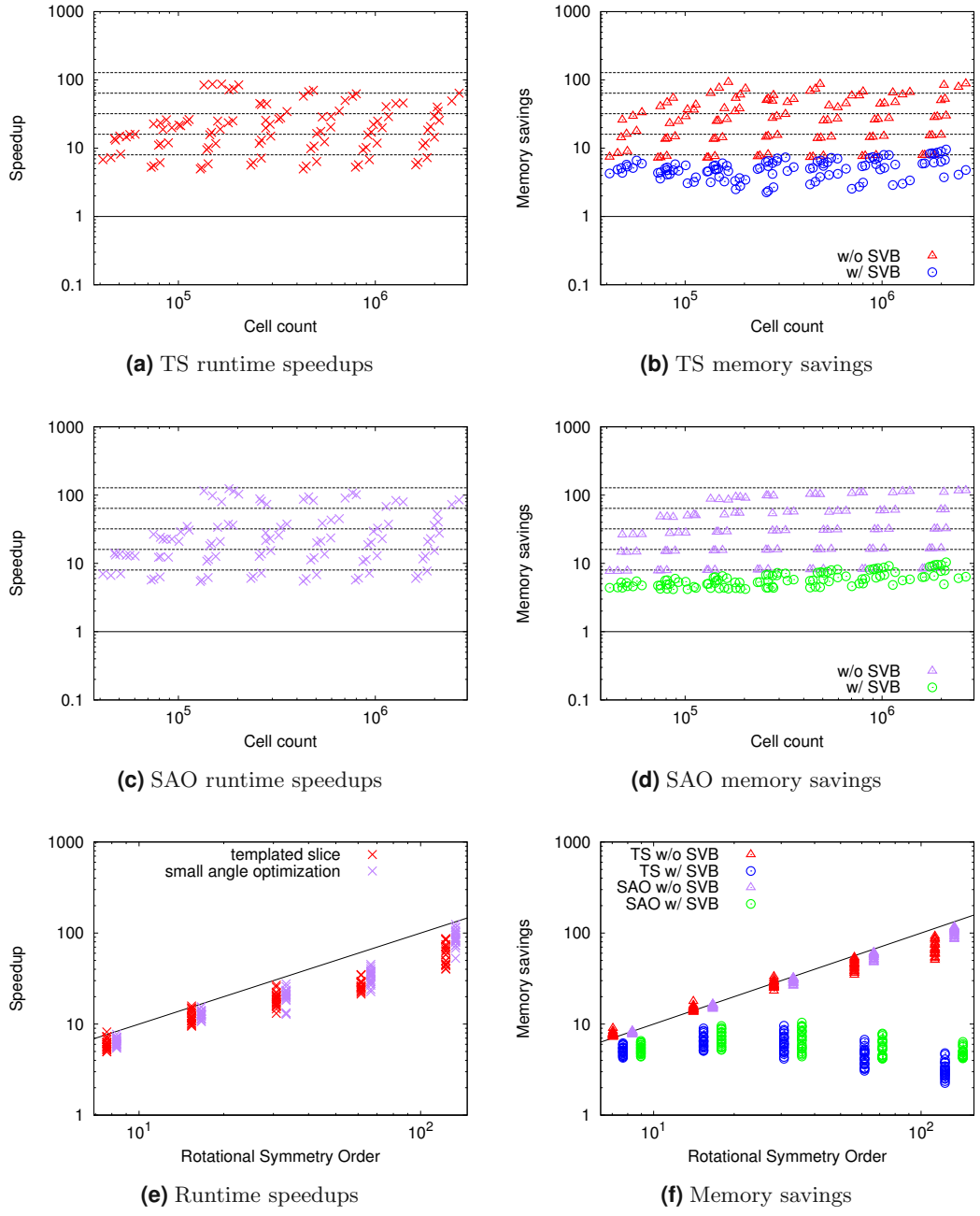
(a) Conventionally generated

(b) TS templated approach

(c) SAO templated approach

Figure 7.8: Quality comparison of meshes generated for a 16-polygon

All meshes have been generated with a requested cell size of 2.5×10^{-3} and a smallest angle of 30°. Elements near the center of (b) have a smallest angle which violates the smallest angle constraint. The conventionally generated mesh (a) and the SAO approach (c) both fulfill the requested element quality.



(a) TS runtime speedups

(b) TS memory savings

(c) SAO runtime speedups

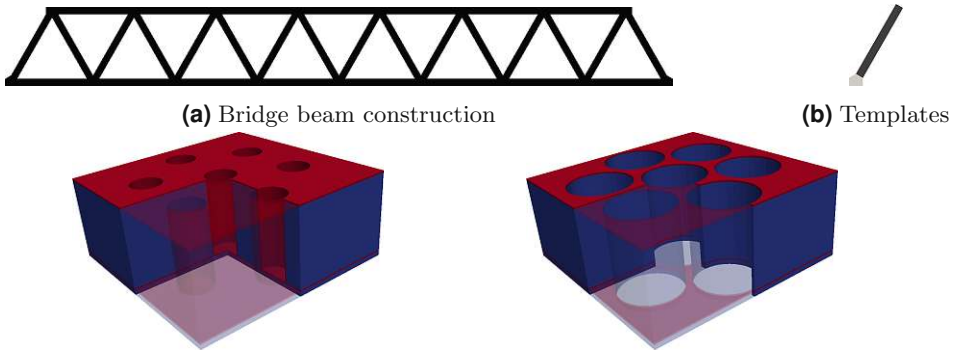
(d) SAO memory savings

(e) Runtime speedups

(f) Memory savings

Figure 7.9: Benchmark results for the 3D open TSV structure

The top row (a)-(b) and middle row (c)-(d) visualizes benchmark results for different cell count values for the template slice and small angle optimization approach, respectively. The dashed black lines indicate the expected runtime speedups and memory savings of the different rotational symmetry orders $8, 16, 32, 64$, and 128. Benchmark results for different rotational symmetry orders are visualized in the bottom row (e)-(f). Only meshes with a minimal cell count of 10^5 are considered for the rotational symmetry plots.



(a) Bridge beam construction **(b)** Templates

(c) Multiple open TSV structure **(d)** Trunk template

Figure 7.10: Benchmark objects with similarities

The geometries of the beam bridge construction and the multiple open TSV structure are visualized in (a) and (c), respectively. (b) and (d) show the templates for the bridge and the multiple open TSV structure, respectively. The bridge beam construction has two different templates being the beam and the beam connector. The templated structure consists of 31 beam instances and 17 beam connector instances. The multiple open TSV structure has two templates, being the trunk (d) and the slice template used in Section 7.3

7.4 General Similarities

Templated meshes for objects with general similarities are generated using the same setup as described in Section 7.3. Two objects with similarities, one from the field of structural mechanics and one from the field of microelectronics, are investigated in this section:

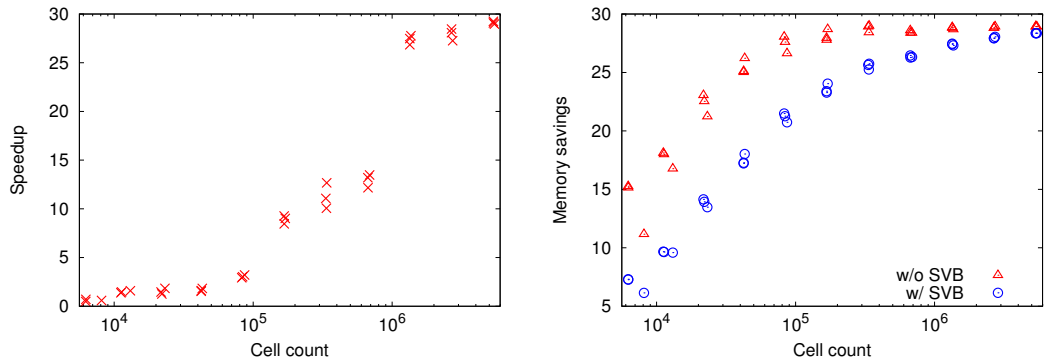
(i) A 2D bridge beam construction (Figure 7.10a)

(ii) A 3D structure with multiple open TSVs [27] (Figure 7.10c)

The expected memory savings and runtime speedups for both objects are not trivial to calculate due to their heterogeneous templated structures. However, the expected memory savings for uniform cell sizes are estimated using the ratio of the sum of the template volumes and the structure instance volume. Memory savings of 28.5 are estimated for the bridge beam construction. The expected memory savings for the multiple open TSV structure are estimated to be a much lower value of 1.6. This low estimation originates from the trunk template. It is only instanced once in the structure instance and has a relative large volume (compared to the volume of the TSVs).

Benchmark results for the bridge beam construction are given in Figure 7.11. Similar to the previous chapters, expected runtime speedups and memory savings are observed for cell counts larger than 10^6. Runtime speedups range from 10 to 14 for meshes with more than 10^5 cells and even exceed 28 for meshes with more than 10^6 cells.
Memory savings, when using the templated mesh data structure without SVB, are larger than 25 for meshes with more than 4×10^4 cells. When using the templated mesh data structure with SVB, the same memory savings are observed for meshes with more than 4×10^5 cells. These benchmark results coincide with the (estimated) expected memory savings.

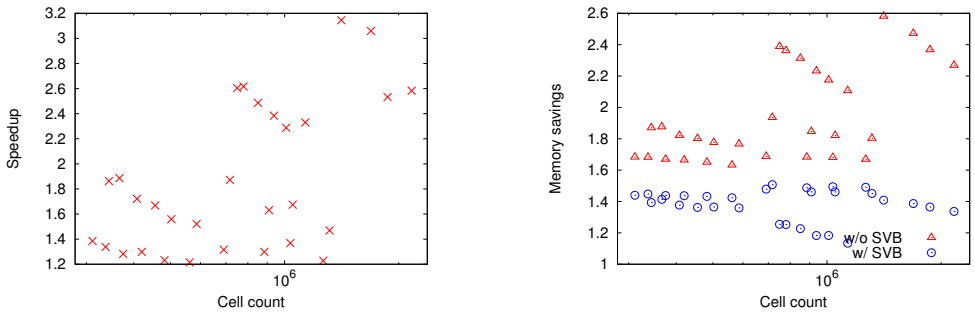
Benchmark results for the multiple 3D open TSV structure are given in Figure 7.12. Runtime speedups range from 1.2 to 3.2, memory savings range from one to 1.5 when using the data structure with SVB and from 1.6 to 2.6 when using the data structure without SVB. However, in contrast to the 2D bridge beam construction, no correlation of the mesh cell count and the memory savings is observed. These benchmark results also coincide well with the (estimated) expected memory savings.



(a) Runtime speedups **(b)** Memory savings

Figure 7.11: Benchmark results for the 2D bridge beam construction

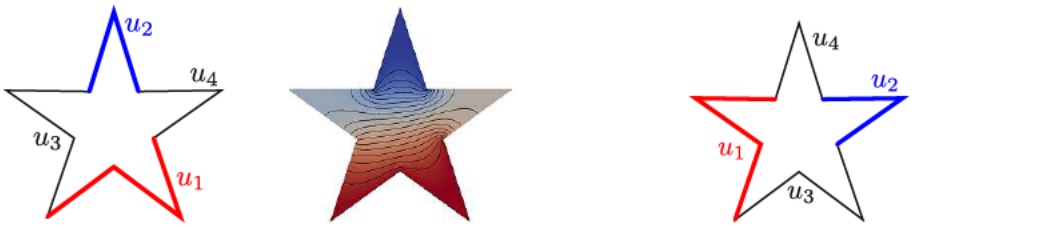
Runtime and memory benchmark results are visualized for different cell count values on the left and on the right, respectively. The expected runtime speedups and memory savings are observed for high cell counts (larger than 10^6).



(a) Runtime speedups **(b)** Memory savings

Figure 7.12: Benchmark results for the multiple open TSV structure

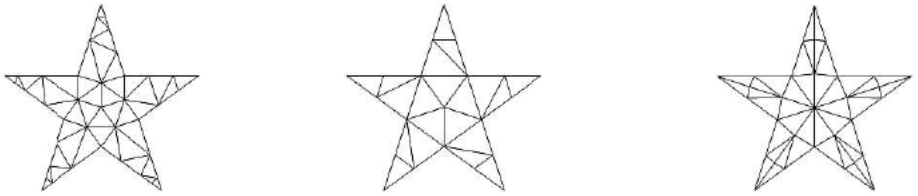
Runtime and memory benchmarks results are visualized for different cell count values on the left and on the right, respectively. No correlation of the mesh cell count and the memory savings is observed.



(a) Boundary conditions **(b)** Solution **(c)** Boundary conditions of the dual problem

Figure 7.13: The Gummel Star with non-symmetric boundary conditions

Boundaries colored in red and blue are Dirichlet boundaries with constant value $u_1 = 1$ and $u_2 = 0$, respectively. The remaining boundaries u_2 and u_3 are Neumann boundaries. The solution to that boundary value problem is visualized in (b). The boundary conditions of the dual problem are visualized in (c).



(a) Non-symmetric **(b)** Rotational symmetric **(c)** Reflective-rotationally symmetric

Figure 7.14: Non-symmetric and symmetric example meshes of the Gummel Star

Figure 7.15: All rotations of the non-symmetric Gummel Star boundaries

7.5 FEM-based Symmetry Analysis using a Five-pointed Star

An investigation of the effects of the templated approach on the solution of a simulation problem with symmetric geometry is presented in this section. In particular, a boundary value problem based on the Laplace equation [57] is investigated on a symmetric five-pointed star. This simulation problem, visualized in Figure 7.13, is referred to as *Gummel Star* [126] and has the following two interesting properties: First, due to the rotational symmetry of the simulation domain, rotating the boundary conditions results in a solution which is equal to the rotated solution of the initial problem. Second, the dual problem – the simulation problem where the Dirichlet and the Neumann boundaries are swapped (cf. Figure 7.13c) – is equal to the *reflected* simulation problem, hence resulting in a solution which is equal to the reflected solution of the initial problem. However, as stated in Chapter 1, these two properties might not hold for numerical solutions when using non-symmetric meshes.

To investigate the effects on the numerical solution, three types of meshes have been generated for different target cell sizes: a non-symmetric, a rotationally symmetric, and a reflective-rotationally symmetric mesh. All meshes have been created with a smallest angle quality larger than 30° [109]. Example meshes for each type are visualized in Figure 7.14. For each mesh and each rotation of the boundary conditions (cf. Figure 7.15), solutions to the boundary value problem as well as the dual boundary value problem have been calculated using the free open source FEM software framework FEniCS [4]. These solutions are compared by first rotating (for the rotated boundary conditions) or reflecting (for the dual problem) them to be aligned and then applying the maximum norm on the transformed solutions. The solution difference of two solutions u and v is therefore defined as:

$$d(u,v) := \max_{x\in D} \|u(x) - v(T(x))\|_\infty = \max_{x\in D} |u(c) - v(T(x))| \tag{7.1}$$

D is the simulation domain and T is the transformation which aligns the solution v to the solution u. It is to be expected that meshes with rotational symmetries yield numerically equal solutions when rotating the boundary conditions. Additionally, meshes with reflective symmetries are expected to yield numerically equal solutions when comparing the initial problem to the dual problem.

The solution differences for the rotated boundary conditions on meshes with two different cell counts are given in Table 7.2 and Table 7.3. Each row and each column represents the solution to the boundary value problem with rotated boundary conditions. The differences on the non-symmetric meshes are at least two orders of magnitude, for the coarser meshes (cf. Table 7.2) even three orders of magnitude, larger then the differences on the symmetric meshes. Additionally, keeping in mind that the largest value of the solution to this boundary value problem is one, the errors for the non-symmetric mesh range up to 11%.

The solution differences for the initial and the dual problem for meshes with different cell counts are given in Table 7.4. The solution differences are at least two orders of magnitude smaller when using a reflective-rotationally symmetric mesh compared to a non-symmetric mesh. Additionally, due to the lack of reflective symmetry, the solution differences for the rotationally symmetric mesh are about one order of magnitude larger than the differences for the reflective-rotationally symmetric mesh.

As expected, meshes with rotational symmetries yield solutions which are (numerically) equal to each other when rotating the boundary conditions of the Gummel star. Also, meshes with reflective symmetries yield solutions to the initial and the dual problem which are, again, numerically equal.

	No rotation	72° Rotation	144° Rotation	216° Rotation	288° Rotation
Non-symmetric mesh with 109 triangles					
No rotation	0	5.0×10^{-2}	1.1×10^{-1}	7.2×10^{-2}	4.2×10^{-2}
Rotation 72°		0	1.1×10^{-1}	7.0×10^{-2}	5.6×10^{-2}
Rotation 144°			0	1.1×10^{-1}	1.1×10^{-1}
Rotation 216°				0	1.0×10^{-1}
Rotation 288°					0
Rotationally symmetric mesh with 120 triangles					
No rotation	0	1.1×10^{-5}	9.8×10^{-6}	6.1×10^{-6}	8.4×10^{-6}
Rotation 72°		0	1.2×10^{-5}	6.7×10^{-6}	9.7×10^{-6}
Rotation 144°			0	7.4×10^{-6}	6.6×10^{-6}
Rotation 216°				0	8.4×10^{-6}
Rotation 288°					0
Reflective-rotationally symmetric mesh with 130 triangles					
No rotation	0	1.6×10^{-5}	1.2×10^{-5}	9.4×10^{-6}	1.6×10^{-5}
Rotation 72°		0	1.6×10^{-5}	1.0×10^{-5}	1.2×10^{-5}
Rotation 144°			0	8.5×10^{-6}	1.0×10^{-5}
Rotation 216°				0	1.6×10^{-5}
Rotation 288°					0

Table 7.2: Solution differences for the rotated boundary conditions on meshes with about 1.2×10^2 triangles. All solution differences are evaluated by rotating the solutions to a position where they are aligned and then applying the maximum norm. The solution differences for the non-symmetric mesh are at least three orders of magnitude larger than the solution differences for the rotationally and the reflective-rotationally symmetric mesh. The solution differences for the rotationally and reflective-rotationally symmetric mesh are not exactly zero due to numeric issues and round-off errors. However, these differences indicate that these solutions are numerically equal.

	No rotation	72° Rotation	144° Rotation	216° Rotation	288° Rotation
Non-symmetric mesh with 13921 triangles					
No rotation	0	1.1×10^{-2}	2.5×10^{-2}	3.5×10^{-2}	1.9×10^{-2}
Rotation 72°		0	2.5×10^{-2}	3.5×10^{-2}	1.9×10^{-2}
Rotation 144°			0	1.3×10^{-2}	1.5×10^{-2}
Rotation 216°				0	2.2×10^{-2}
Rotation 288°					0
Rotationally symmetric mesh with 13850 triangles					
No rotation	0	9.2×10^{-5}	8.0×10^{-5}	7.7×10^{-5}	9.2×10^{-5}
Rotation 72°		0	8.4×10^{-5}	7.7×10^{-5}	8.0×10^{-5}
Rotation 144°			0	7.9×10^{-5}	8.0×10^{-5}
Rotation 216°				0	8.9×10^{-5}
Rotation 288°					0
Reflective-rotationally symmetric mesh with 14160 triangles					
No rotation	0	1.1×10^{-4}	1.3×10^{-4}	9.5×10^{-5}	1.1×10^{-4}
Rotation 72°		0	9.9×10^{-5}	8.0×10^{-5}	1.3×10^{-4}
Rotation 144°			0	7.9×10^{-5}	8.0×10^{-5}
Rotation 216°				0	9.9×10^{-5}
Rotation 288°					0

Table 7.3: Solution differences for the rotated boundary conditions on meshes with about 1.4×10^4 triangles. All solution differences are evaluated by rotating the solutions to a position where they are aligned and then applying the maximum norm. The solution differences for the non-symmetric mesh are at least two orders of magnitude larger than the solution differences for the rotationally and the reflective-rotationally symmetric mesh. The solution differences for the rotationally and reflective-rotationally symmetric mesh are not exactly zero due to numeric issues and round-off errors. However, these differences indicate that these solutions are numerically equal.

Symmetry type	Cell count	Solution difference using the maximum norm
Non-symmetric	109	5.0×10^{-2}
Rotationally symmetric	120	1.4×10^{-2}
Reflective-rotationally symmetric	130	1.6×10^{-5}
Non-symmetric	1758	2.4×10^{-2}
Rotationally symmetric	1740	3.2×10^{-3}
Reflective-rotationally symmetric	1750	6.5×10^{-5}
Non-symmetric	13921	1.2×10^{-2}
Rotationally symmetric	13850	1.1×10^{-3}
Reflective-rotationally symmetric	14160	1.1×10^{-4}
Non-symmetric	111244	1.2×10^{-2}
Rotationally symmetric	111260	1.1×10^{-3}
Reflective-rotationally symmetric	111660	2.8×10^{-4}

Table 7.4: Solution differences of the initial problem and its dual problem for meshes with different symmetries and cell counts. Solution differences are evaluated by reflecting the dual solution to make it aligned to the solution of the initial problem and then applying the maximum norm. As expected, the differences for the reflective-rotationally mesh are always less then the differences for the non-symmetric or the rotationally symmetric mesh. However, the rotationally symmetric meshes yields solutions which are more similar than the solutions for the non-symmetric mesh.

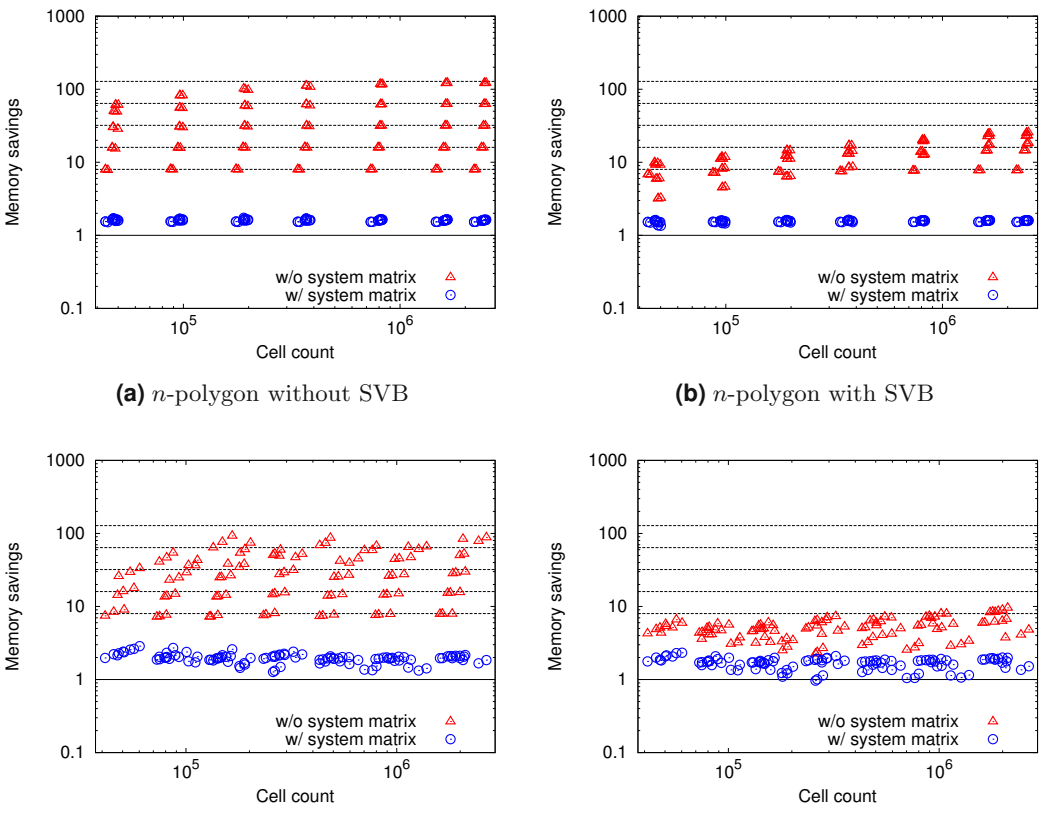
7.6 Memory and Runtime Benefits in FEM applications

Volumetric meshes are widely used for simulations, especially discretization-based simulations like the FEM. This section covers memory savings of templated meshes, where also the memory usage of a FEM simulation is considered. Up to this point, memory savings are calculated by taking the quotient of the memory usage of a templated mesh and the memory usage of a conventionally generated mesh. The memory savings in this section, however, also include the required memory for a FEM simulation for the templated mesh and the conventionally generated mesh.

The FEM discretization is usually stored using a square matrix, called the system matrix, which is commonly sparse [103]. The size and the sparsity pattern of that matrix depends on the topology of the mesh on which the simulation is carried out. Different sparse matrix formats have been proposed. For example, the compressed sparse row (CSR) format, which is used in this work, is a reasonable, general purpose sparse matrix storage format which usually allows for good performance [141]. When using linear FEM, the focus in this section, the unknowns in the system matrix can be identified with the mesh vertices.

A naive approach for storing the system matrix of a templated mesh is to use the system matrix of its structure instance. Memory benchmark results which include the required memory size for the system matrix of a linear FEM using this naive approach are given in Figure 7.16. It can be seen that the memory savings which include the system matrix sizes are significantly lower than the memory savings without the system matrix sizes. Memory savings including the system matrix only range from one (no saving) to three.

However, it is possible to use a templated approach for storing the system matrix. Listing 7.1 presents the `TCSR_MATRIX` data structure which represents the system matrix of a templated mesh which uses a templated approach for memory optimization. For each mesh template its system matrix is calculated as usual assuming that every mesh template vertex is an unknown. This template matrix is stored in the CSR format using the `CSR_MATRIX` data structure. Additionally, for each instance a mapping is required to map the local unknowns to global unknowns. This is achieved by the `local_to_global_mapping` in the templated CSR matrix data structure. If the instance of a template mesh vertex within the structure instance is part of a Dirichlet boundary and therefore not an unknown, the corresponding mesh vertex is mapped to an invalid value, i.e., to -1, for that particular mesh instance. Using this mapping, a templated CSR matrix can be converted to a classical CSR matrix using Algorithm 7.1. This algorithm iterates over all instances of all templates and adds the matrix entries of the local template matrix to the corresponding global CSR matrix entries using the `local_to_global_mapping`. Vertices which are on the *inside* of an instance only have neighbor vertices inside that instance. Therefore, these global unknowns are updated only once in the global CSR matrix. For a local unknown, which associated vertex is in any instance interface, the local coupling with neighbor vertices is only calculated using the local cell connections. When transforming these local vertices to the structure instance, the sum in Line 14 (cf. Algorithm 7.1) adds up all valid couplings of all mesh instances in which the vertex is. This sum reflects the linearity of the inner product which is used to calculate the coupling for a FEM problem. Additionally, the evaluation of that coupling is invariant under rigid transforms [103]. Therefore, the resulting CSR matrix of Algorithm 7.1 with the input being a templated CSR matrix based on a templated mesh will be the same as the CSR matrix of its structure instance.



(a) n-polygon without SVB

(b) n-polygon with SVB

(c) Open TSV structure without SVB

(d) Open TSV structure with SVB

Figure 7.16: Memory benchmark results including the FEM system matrix

The memory savings including the system matrix are calculated using the quotient of the memory usage of the conventionally generated meshes plus the memory usage for its system matrix and the memory usage of the templated mesh plus the usage of the system matrix of its structure instance. The rotationally symmetric objects n-polygon and the open TSV structure presented in Section 7.3 are used for this benchmark.

Listing 7.1: Templated CSR matrix data structure

```
struct CSR_MATRIX
{
  NUMERIC_TYPE *    values;
  int *             column_indices;
  int *             offsets;
  int size;
};

struct TCSR_INSTANCE
{
  int *             local_to_global_mapping;
};

struct TCSR_TEMPLATE
{
  CSR_MATRIX        template_matrix;
  TCSR_INSTANCE *   instances;
  int               instance_count;
};

struct TCSR_MATRIX
{
  TSCRTemplate *    templates;
  int               template_count;
};
```

input : Templated CSR matrix A
output : CSR matrix B

begin
  $B \leftarrow$ zero matrix
  foreach *TSCRTemplate* $T \in A$ **do**
    foreach *TCSR_INSTANCE* $I \in T$ **do**
      for *row* $\leftarrow 1$ **to** *T.template_matrix.size* **do**
        $i_{\text{start}} \leftarrow$ T.template_matrix.offsets[row]
        $i_{\text{end}} \leftarrow$ T.template_matrix.offsets[row+1]
        for $i \leftarrow i_{\text{start}}$ **to** i_{end} **do**
          column $\leftarrow$ T.template_matrix.column_indices[i]
          value $\leftarrow$ T.template_matrix.values[i]
          global_row $\leftarrow$ I.local_to_global_mapping[row]
          global_column $\leftarrow$ I.local_to_global_mapping[column]
          if *global_row* $\neq -1 \wedge$ *global_column* $\neq -1$ **then**
            B(global_row, global_column) $\leftarrow B$(global_row, global_column)+ value
          end
        end
      end
    end
  end
end

Algorithm 7.1: Convert templated CSR matrix to CSR matrix

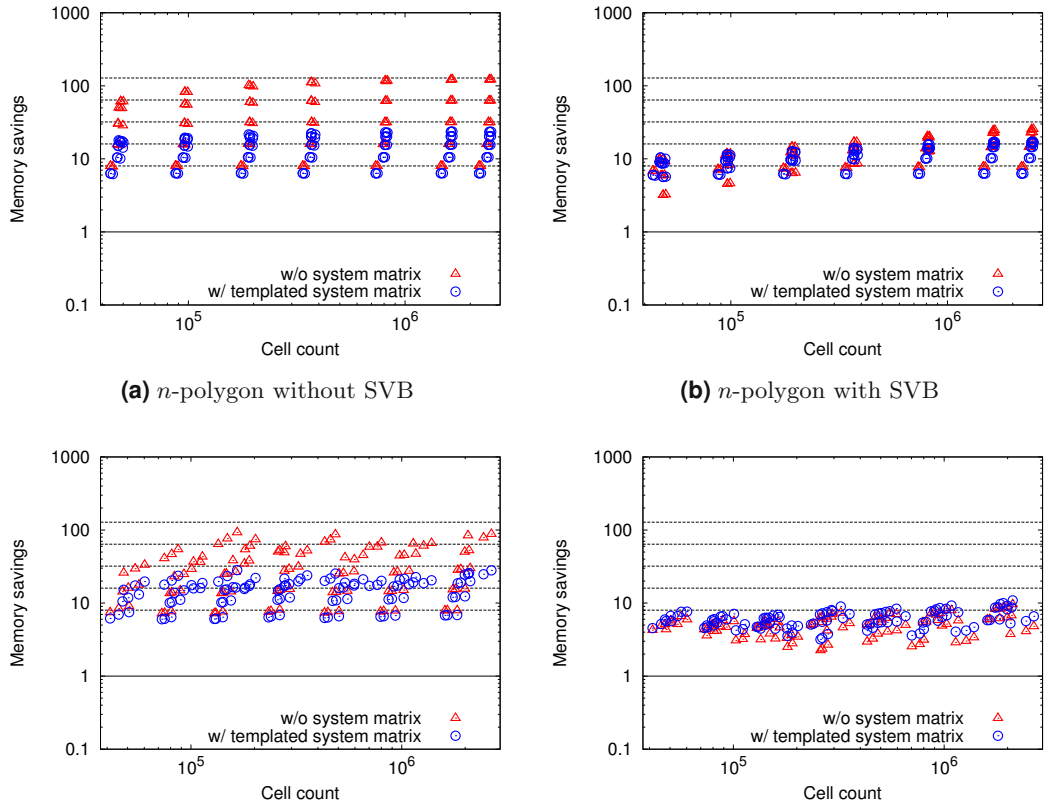
Solution methods for FEMs are typically based on iterative algorithms utilizing Krylov subspaces, for example the conjugate gradient algorithm [141]. One main advantage of Krylov subspace methods is, that the system matrix is not altered. The main matrix operation used in each iteration step is the matrix-vector product. The matrix-vector product for a templated CSR matrix is presented in Algorithm 7.2.

```
   input  : Templated CSR matrix A, vector b
   output : Vector result
 1 begin
 2     result ← zero vector
 3     foreach TSCRTemplate T ∈ A do
 4         foreach TCSR_INSTANCE I ∈ T do
 5             for row ← 1 to T.template_matrix.size do
 6                 i_start ← T.template_matrix.offsets[row]
 7                 i_end ← T.template_matrix.offsets[row+1]
 8                 for i ← i_start to i_end do
 9                     column ← T.template_matrix.column_indices[i]
10                     value ← T.template_matrix.values[i]
11                     global_row ← I.local_to_global_mapping[row]
12                     global_column ← I.local_to_global_mapping[column]
13                     if global_row ≠ −1∧ global_column ≠ −1 then
14                         result[global_row] ← result[global_row] + value ×
                           b[global_column]
15                     end
16                 end
17             end
18         end
19     end
20 end
```

Algorithm 7.2: Templated CSR matrix-vector product

Memory benchmark results using the templated CSR matrix data structure rather than a (classical) CSR matrix of the structure instance are presented in Figure 7.17. The memory savings including the system are significantly higher when using the templated CSR matrix data structure instead of using a CSR matrix of the structure instance. When storing the templated mesh using the data structure with SVB, the memory savings including the templated system matrix are as good as the case without considering the system matrix (compare Figure 7.17b and Figure 7.17d). In some cases, the memory savings including the templated system matrix are even better than the memory savings which do not consider any system matrix because the overhead of the shared vertex bookkeeping is less significant.

The templated CSR matrix-vector product algorithm, however, is more complex due to the additional array lockup in Algorithm 7.2, Lines 11 and 12. Runtime benchmarks for the matrix-vector product using a CSR matrix and a templated CSR matrix are given in Figure 7.18. The matrix-vector product using a templated CSR matrix is about two to six times slower than the matrix-vector product using a conventional CSR matrix. Therefore, the solution process is also expected to take two to six times longer. This might be an issue, if computation times are important. However, for memory-limited computing platforms the templated CSR matrix approach supports larger simulation meshes.



(a) n-polygon without SVB

(b) n-polygon with SVB

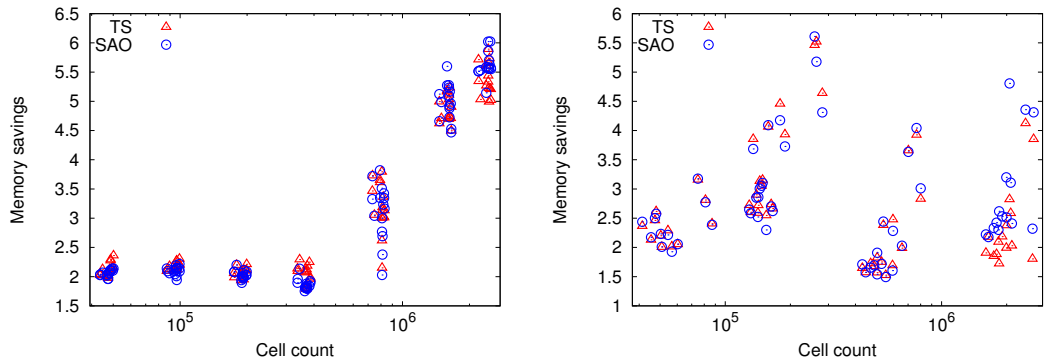
(c) Open TSV structure without SVB

(d) Open TSV structure with SVB

Figure 7.17: Memory benchmark results using the templated CSR matrix

Memory savings for the templated mesh including the templated CSR matrix are calculated based on the memory usage of a conventionally (non-symmetric) mesh including the memory usage of its CSR matrix.

For most typical FEM applications, like fluid dynamics or electronic device simulation, linear solvers suffer from slow convergence as well as lack of robustness [141]. To compensate these issues, preconditioners are used. Black-box preconditioners, like the ILU factorization or the block Jacobi preconditioner, require access to the entries of the system matrix. Accessing entries of the system matrix is a potential issue when using the templated CSR matrix data structure, because the system matrix entries of boundary vertex unknowns are not directly provided in the templated CSR matrix data structure. However, matrix-free preconditioners are an active field of research and complement the presented approach based on templated CSR matrices [45][117].



(a) n-polygon without SVB

(b) 3D open TSV structure

Figure 7.18: Runtime benchmark results for templated CSR matrix-vector product

Relative runtime values are calculated by the relation of the templated CSR matrix-vector product runtime and the CSR matrix-vector product of the conventionally generated mesh system matrix. A correlation between runtime values and the number of cells can only be observed for the 2D n-polygon structures. For the 3D open TSV structures, no correlation can be observed because the number of neighbor vertices is less restricted (compared to 2D high quality meshes) in high quality 3D meshes.

Chapter 8

Conclusion and Outlook

This thesis presents theoretical approaches as well as algorithms and data structures for storing, generating, and adapting meshes with symmetries and/or similarities. A theoretical background for geometries and meshes has been presented in Chapter 2. Related research work has been investigated and discussed in Chapter 3. A theory for templated structures has been developed (cf. Chapter 4), which has been used to formulate general mesh generation and adaptation algorithms (cf. Chapter 5). Special (less complicated) algorithms for reflective and rotational symmetries (and their combinations) have been proposed in Chapter 6. The developed data structures and algorithms have been evaluated in Chapter 7.

This last chapter concludes the thesis by re-evaluating the research questions stated in the beginning (Section 8.1) and provides an outlook to possible further research directions (Section 8.2).

8.1 Conclusion

The thesis is evaluated in this section by answering the stated research questions (cf. Chapter 1) based on the conducted research. The general conclusion, however, is that the memory savings and runtime speedups meet or even exceed the expectations in most cases. Additionally, symmetric meshes increase the accuracy of FEM solutions in most (about three out of four) cases. In the following, the research questions will be evaluated one after another.

How can templated meshes and templated geometries be defined? Are there any restrictions or issues? Which restrictions and which issues apply to objects with symmetries?

Templated meshes and geometries have been defined in Section 4.1. The definitions and lemmas provide solid and abstract mechanisms for geometries and meshes with symmetries and/or similarities.

As discussed in Section 4.2, issues arise at locations where two template instances meet (cf. Lemma 4.2). The conformity has to be ensured at these locations. To identify and handle these issues in algorithms, the boundary patch partition (cf. Section 4.3) has been developed. Additional issues arise for templated structures with irregular instance graphs as some mesh generation and adaptation algorithms get more intricate (cf. Section 5.1.2 and Section 5.2.2)

However, for objects with reflecting symmetries, there are no issues or restrictions as shown in Section 6.2 (cf. Lemma 6.1) and Section 6.4.1 (cf. Lemma 6.2), which is also true for objects which have rotational and reflective symmetries (cf. Section 6.4.2). Conformity issues potentially occur for objects with rotational symmetries on instance interfaces of slice templates. However, the boundary patch partition is much easier to determine for such objects and the resulting instance graph is regular (cf. Section 6.3).

How can properties, like the Delaunay property, be abstracted to templated meshes?

The template-aware locally Delaunay property (cf. Section 4.4) is the natural abstraction of the locally Delaunay property for templated meshes. Similar to the Delaunay Lemma, a lemma for evaluating the Delaunay property of a templated mesh has been devised (cf. Lemma 4.5) and proved. Additionally, an algorithm has been developed which adapts a templated mesh in a way that its structure instance becomes Delaunay (cf. Algorithm 5.8 in Section 5.2.5).

Which algorithms for conventional meshes also work for templated meshes? Which modifications are required for these algorithms?

Two general mesh generation algorithms for templated meshes have been developed and presented in Section 5.1:

(i) An algorithm which generates volumetric meshes for each template independently and then adapts the instance interfaces to ensure conformity (cf. Algorithms 5.1 to 5.4).

(ii) An algorithm which at first generates a conforming surface mesh for all templates and then uses these surface mesh to generate volumetric mesh templates (cf. Algorithms 5.5 to 5.7).

Both algorithms utilize conventional mesh generation algorithms for surface and volumetric mesh generation. Thus, existing implementations of mesh generation algorithms can be re-used for templated mesh generation algorithms. Although the second algorithm requires the boundary patch partition and thus is more complicated, it is numerically more stable and generates meshes of higher quality.

Issues and modifications for mesh adaptation algorithms have been discussed and presented in Section 5.2. Mesh adaptation algorithms do not require any modification as long as they work on *interior* elements of a mesh template. If an algorithm alters the surface of a mesh template, the conformity of the structure instance has to be preserved (cf. Lemma 4.2). Thus, mesh adaptation algorithms have less freedom in modifying elements on the surface of mesh templates. In some cases, certain modifications are even forbidden.

How can a templated mesh be generated based on symmetric geometries?

The two general mesh generation algorithms presented in Section 5.1 work for symmetric geometries. However, due to the simplicity of reflective and rotational symmetries (compared to general similarities), specializations of these algorithms have been proposed for geometries with reflective symmetries, rotational symmetries, and their combination (cf. Chapter 6). These algorithms are less complicated, because they take direct advantage of properties induced by these symmetries. For example, reflective symmetries do not have conformity issues on instance interfaces (cf. Lemma 6.1 and Lemma 6.2).

What are the effects on mesh element quality when using a templated mesh generation algorithm compared to a conventional mesh generation algorithm? How do non-symmetric meshes (of symmetric geometries) affect the solution of simulation processes?

It has been shown that the quality of the templated approach is as good as the quality of the conventional approach in most cases and minimally worse otherwise. An analysis on mesh element quality for reflective and rotational symmetric objects is given in Section 7.2 and Section 7.3, respectively.

If a simulation problem has a symmetric geometry, transforming the boundary conditions with the symmetry transformation yields a mathematical solution which is equal to the solution of the initial problem transformed by that same transformation. However, a FEM-based symmetry analysis yields that utilizing non-symmetric meshes result in numerical solutions which are numerically not equal (cf. Section 7.5). A symmetric mesh is required to obtain numerically equal solutions.

How much memory can be saved when using templated meshes?

Expected memory savings are achieved for all 2D benchmark objects when using the data structure without SVB. For 3D rotational symmetric benchmark objects with high rotational symmetry orders, the savings are slightly lower than the expected values. When using the data structure with SVB, memory savings are lower due to the bookkeeping effort. Benchmarks on memory usage for templated meshes have been presented in Section 7.2, Section 7.3, and Section 7.4. The best memory savings for each benchmark object is given in Table 8.1.

When additionally considering the memory usage of the FEM system matrix, memory savings drop significantly. Therefore, a templated matrix data structure based on the CSR matrix format has been proposed which compensates these losses (cf. Section 7.6). However, due to the increased complexity of the data structure, the matrix-vector product of that templated CSR matrix format is two to six times slower than the matrix-vector product using a classical CSR matrix. In cases where the simulation on a target computing platform is memory-limited or runtime is no issue, the templated CSR matrix approach enables the use of larger meshes.

What are the effects on the runtime performance for templated mesh generation algorithms?

Expected runtime speedups are achieved or even exceeded for most benchmark objects. Only for 3D open TSV objects with a high rotational symmetry order, the speedup is slightly smaller than expected. Benchmark results on runtime speedup for templated mesh generation have been presented in Section 7.2, Section 7.3, and Section 7.4. The best runtime speedup for each benchmark object is given in the rightmost column of Table 8.1.

8.2 Outlook

Although important research questions have been answered (cf. Section 8.1), additional research is required to further optimize the utilization of symmetries and similarities for mesh generation processes. Some potential research suggestions are given in the following.

The theory presented in Appendix A is a solid background for this work. However, the sets used for mesh elements and geometries are restricted to linear sets due to the IPC property (cf. Definition A.28), which is required by the boundary patch partition. Some mesh elements used in the literature, e.g., hexahedrons, might not be linear which will exclude them from the templated approaches in this work. Therefore, further research should investigate, how the geometry space $\mathfrak{L}^n$ can be abstracted to include more general sets.

This work primarily focuses on the theory and the generation of templated meshes. However, there are a lot of mesh adaptation algorithms which have only been briefly discussed. A more-in-depth investigation of these algorithms would enable an increase of the mesh adaptation possibilities for templated meshes. Additionally, special cases of these algorithms for symmetric objects would decrease instance interface issues or issues with irregular instance graphs.

A combined pipeline of automatic symmetry and similarity detection and mesh generation of symmetric objects has briefly been covered in previous work [48]. A more in-depth study of a combined approach would increase the usability of templated meshes and their generation. Furthermore, future research should investigate automatic decomposition of geometries with similarities or symmetries for templated mesh generation.

Benchmark object	Dimension	Expected or estimated improvement	Best memory saving w/o SVB	Best memory saving w/ SVB	Best runtime speedup
aircraft	2D	2	2	2	**2.7**
MOSFET	2D	2	**2.2**	2.1	**2.2**
FinFET	3D	4	4.2	4	**4.4**
8-polygon	2D	8	8	7.9	8
16-polygon	2D	16	16	15.4	16.4
32-polygon	2D	32	32.2	27.7	33.2
64-polygon	2D	64	64.4	39.4	64.5
128-polygon	2D	128	128.3	36.5	117
8-TSV	3D	8	**9**	6	8.2
16-TSV	3D	16	**17.8**	9	15.8
32-TSV	3D	32	33.3	9.6	26.7
64-TSV	3D	64	54.3	6.7	35
128-TSV	3D	128	92.5	4.8	86.7
bridge	2D	28.5	29	28.4	29.2
multi-TSV	3D	1.6	**2.6**	1.5	**3.1**

Table 8.1: The best memory savings and runtime speedups for each benchmark object. n-TSV refers to a TSV with a rotational symmetry order of n. The aircraft, the MOSFET, and the FinFET are covered in Section 7.2, n-polygons and n-TSVs are covered in Section 7.3, and the bridge and multi-TSV structure are covered in Section 7.4. Savings which are at least 10% better than the expected values are colored in green, and savings which are at least 10% worse then the expected values are colored in red.

As of yet, investigations on parallelization of the algorithms involved in templated mesh generation have been conducted. In the light of stagnating single-core performance and ever-growing core-counts of modern multi-core processors, future research should cover this research gap to further improve runtime performance and to make efficient use of modern computing platforms.

Appendix A

Mathematical and Geometrical Background

Mathematical and geometrical definitions and lemmas as well as proof of lemmas important for this work are presented in this appendix. Unless explicitly stated otherwise, the topological vector space $(\mathbb{R}^n, \tau_n), n \geq 1$, where τ_n is the standard topology induced by the Euclidean norm[1], is used throughout this thesis. More specifically, this work focuses on the topological vector spaces $(\mathbb{R}^2, \tau_2)$ and $(\mathbb{R}^3, \tau_3)$.

Definition A.1 (Closure, interior, boundary)**.** For $X \subseteq \mathbb{R}^n$, the closure of X based on the topology τ_n is denoted as $\mathrm{cl}(X)$, the interior is denoted as $\mathrm{int}(X)$, and the boundary is defined as $\mathrm{bnd}(X) = \mathrm{cl}(X) \setminus \mathrm{int}(X)$

Definition A.2 (Composition of functions)**.** Let f and g be two functions. The composition of g and f is defined as:

$$(g \circ f)(x) := g(f(x)), x \in f^{-1}\left(\mathrm{img}(f) \cap \mathrm{dom}(g)\right) \tag{A.1}$$

$g \circ f$ therefore maps from the domain $\mathrm{dom}(g \circ f) = f^{-1}\left(\mathrm{img}(f) \cap \mathrm{dom}(g)\right)$ to the image $\mathrm{img}(g \circ f) = g\left(\mathrm{img}(f) \cap \mathrm{dom}(g)\right)$. The composition of two functions g and f is invalid, if $\mathrm{img}(f) \cap \mathrm{dom}(g) = \emptyset$.

Definition A.3 (n-ball, open n-ball, n-half-ball)**.** The following sets are defined for $\boldsymbol{x} \in \mathbb{R}^n$:

$$\overline{\mathcal{B}}_r^n(\boldsymbol{x}) := \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\|_2 \leq r\} \tag{A.2}$$
$$\mathcal{B}_r^n(\boldsymbol{x}) := \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\|_2 < r\} \tag{A.3}$$
$$\mathcal{H}_r^n(\boldsymbol{x}) := \{\boldsymbol{y} \in \mathbb{R}^n : \|\boldsymbol{y} - \boldsymbol{x}\|_2 \leq r \wedge y_n \geq 0\} \tag{A.4}$$

$\overline{\mathcal{B}}_r^n(\boldsymbol{x})$ is called the (closed) n-ball with center $\boldsymbol{x}$ and radius r and $\overline{\mathcal{B}}^n := \overline{\mathcal{B}}_1^n(\boldsymbol{0})$ is called the (closed) unit n-ball. $\mathcal{B}_r^n(\boldsymbol{x})$ is called the open n-ball and $\mathcal{B}^n := \mathcal{B}_1^n(\boldsymbol{0})$ is called the open unit n-ball. $\mathcal{H}_r^n(\boldsymbol{x})$ is called the (closed) n-half-ball and $\mathcal{H}^n := \mathcal{H}_1^n(\boldsymbol{0})$ is called the (closed) unit n-half-ball.

[1] Euclidean norm for $\boldsymbol{x} \in \mathbb{R}^n$: $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$

(a) Connected **(b)** Not connected

Figure A.1: Connected property of sets

For (b), choosing the blue subset to represent X_1 and the red subset to represent X_2 yields a partition of the set with $\mathrm{cl}(X_1) \cap X_2 = X_1 \cap \mathrm{cl}(X_2) = \emptyset$ (b). Therefore, (b) is not connected. For (a), no such two sets exist.

Definition A.4 (Bounded, connected, piecewise connected)**.** Let $X \subseteq \mathbb{R}^n$. X is called bounded if either X is empty or there is a point $\boldsymbol{x} \in X$ and a radius $r > 0$: $X \subseteq \overline{\mathcal{B}}_r^n(\boldsymbol{x})$. X is called connected if either X is empty or for all non-empty disjunct sets X_1, X_2 which cover X, the intersections $\mathrm{cl}(X_1) \cap X_2$ and $X_1 \cap \mathrm{cl}(X_2)$ are not empty. X is called piecewise connected, if there exists a finite number of connected sets $X_1, \ldots, X_k$, with $X = \bigcup_{i=1}^k X_i$.

Every connected set is trivially piecewise connected. A closed bounded set in $\mathbb{R}^n$ is compact. Figure A.1 visualizes the connected property.

A hyperplane is defined a linear subset of $\mathbb{R}^n$ which has a dimension of $n - 1$:

Definition A.5 (Hyperplane)**.** The set

$$\mathbb{H}_{\boldsymbol{n},d} := \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} \cdot \boldsymbol{n} = d\} \tag{A.5}$$

is called a hyperplane with normal vector $\boldsymbol{n}$ and distance to the origin d.

An important type of subsets of $\mathbb{R}^n$ are manifolds.

Definition A.6 (k-manifold, dimension of a manifold, element space, underlying space)**.** A piecewise connected, compact, non-empty set $X \subseteq \mathbb{R}^n$ is called a k-manifold if every point $\boldsymbol{x} \in X$ has a neighborhood based on the topology $\tau_n|_X$ which is homeomorph [135] to either $\mathcal{B}^k$ or $\mathcal{H}^k$. X is said to be of dimension k: $\mathrm{DIM}(X) = k$. A set containing a single point $\{\boldsymbol{x}\}, \boldsymbol{x} \in \mathbb{R}^n$ is said to be a 0-manifold.

The set of all k-manifolds of $\mathbb{R}^n$ is denoted as $\mathfrak{M}_k^n$ and $\mathfrak{M}^n := \bigcup_{k=0}^n \mathfrak{M}_k^n$. A non-empty subset of $\mathfrak{M}^n$ is called element space and a set $E \in \mathcal{E}$ is called element. The pointwise union of all elements of an element space $\mathcal{E}$ is called the underlying space of $\mathcal{E}$: $\mathrm{us}(\mathcal{E}) := \bigcup_{E \in \mathcal{E}} E$. The underlying space is the identity if $X \in \mathfrak{M}^n$: $\mathrm{us}(X) = X$.

In the literature, manifolds are usually not required to be compact and do not include their boundary. The definition in this work has been chosen to ease notation.

Definition A.7 (Cells, facets)**.** Let $\mathcal{E}$ be an element space. The cell dimension of $\mathcal{E}$ is the maximum dimension of all its elements: $\mathrm{DIM}_{\mathrm{cell}}(\mathcal{E}) := max_{E\in\mathcal{E}}\{\mathrm{DIM}(E)\}$, the facet dimension is defined as $\mathrm{DIM}_{\mathrm{facet}}(\mathcal{E}) := \mathrm{DIM}_{\mathrm{cell}}(\mathcal{E}) - 1$. An element E is called cell if $\mathrm{DIM}(E) = \mathrm{DIM}_{\mathrm{cell}}(\mathcal{E})$ and facet if $\mathrm{DIM}(E) = \mathrm{DIM}_{\mathrm{facet}}(\mathcal{E})$.

The set of all elements of a $\mathcal{E}$ with dimension k is denoted as $\mathrm{elem}_k(\mathcal{E})$, the set of all vertices is denoted as $\mathrm{vertices}(\mathcal{E}) := \mathrm{elem}_0(\mathcal{E})$, the set of all cells is denoted as $\mathrm{cells}(\mathcal{E}) := \mathrm{elem}_{\mathrm{DIM}_{\mathrm{cell}}(\mathcal{E})}(\mathcal{E})$ and the set of all facets is denoted as $\mathrm{facets}(\mathcal{E}) := \mathrm{elem}_{\mathrm{DIM}_{\mathrm{facet}}(\mathcal{E})}(\mathcal{E})$.

The standard definition of the interior and the boundary is non-intuitive for k-manifolds in $\mathbb{R}^n$ with $k < n$. For example, the interior of a triangle T in $\mathbb{R}^3$ is empty and the boundary (defined as the closure without the interior) is $\mathrm{cl}(T)$. Therefore, a more intuitive definition of the interior and the boundary of manifolds is used in this work:

Definition A.8 (Relative interior, relative boundary)**.** The relative interior of a k-manifold X is defined as the set of all $\boldsymbol{x} \in X$ which have a neighborhood based on the topology $\tau_n|_X$ which is homeomorph to $\mathcal{B}^k$. The relative interior of a set X is denoted as $\mathrm{int}_k^\star(X)$. The relative boundary of a manifold X is defined as: $\mathrm{bnd}_k^\star X := X \setminus \mathrm{int}_k^\star(X)$. If no k is specified, the dimension of X is used: $\mathrm{int}^\star(X) := \mathrm{int}^\star_{\mathrm{DIM}(X)}(X)$ and $\mathrm{bnd}^\star(X) := \mathrm{bnd}^\star_{\mathrm{DIM}(X)}(X)$

The boundary of a k-manifold is a $k-1$-manifold. For example, the closed unit n-ball $\overline{\mathcal{B}}^n$ is an n-manifold. The interior of $\overline{\mathcal{B}}^n$, $\mathrm{int}^\star(\overline{\mathcal{B}}^n)$, is the open unit n-ball $\mathcal{B}^n$. The boundary of $\overline{\mathcal{B}}^n$ $\mathrm{bnd}^\star(\overline{\mathcal{B}}^n)$ is the set $\{\boldsymbol{x} \in \mathbb{R}^n, \|\boldsymbol{x}\|_2 = 1\}$ which is an $n-1$-manifold. Some properties of manifolds are presented in the next lemma.

Lemma A.1 (Manifold properties)**.** *Let $X, A, B \subseteq \mathbb{R}^n$ be manifolds.*

(i) X is a k-manifold if and only if $\mathrm{int}_k^\star(X) \neq \emptyset$.

(ii) Let $A \subseteq B$. Then, $\mathrm{DIM}(A) \leq \mathrm{DIM}(B)$.

(iii) Let $A \subseteq B$. Then, for $k \geq \mathrm{DIM}(B)$, $\mathrm{int}_k^\star(A) \subseteq \mathrm{int}_k^\star(B)$.

(iv) Let A and B be manifolds. Then, for $k \geq \max(\mathrm{DIM}(A), \mathrm{DIM}(B))$, $\mathrm{int}_k^\star(A \cap B) = \mathrm{int}_k^\star(A) \cap \mathrm{int}_k^\star(B)$.

Proof.

(i) Follows trivially from the definition of a manifold and the relative interior.

(ii) Assume, $\mathrm{DIM}(A) > \mathrm{DIM}(B)$, then there is an $\boldsymbol{x} \in A$ which is also in B. $\boldsymbol{x}$ has a neighbor U_B based on the topology $\tau_n|_B$ which is homeomorph to $\mathcal{B}^{\mathrm{DIM}(B)}$. $\boldsymbol{x}$ has also a neighborhood U_A based on the topology $\tau_n|_A$ which is homeomorph to $\mathcal{B}^{\mathrm{DIM}(A)}$. Because $A \subseteq B$, U_A is a subset of B. U_A can be scaled down to be a subset of U_B. However, there is no neighborhood U_A which is homeomorph to $\mathcal{B}^{\mathrm{DIM}(A)}$ which is a subset of U_B (which is homeomorph to $\mathbb{R}^{\mathrm{DIM}(B)}$) with $\mathrm{DIM}(A) > \mathrm{DIM}(B)$. Therefore, $\mathrm{DIM}(A)$ has to be less or equal to $\mathrm{DIM}(B)$.

(iii) For $k > \mathrm{DIM}(B)$, $\mathrm{int}_k^\star(A)$ and $\mathrm{int}_k^\star(B)$ is empty. Otherwise, for $\mathrm{DIM}(A) < k$ ($\mathrm{DIM}(A)$ can not be larger than k due to (ii)), $\mathrm{int}_k^\star(A)$ is empty.

Let $\mathrm{DIM}(A) = \mathrm{DIM}(B) = k$. For every $\boldsymbol{x} \in \mathrm{int}_k^\star(A)$ there is a neighborhood U_A of $\boldsymbol{x}$ based on the topology $\tau_n|_A$ which is homeomorph to $\mathcal{B}^k$. Because $A \subseteq B$, $\boldsymbol{x}$ is also in B. Every neighborhood of $\boldsymbol{x}$ based on the topology $\tau_n|_B$ is either homeomorph to $\mathcal{B}^k$ or $\mathcal{H}^k$. Because U_A is also a subset of B, every neighborhood of $\boldsymbol{x}$ based on the topology $\tau_n|_B$ has to be homeomorph to $\mathcal{B}^k$. Therefore, $\boldsymbol{x}$ is in $\mathrm{int}_k^\star(B)$.

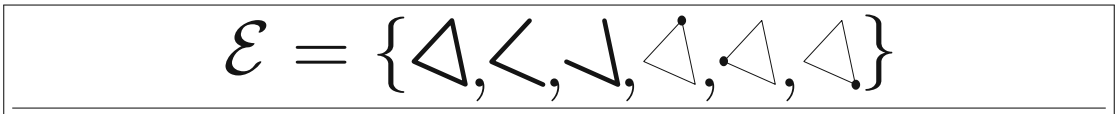


Figure A.2: Element faces are not *minimal*

The element space $\mathcal{E}$ represents a triangle with two polylines boundary elements and three vertices. The boundary of the triangle can be represented as the union of the two polylines. However, the line connecting the lower two vertices occurs twice in that union.

(iv) If $\mathrm{DIM}(A) < k$ or $\mathrm{DIM}(B) < k$, $\mathrm{int}_k^\star(A \cap B)$ and $\mathrm{int}_k^\star(A) \cap \mathrm{int}_k^\star(B)$ are empty.

Let $\mathrm{DIM}(A) = \mathrm{DIM}(B) = k$. For $\boldsymbol{x} \in \mathrm{int}_k^\star(A) \cap \mathrm{int}_k^\star(B)$, $\boldsymbol{x}$ is in $\mathrm{int}_k^\star(A)$ and has a neighborhood U_A based on the topology $\tau_n|_A$ which is homeomorph to $\mathcal{B}^k$. Also, $\boldsymbol{x}$ is in $\mathrm{int}_k^\star(B)$ and has a neighborhood U_B based on the topology $\tau_n|_B$ which is homeomorph to $\mathcal{B}^k$. $U_A \cap U_B$ is a neighborhood based on the topology $\tau_n|_{A\cap B}$ of $\boldsymbol{x}$ which is homeomorph to $\mathcal{B}^k$. Therefore, $\mathrm{int}_k^\star(A) \cap \mathrm{int}_k^\star(B)$ is a subset $\mathrm{int}_k^\star(A \cap B)$.

On the other hand, every $\boldsymbol{x} \in \mathrm{int}_k^\star(A \cap B)$ has a neighborhood $U_{A\cap B}$ based on the topology $\tau_n|_{A\cap B}$ which is homeomorph to $\mathcal{B}^k$. Every neighborhood U_A of $\boldsymbol{x}$ based on the topology $\tau_n|_A$ is either homeomorph to $\mathcal{B}^k$ or $\mathcal{H}^k$. The neighborhood $U_{A\cap B}$ can be scaled down to be a subset of U_A. Therefore, U_A has to be homeomorph to $\mathcal{B}^k$ leading to $\boldsymbol{x}$ also being in $\mathrm{int}_k^\star(A)$. A similar argument can be applied to the set B. Therefore, $\boldsymbol{x}$ is in $\mathrm{int}_k^\star(A)$ and in $\mathrm{int}_k^\star(B)$ and consequently $\mathrm{int}_k^\star(A \cap B)$ is a subset of $\mathrm{int}_k^\star(A) \cap \mathrm{int}_k^\star(B)$.

□

The boundary of manifolds used in this work can be represented as union of other manifolds with lower dimensions. For example, the boundary of a triangle can be represented as the union of its three lines. Boundary elements which are *maximal*, i.e. there are no larger elements which are included in the boundary, are called facets. The facets and recursively all facets of facets of a manifold are called faces.

Definition A.9 (Facet of a set, face of a set)**.** Let $\mathcal{E}$ be an element space. For $E \in \mathcal{E}$, a manifold $F \neq \emptyset \in \mathcal{E}$ is called a facet of E if $F \subseteq \mathrm{bnd}^\star(E)$ and there is no $X \in \mathcal{E} \setminus \{F, E\}$ for which $X \subseteq \mathrm{bnd}^\star(E)$ and $F \subseteq X$. The set of all facets of $E \in \mathcal{E}$ is denoted as $\mathrm{facets}_\mathcal{E}(E)$. The set of all faces of an element $E \in \mathcal{E}$ is defined as follows:

$$\mathrm{faces}_\mathcal{E}(E) := \{E\} \cup \bigcup_{F \in \mathrm{facets}_\mathcal{E}(E)} \mathrm{faces}_\mathcal{E}(F) \tag{A.6}$$

$$\mathrm{faces}_{\mathcal{E},k}(E) := \{f \in \mathrm{faces}_\mathcal{E}(E) | \mathrm{DIM}(f) = k\} \tag{A.7}$$

The $\mathrm{DIM}(E)$-dimensional faces of an element E is E itself ($\mathrm{faces}_{\mathrm{DIM}(E)}(E) = \{E\}$) and the $\mathrm{DIM}(E) - 1$-dimensional faces are the facets ($\mathrm{faces}_{\mathrm{DIM}(E)-1}(E) = \mathrm{facets}(E)$).

Intuitively, the facets of an element E should be a *minimal* cover of its boundary $\mathrm{bnd}^\star(E)$. Figure A.2 visualizes an element space where the boundary representation of a cell is not *minimal*. Therefore, the term *face-complete* is introduced:

Definition A.10 (Face-complete, face-completion)**.** An element space $\mathcal{E}$ is called face-complete, if for every element $E \in \mathcal{E}$ the union of all facets of E is equal to the relative boundary of E and the intersection of the relative interior of two different facets of E is empty.

Let $X \subseteq \mathcal{E}$. The face-completion of X in $\mathcal{E}$ is defined as the intersection of all subsets of $\mathcal{E}$ which are face-complete and supersets of X:

$$\mathrm{fc}_{\mathcal{E}}(X) := \bigcap_{X \subseteq Y \subset \mathcal{E}, Y \text{ is face-complete}} Y \tag{A.8}$$

If no reference element space is specified, the mesh element space is used:

$$\mathrm{fc}(X) := \mathrm{fc}_{\mathfrak{E}}(X) \tag{A.9}$$

Similar to the faces and facets, the co-faces and co-facets of an element E are defined as the elements of which E is a face or a facet, respectively.

Definition A.11 (Co-face, co-facet)**.** The co-faces of an element E of an element space $\mathcal{E}$ are all elements for which E is a face:

$$\mathrm{cofaces}_{\mathcal{E}}(E) := \{F \in \mathcal{E} | E \in \mathrm{faces}_{\mathcal{E}}(F)\} \tag{A.10}$$

$$\mathrm{cofaces}_{\mathcal{E},k}(E) := \{F \in \mathrm{cofaces}_{\mathcal{E}}(E) | \mathrm{DIM}(F) = k\} \tag{A.11}$$

The co-facets of an element E are the co-faces with one dimension larger than the element E:

$$\mathrm{cofacets}_{\mathcal{E}}(E) := \mathrm{cofaces}_{\mathcal{E},\mathrm{DIM}(E)+1} \tag{A.12}$$

Elements which share a common face are called neighbors.

Definition A.12 (Neighbor element)**.** The neighbor elements of an element $E \in \mathcal{E}$ of an element space $\mathcal{E}$ are all elements which have a common face with E:

$$\mathrm{neighbors}_{\mathcal{E}}(E) = \{N \in \mathcal{E} | \mathrm{faces}_{\mathcal{E}}(E) \cap \mathrm{faces}_{\mathcal{E}}(N) \neq \emptyset\} \tag{A.13}$$

$$\mathrm{neighbors}_{\mathcal{E},j,k}(E) = \{N \in \mathcal{E} | \mathrm{DIM}(N) = k \wedge \mathrm{faces}_{\mathcal{E},j}(E) \cap \mathrm{faces}_{\mathcal{E},j}(N) \neq \emptyset\} \tag{A.14}$$

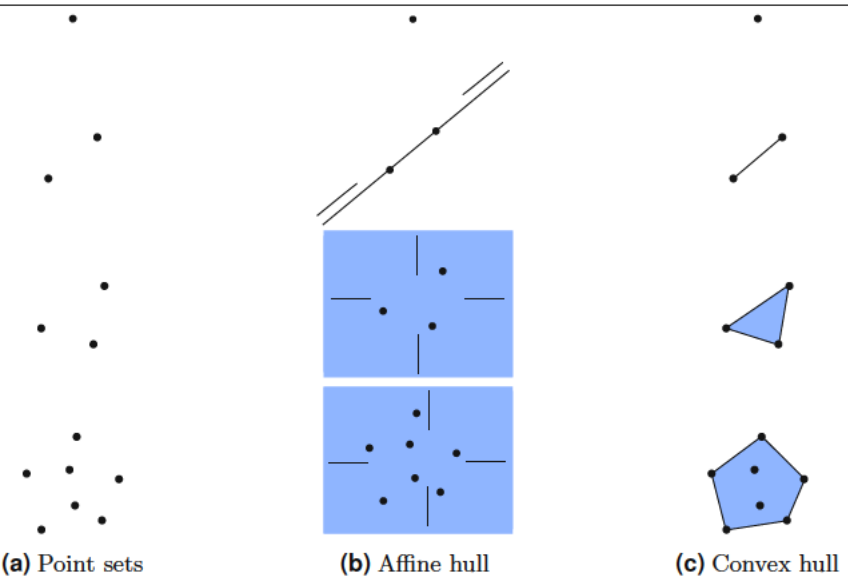
A very important property, especially for discretization-based simulation methods, is *conformity*. The intersection of two different elements of a conforming element space is a face of both elements. This property enables straight-forward interfaces between two neighboring cells which simplifies interaction between them.

Definition A.13 (Conforming)**.** An element space $\mathcal{E}$ is said to be conforming if for every two sets $E_1, E_2 \in \mathcal{E}$, their intersection $E_1 \cap E_2$ is either empty or a face of both.

Conforming element spaces play a central role in this work.

Definition A.14 (Element complex)**.** A conforming element space is called element complex.

Connected compact k-manifolds are the basic building blocks of meshes, called mesh elements. *Linear* mesh elements are the most basic elements used in this work. They are defined using affine and convex hulls.



(a) Point sets **(b)** Affine hull **(c)** Convex hull

Figure A.3: Affine and convex hull

(a) visualizes different sets of points. The affine hull and the convex hull of the corresponding point set is given in (b) and (c), respectively.

Definition A.15 (Affine, affine hull, k-flat)**.** An affine combination of a set of points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} \subseteq \mathbb{R}^n$ is defined as the linear combination $\sum_{i=1}^{k} w_i \boldsymbol{x}_i$ with $w_i \in \mathbb{R}$ and $\sum_{i=1}^{k} w_i = 1$. A point p is said to be affinely dependent on X, if there are weights w_i such that p can be written as an affine combination of X. If there are no such weights, p is called affinely independent of X. A set X is said to be affinely independent if every point $\boldsymbol{x} \in X$ is affinely independent of $X \setminus \{\boldsymbol{x}\}$. The affine hull $\mathrm{aff}(X)$ of a set X is defined as the set of all affine combinations of X. The affine hull of set X of $k+1$ affinely independent points is called a k-flat.

Definition A.16 (Convex, convex hull, co-convex)**.** A set $X \subseteq \mathbb{R}^n$ is said to be convex if for every two points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in X$ and for every $0 < \lambda < 1$, the point $\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2$ also is in X. The convex hull of a (not necessarily finite) set X is defined as: $\mathrm{conv}(X) := \bigcap_{X \subseteq K, K \text{ is convex}} K$.

An affine combination with all weights $w_i \geq 0$ is called a convex combination. The convex hull of a set is equal to all convex combinations of points from that set. Figure A.3 shows examples of the affine and convex hull of different sets of points.

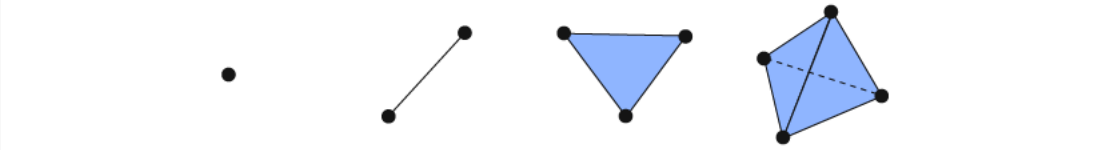


Figure A.4: Simplices

From left to right: a 0-simplex (vertex), a 1-simplex (line, edge), a 2-simplex (triangle), and a 3-simplex (tetrahedron).

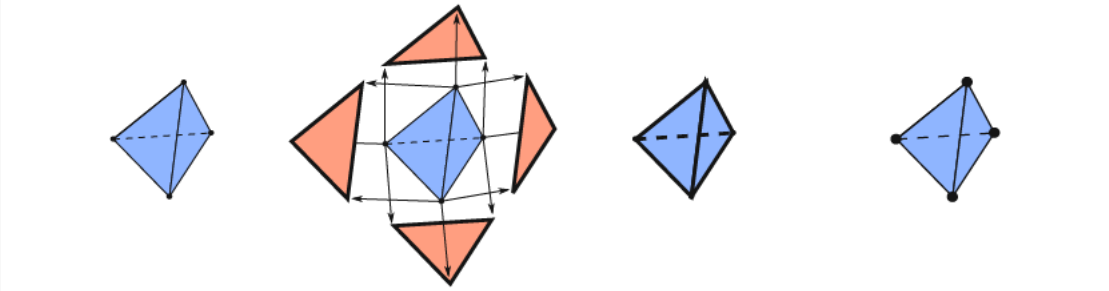


Figure A.5: Faces

From left to right: a tetrahedron, its 2D faces (four triangles), its 1D faces (six edges), and its 0D faces (four vertices).

The most basic *linear* mesh elements are called simplices.

Definition A.17 (k-simplex)**.** Given an affinely independent set $X = \{x_1, \ldots, x_{k+1}\}$, the set

$$\text{simplex}(X) = \text{conv}(X) \tag{A.15}$$

is called a k-simplex. The set of all k-simplices in $\mathbb{R}^n$ is denoted as ${}_{\widetilde{\text{simplices}}}\mathcal{E}^n_k$, the set of all j-simplices, $j \leq k$ is denoted as ${}_{\widetilde{\text{simplices}}}\mathcal{E}^n_k := \bigcup_{k=0}^{n} {}_{\widetilde{\text{simplices}}}\mathcal{E}^n_k$, and the set of all k-simplices, $k \leq n$ is denoted as ${}_{\text{simplices}}\mathcal{E}^n := {}_{\text{simplices}}\mathcal{E}^n_n$.

A 0-simplex is called a vertex, a 1-simplex is called a line or edge, a 2-simplex is called a triangle, and a 3-simplex is called a tetrahedron. Simplices are visualized in Figure A.4. ${}_{\widetilde{\text{simplices}}}\mathcal{E}^n_k$ is empty for $k > n$ because there are there are no affinely independent sets of k points in $\mathbb{R}^n$.

For every vertex $V \in_{\text{simplices}} \mathcal{E}^n$, its facet set and therefore its face set is empty. For every other k-simplex S being the convex hull of $\{s_1, \ldots, s_{k+1}\}$, all facets of S are simplices using a $k-1$-subset:

$$\text{facets}_{{}_{\text{simplices}}\mathcal{E}^n}(S) = \{\text{simplex}(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{k+1}, 1 \leq i \leq k)\} \tag{A.16}$$

Figure A.5 visualizes a tetrahedron with all of its faces.

Non-trivial *linear* elements are called polyhedrons:

Definition A.18 (k-polyhedron, facet of a polyhedron, linear k-cell, linear set, polyhedron space)**.** The convex hull of a set $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j\} \subseteq \mathbb{R}^n$ with $\mathrm{aff}(X)$ being a k-flat is called a convex k-polyhedron. For a convex k-polyhedron P, a convex $k-1$ polyhedron $F \neq \emptyset$ is called a facet if $F \subseteq \mathrm{bnd}^\star(P)$ and there is no $X \neq F$ which is a $k-1$-polyhedron and $F \subseteq X$. The set of all facets of P is denoted as $\mathrm{facets}(P)$. The finite union of convex k-polyhedra $P_1, \ldots, P_j$, which lie in a common k-flat, is called a k-polyhedron or linear k-cell. A set $A \subset \mathbb{R}^n$ is called linear if it is a polyhedron.

The set of all k-polyhedra in $\mathbb{R}^n$ is denoted as $\widetilde{\mathrm{poly}}\mathcal{E}_k^n$, the set of all j-polyhedra, $j \leq k$ is denoted as ${}_{\mathrm{poly}}\mathcal{E}_k^n := \bigcup_{k=0}^n \widetilde{\mathrm{poly}}\mathcal{E}_k^n$, and the set of all k-polyhedra, $k \leq n$ is called polyhedron space and denoted as ${}_{\mathrm{poly}}\mathcal{E}^n :=_{\mathrm{poly}} \mathcal{E}_n^n$.

Convex k-polyhedra and linear k-cells are compact k-manifolds. A linear 0-cell is a vertex, a linear 1-cell is a line, a linear 2-cell is a called a polygon, and a linear 3-cell is called a (not necessarily convex) polyhedron.

Although this work mainly focuses on simplices, some algorithms and proofs also hold for a larger class of elements. A hypercube-motivated approach is used to define more elements, especially quadrilaterals and hexahedrons.

Definition A.19 (Parameterized element)**.** Given a continuous parameterization $f(\alpha_1, \ldots, \alpha_k) : [0,1]^k \to \mathbb{R}^n$, the set

$$\mathcal{P}(f) := \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x} = f(\alpha_1, \ldots, \alpha_k), 0 \leq \alpha_1, \ldots, \alpha_k \leq 1\} \tag{A.17}$$

is called a parameterized element if the following restrictions are met:

(i) $\mathcal{P}(f)$ is a manifold.

(ii) $\mathcal{P}(f)$ is not degenerated, meaning that for all parameters $0 < \alpha_i < 1$, $f(\alpha_1, \ldots, \alpha_k)$ is in $\mathrm{int}^\star(\mathcal{P}(f))$: $\forall 0 < \alpha_1, \ldots, \alpha_k < 1 : f(\alpha_1, \ldots, \alpha_k) \in \mathrm{int}^\star(\mathcal{P}(f))$

(iii) The boundary of the parameter space $\mathrm{bnd}^\star([0,1]^k)$ maps to the boundary of the element $\mathrm{bnd}^\star(\mathcal{P}(f))$: $f(\mathrm{bnd}^\star([0,1]^k)) = \mathrm{bnd}^\star(\mathcal{P}(f))$.

To exclude exotic subsets of $\mathbb{R}^n$, only parameterizations which lead to manifolds are allowed (restriction (i)). Restriction (ii) prevents the parameterized element from being degenerated and *self-intersecting*. The third restriction (iii) ensures, that the parameterized element is bounded and closed and therefore compact. Additionally, (iii) gives a parameterization for the boundary of the parameterized element.

Definition A.20 (k-interpolation combination)**.** Given an ordered tuple $T = (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{2^k}), \boldsymbol{t}_i \in \mathbb{R}^n$, the function

$$I_{(\boldsymbol{t}_1,\ldots,\boldsymbol{t}_{2^k})}(\alpha_1, \ldots, \alpha_k) := \begin{matrix}(1 - \alpha_k) & I_{(\boldsymbol{t}_1,\ldots,\boldsymbol{t}_{2^{k-1}})}(\alpha_1, \ldots, \alpha_{k-1}) \\ \alpha_k & I_{(\boldsymbol{t}_{2^{k-1}+1},\ldots,\boldsymbol{t}_{2^k})}(\alpha_1, \ldots, \alpha_{k-1})\end{matrix} + \tag{A.18}$$

with

$$I_{(\boldsymbol{t}_1,\boldsymbol{t}_2)}(\alpha) := (1-\alpha)\boldsymbol{t}_1 + \alpha\boldsymbol{t}_2 \tag{A.19}$$

is called a k-interpolation combination.

The definition of the k-interpolation combination is motivated by the recursive parameterization of a hypercube. Note, that a k-interpolation combination is not a linear function in general.

Therefore, when using a parameterized element based on a k-interpolation combination, the element does not need to be linear. However, for fixed $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_{2^k}$ the k-interpolation combination is linear in α_i.

When using pairwise different points, the parameterized element using a 2-interpolation combination results in a quadrilateral and using a 3-interpolation combination results in a hexahedron. The k-interpolation combination is injective for these element types.

Definition A.21 (Quadrilateral)**.** Given an ordered tuple $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_4)\,, \boldsymbol{q}_i \in \mathbb{R}^n, \boldsymbol{q}_i \neq \boldsymbol{q}_j$ for $i \neq j$, the set

$$\operatorname{quad}(Q) := \mathcal{P}(I_Q) \tag{A.20}$$

is called a quadrilateral. For $n \geq 2$, $\widetilde{\operatorname{quad}}\mathcal{E}^n$ is defined as the set of all quadrilaterals in $\mathbb{R}^n$, $\widetilde{\operatorname{quad}}\mathcal{E}^n = \emptyset$ otherwise. The recursive set of all quads is denoted as ${}_{\operatorname{quad}}\mathcal{E}^n := \widetilde{\operatorname{quad}}\,\mathcal{E}^n \cup_{\text{simplices}} \mathcal{E}_1^n$.

Quadrilaterals in $\mathbb{R}^2$ are convex and their boundary is piecewise linear. The facets of a quadrilateral are four lines:

$$\operatorname{facets}_{{}_{\operatorname{quad}}\mathcal{E}^n}(\operatorname{quad}((\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \boldsymbol{q}_4))) = \left\{ \begin{array}{l} \operatorname{simplex}(\{\boldsymbol{q}_1, \boldsymbol{q}_2\}), \operatorname{simplex}(\{\boldsymbol{q}_1, \boldsymbol{q}_3\}), \\ \operatorname{simplex}(\{\boldsymbol{q}_2, \boldsymbol{q}_4\}), \operatorname{simplex}(\{\boldsymbol{q}_3, \boldsymbol{q}_4\}) \end{array} \right\} \tag{A.21}$$

Definition A.22 (Hexahedron)**.** Given an ordered tuple $H = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_8)\,, \boldsymbol{h}_i \in \mathbb{R}^n, \boldsymbol{h}_i \neq \boldsymbol{h}_j$ for $i \neq j$, the set

$$\operatorname{hex}(H) := \mathcal{P}(I_H) \tag{A.22}$$

is called a hexahedron. For $n \geq 3$, $\widetilde{\operatorname{hex}}\mathcal{E}^n$ is defined as the set of all hexahedrons in $\mathbb{R}^n$, $\widetilde{\operatorname{hex}}\mathcal{E}^n = \emptyset$ otherwise. The recursive set of all hexahedrons is denoted as ${}_{\operatorname{hex}}\mathcal{E}^n := \widetilde{\operatorname{hex}}\,\mathcal{E}^n \cup_{\operatorname{quad}} \mathcal{E}^n$.

The facets of a hexahedron are six quadrilaterals:

$$\operatorname{facets}_{{}_{\operatorname{hex}}\mathcal{E}^n}(\operatorname{hex}((\boldsymbol{h}_1, \ldots, \boldsymbol{h}_8))) = \left\{ \begin{array}{l} \operatorname{quad}((\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4)), \operatorname{quad}((\boldsymbol{h}_5, \boldsymbol{h}_7, \boldsymbol{h}_6, \boldsymbol{h}_8)), \\ \operatorname{quad}((\boldsymbol{h}_1, \boldsymbol{h}_3, \boldsymbol{h}_5, \boldsymbol{h}_7)), \operatorname{quad}((\boldsymbol{h}_2, \boldsymbol{h}_6, \boldsymbol{h}_4, \boldsymbol{h}_8)), \\ \operatorname{quad}((\boldsymbol{h}_1, \boldsymbol{h}_5, \boldsymbol{h}_2, \boldsymbol{h}_6)), \operatorname{quad}((\boldsymbol{h}_3, \boldsymbol{h}_4, \boldsymbol{h}_7, \boldsymbol{h}_8)) \end{array} \right\} \tag{A.23}$$

While mixed element spaces with triangle and quadrilateral cell elements can be conforming, more element types are required for 3D element spaces because tetrahedrons and hexahedrons do not share any facet type. Mixed element spaces are not a focus of this work, however for the sake of completeness two additional element types are defined using a non-injective k-interpolation combination. A 3-interpolation combination where the last four points are all the same results in a pyramid. A 3-interpolation combination, where the 3rd and 4th as well as the 7th and the 8th point are equal results in a triangular prism, also called wedge.

Definition A.23 (Pyramid)**.** Given an ordered tuple $P = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_5)\,, \boldsymbol{p}_i \in \mathbb{R}^n, \boldsymbol{p}_i \neq \boldsymbol{p}_j$ for $i \neq j$, the set

$$\operatorname{pyramid}(P) := \mathcal{P}(I_{(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_5, \boldsymbol{p}_5, \boldsymbol{p}_5)}) \tag{A.24}$$

is called a pyramid. For $n \geq 3$, $\widetilde{\operatorname{pyramid}}\mathcal{E}^n$ is defined as the set of all pyramids in $\mathbb{R}^n$, $\widetilde{\operatorname{pyramid}}\mathcal{E}^n = \emptyset$ otherwise. The recursive set of all pyramids is denoted as ${}_{\operatorname{pyramid}}\mathcal{E}^n := \widetilde{\operatorname{pyramid}}\,\mathcal{E}^n \cup_{\operatorname{quad}} \mathcal{E}^n \cup_{\text{simplices}} \mathcal{E}_2^n$.

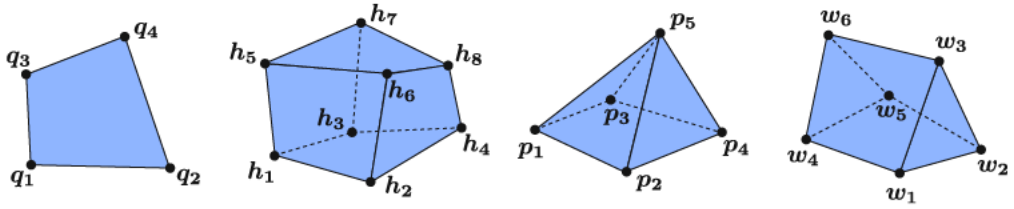


Figure A.6: Non-simplex elements

From left to right: a quadrilateral, a hexahedron, a pyramid, and a wedge.

The facets of a pyramid are four triangles and one quadrilateral:

$$\text{facets}_{\text{pyramid}\mathcal{E}^n}(\text{pyramid}((p_1,\ldots,p_5))) = \left\{ \begin{array}{l} \text{simplex}(\{p_1,p_2,p_5\}), \text{simplex}(\{p_1,p_3,p_5\}), \\ \text{simplex}(\{p_2,p_4,p_5\}), \text{simplex}(\{p_3,p_4,p_5\}), \\ \text{quad}((p_1,p_2,p_3,p_4)) \end{array} \right\} \tag{A.25}$$

Definition A.24 (Wedge)**.** Given an ordered tuple $W = (w_1,\ldots,w_6), w_i \in \mathbb{R}^n, w_i \neq w_j$ for $i \neq j$, the set

$$\text{wedge}(W) := \mathcal{P}(I_{(w_1,w_2,w_3,w_3,w_4,w_5,w_6,w_6)}) \tag{A.26}$$

is called a wedge. For $n \geq 3$, $\widetilde{\text{wedge}}\mathcal{E}^n$ is defined as the set of all wedges in $\mathbb{R}^n$, $\widetilde{\text{wedge}}\mathcal{E}^n = \emptyset$ otherwise. The recursive set of all wedges is denoted as ${}_{\text{wedge}}\mathcal{E}^n := \widetilde{\text{wedge}}\,\mathcal{E}^n \cup_{\text{quad}} \mathcal{E}^n \cup_{\text{simplices}} \mathcal{E}^n_2$.

The facets of a wedge are two triangles and three quadrilaterals:

$$\text{facets}_{\text{wedge}\mathcal{E}^n}(\text{wedge}((w_1,\ldots,w_6))) = \left\{ \begin{array}{l} \text{simplex}(\{w_1,w_2,w_3\}), \text{simplex}(\{w_4,w_5,w_6\}), \\ \text{quad}((w_1,w_3,w_4,w_6)), \text{quad}((w_3,w_2,w_6,w_5)), \\ \text{quad}((w_2,w_1,w_5,w_4)) \end{array} \right\} \tag{A.27}$$

Examples of quadrilaterals, hexahedrons, pyramids, and wedges are shown in Figure A.6.

Simplices can also be represented using parameterized elements: $\mathcal{P}(I_{(l_1,l_2)})$ is a line, $\mathcal{P}(I_{(t_1,t_2,t_3,t_3)})$ is a triangle, and $\mathcal{P}(I_{(e_1,e_2,e_3,e_3,e_4,e_4,e_4,e_4)})$ is a tetrahedron. All elements presented here are connected compact manifolds. A vertex is a 0-manifold, a line is a 1-manifold, triangles and quadrilaterals are 2-manifolds, and tetrahedrons, hexahedrons, pyramids, and wedges are 3-manifolds.

The presented mesh elements are combined in the mesh element space which is defined as follows:

Definition A.25 (Mesh element space, geometry space)**.** The set of all linear mesh elements is called mesh element space and is defined as:

$$\mathfrak{E}^n = \{A \in_{\text{simplices}} \mathcal{E}^n \cup_{\text{quad}} \mathcal{E}^n \cup_{\text{hex}} \mathcal{E}^n \cup_{\text{pyramid}} \mathcal{E}^n \cup_{\text{wedge}} \mathcal{E}^n | A \text{ is linear}\} \tag{A.28}$$

The geometry space is defined as the set of all finite unions of mesh elements which are also manifolds:

$$\mathfrak{L}^n := \left\{ A \in \mathfrak{M}^n | A = \bigcup_{i=1}^{k} A_i, A_i \in \mathfrak{E}^n \right\} \tag{A.29}$$

Elements of $\mathfrak{L}^n$ are called geometries. If no reference element space for the facet set, the face set, the co-facet set, the co-face set, and the neighbor set is specified, the mesh element space is used. Let $E \in \mathfrak{E}^n$ be a mesh element.

$$\begin{aligned}
\text{facets}(E) &:= \text{facets}_{\mathfrak{E}^n}(E) && \text{(A.30)}\\
\text{faces}(E) &:= \text{faces}_{\mathfrak{E}^n}(E) && \text{(A.31)}\\
\text{faces}_k(E) &:= \text{faces}_{\mathfrak{E}^n,k}(E) && \text{(A.32)}\\
\text{cofacets}(E) &:= \text{cofacets}_{\mathfrak{E}^n}(E) && \text{(A.33)}\\
\text{cofaces}(E) &:= \text{cofaces}_{\mathfrak{E}^n}(E) && \text{(A.34)}\\
\text{cofaces}_k(E) &:= \text{cofaces}_{\mathfrak{E}^n,k}(E) && \text{(A.35)}\\
\text{neighbors}(E) &:= \text{neighbors}_{\mathfrak{E}^n}(E) && \text{(A.36)}\\
\text{neighbors}_{j,k}(E) &:= \text{neighbors}_{\mathfrak{E}^n,j,k}(E) && \text{(A.37)}
\end{aligned}$$

The geometry space $\mathfrak{L}^n$ is equal to the set of all polyhedra which are manifolds: $\mathfrak{L}^n = \mathfrak{M}^n \cap_{\text{poly}} \mathcal{E}^n$. The element spaces ${}_{\text{simplices}}\mathcal{E}^n_k$, ${}_{\text{quad}}\mathcal{E}^n$, ${}_{\text{hex}}\mathcal{E}^n$, ${}_{\text{pyramid}}\mathcal{E}^n$, ${}_{\text{wedge}}\mathcal{E}^n$, and $\mathfrak{E}^n$ are face-complete for $k \geq 0, n \geq 1$.

Definition A.26 (Covering, partition)**.** Let $X \subseteq \mathbb{R}^n$ and $P = \{P_1, \dots P_m\}$ with P_i being non-empty subsets of $\mathbb{R}^n$. P is called a covering of X, if $X = \bigcup_{i=1}^{k} P_i$. If, additionally, all P_i are pairwise disjunct, P is called a partition of X.

However, it is generally not possible to have a non-trivial partition of a closed set consisting of closed sets. Therefore, a slight abstraction of a partition is defined.

Definition A.27 (Manifold partition)**.** Let $X \in \mathfrak{M}^n$ with $\text{DIM}(X) = k$ and $S = \{S_1, \dots, S_m\}$ be a covering of X with $S_i \in \mathfrak{M}^n$. S is called a manifold partition of X if the intersection $\text{int}^\star_k(S_i) \cap \text{int}^\star_k(S_j)$ is empty for all $S_i \neq S_j$.

A manifold partition of a k-manifold allows for an arbitrary number of – not necessarily disjunct – manifolds which have a dimension less than k. For example, the closed unit n-ball can have a valid manifold partition which consists of the closed unit n-ball itself and its boundary. Using this definition, it is possible to create a manifold partition of a k-manifold X using manifolds.

The choice of $\mathfrak{L}^n$ and $\mathfrak{E}^n$ only containing linear elements is motivated by the requirements for the boundary patch partition (cf. Section 4.3), which requires the intersection of sets. This intersection should be representable by mesh elements or geometries. However, the intersection of two manifolds is generally not a manifold. This is even true for simple manifolds as visualized in Figure A.7. Instead of representing the intersection of sets with a single mesh element or geometry, the approach used in this work is to represent the intersection as a union of mesh elements or geometries.

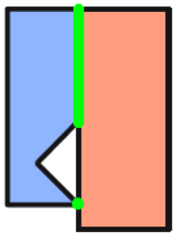


Figure A.7: The intersection of manifolds is not a manifold

The intersection of the two manifolds colored in blue and red is highlighted in green. This intersection is not a manifold.

Definition A.28 (Intersection-partitionable, intersection-partitionable-complete (IPC)). Let $\mathcal{E}$ be an element space and $A, B \in \mathcal{E}$ be two elements. A and B are said to be intersection-partitionable (in $\mathcal{E}$), if there is a manifold partition $C = \{C_1, \dots, C_k\}$ of $A \cap B$ with $C_i \in \mathcal{E}$. An element space $\mathcal{E}$ is called intersection-partitionable-complete, if all sets $A, B \in \mathcal{E}$ are intersection-partitionable.

In general, the intersection partition is not unique. Due to the definition of conformity, every element complex is IPC.

Lemma A.2 ($\mathfrak{M}^n$ is not IPC). *The set of all manifolds $\mathfrak{M}^n$ is not IPC.*

Proof. Let $C, X \subseteq \mathbb{R}^{n-1}$ be two non-empty compact sets with $C \subseteq X$ and $f(\boldsymbol{x}) = \operatorname{dist}(\boldsymbol{x}, C)$. Then, f is continuous and $f(\boldsymbol{x}) = 0$ if and only if $\boldsymbol{x} \in C$. Let $A \subseteq X \times [0, \operatorname{diam}(X)]$ be the graph of f and $B = X \times \{0\}$. A and B are n-manifolds and their intersection is C.

Choosing $X = [0,1]^{n-1}$ and $C = \{1/n | n \in \mathbb{N}\} \times [0,1]^{n-2}$ yields A and B being n-manifolds. However, there is no finite manifold partition of $A \cap B = C$. □

The geometry space $\mathfrak{L}^n$, however, is IPC.

Lemma A.3 ($\mathfrak{L}^n$ is IPC). *The geometry space $\mathfrak{L}^n$ is IPC.*

Proof. For $A, B \in \mathfrak{L}^n$, A and B can be represented as a finite union of simplices: $A = \bigcup_{i=1}^{k} A_i$, $B = \bigcup_{j=1}^{m} B_j$. The intersection $A \cap B$ can be reformulated in the following way:

$$A \cap B = \left(\bigcup_{i=1}^{k} A_i \right) \cap \left(\bigcup_{j=1}^{m} B_j \right) = \bigcup_{i=1}^{k} \bigcup_{j=1}^{m} A_i \cap B_j \tag{A.38}$$

The intersection of the simplices A_i and B_i can be represented as a finite union of simplices. Therefore, $A \cap B$ can be represented as a finite union of simplices and $\mathfrak{L}^n$ is IPC. □

The partition of two sets used for the IPC property is called the intersection partition.

Definition A.29 (Intersection partition). For two sets $A, B \in \mathfrak{L}^n$, having the representation $A = \bigcup_{i=1}^{k} A_i$ and $B = \bigcup_{j=1}^{m} B_j$, with $A_i \in \mathfrak{E}^n$ and $B_j \in \mathfrak{E}^n$, the intersection partition of A and B is defined as:

$$\operatorname{ip}(A, B) := \{A_i \cap B_j | i = 1, \dots k, j = 1, \dots, m, A_i \cap B_j \neq \emptyset\} \tag{A.39}$$

Every element C of an intersection partition of two sets X and Y is naturally a subset of the intersection of $X \cap Y$. Additionally, according to Lemma A.1, the dimension of every element C_i of the intersection partition of A and B is at most $\min(\operatorname{DIM}(A), \operatorname{DIM}(B))$.

Two manifold partitions of the same set X can be combined to create a *finer* manifold partition. The intuitive approach for a refinement of two manifold partitions would look like this:

$$S_1 \cap S_2 \stackrel{?}{:=} \{A \cap B | A \in S_1, B \in S_2\} \tag{A.40}$$

However, as mentioned above, the intersection of two manifolds is not a manifold, but for $A, B \in \mathfrak{L}^n$ the intersection partition can be used. The refinement of two manifold partitions is defined as follows:

Definition A.30 (Partition refinement)**.** Let $X \in \mathfrak{L}^n$ and S, P be coverings of X. The refinement of S and P is defined as:

$$\text{refine}(S, P) := \bigcup_{A \in S, B \in P} \text{ip}(A, B) \tag{A.41}$$

Lemma A.4 (Refinement of manifold partition is manifold partition)**.** *The refinement of two manifold partitions S, P of a set $X \in \mathfrak{L}^n$ is a manifold partition of X.*

Proof.

(i) To prove: $|\text{refine}(S, P)| < \infty$:
The finite combination of finite sets is again finite.

(ii) To prove: $\emptyset \notin \text{refine}(S, P)$:
$\emptyset$ is not an element of $\text{refine}(S, P)$ due to the definition of intersection partition.

(iii) To prove: $\text{refine}(S, P)$ is a covering of X:
At first, $X \subseteq \bigcup_{A \in \text{refine}(S,P)} A$ is shown, followed by $X \supseteq \bigcup_{A \in \text{refine}(S,P)} A$.

For all $x \in X$ there are partition elements $A \in S$ and $B \in P$ which both contain x. Therefore, x is in the intersection $A \cap B$ and there is a set C in the intersection partition of A and B which also contains x. C is an element of $\text{refine}(S, P)$. Consequently, $x \in \bigcup_{D \in \text{refine}(S,P)} D$.

On the other hand, for all $x \in \bigcup_{A \in \text{refine}(S,P)} A$, there is a set $A \in \text{refine}(S, P)$ which contains x. In turn, there are partition elements $A_S \in S$ and $A_P \in P$ for which A is in their intersection partition. Because x is in A, x is also A_S and A_P and consequently also in X.

(iv) To prove: $\text{int}_k^\star(A) \cap \text{int}_k^\star(B) = \emptyset$ for all sets $A \neq B$ of $\text{refine}(S, P)$:
Let k be the dimension of X and A and B two different elements of $\text{refine}(S, P)$. For $A \in \text{refine}(S, P)$, there are partition elements $A_S \in S$ and $A_P \in P$ for which A is in the intersection partition of A_S and A_P. Similarly, for $B \in \text{refine}(S, P)$, there are partition elements $B_S \in S$ and $B_P \in P$ for which B is in the intersection partition of B_S and B_P.

If $A_S = B_S$ and $A_P = B_P$, then $\text{int}_k^\star(A) \cap \text{int}_k^\star(B)$ is empty due to the definition of the intersection partition.

Otherwise, from Lemma A.1 and due to the assumption that $\text{DIM}(A)$ and $\text{DIM}(B)$ are less or equal to k follows, that

$$\begin{aligned} \text{int}_k^\star(A) \cap \text{int}_k^\star(B) = \text{int}_k^\star(A \cap B) \subseteq \text{int}_k^\star((S_A \cap P_A) \cap (S_B \cap P_B)) = \\ \text{int}_k^\star((S_A \cap S_B) \cap (P_A \cap P_B)) = \text{int}_k^\star((S_A \cap S_B)) \cap \text{int}_k^\star((P_A \cap P_B)) \end{aligned} \tag{A.42}$$

However, S and P are manifold partitions and $\text{int}_k^\star((S_A \cap S_B)) = \text{int}_k^\star((P_A \cap P_B)) = \emptyset$.

□

Bibliography

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, 2008. doi:10.1109/IEEESTD.2008.4610935.

[2] CGAL, July 2016. URL: http://www.cgal.org/.

[3] CLANG, July 2016. URL: http://clang.llvm.org/.

[4] FEniCS Project, July 2016. URL: https://fenicsproject.org/.

[5] Gmsh, July 2016. URL: http://gmsh.info/.

[6] GNU General Public License, July 2016. URL: http://www.gnu.org/licenses/gpl-3.0.html.

[7] MeshGems, July 2016. URL: http://www.meshgems.com/.

[8] MESQUITE, July 2016. URL: https://trilinos.org/oldsite/packages/mesquite/.

[9] Netgen Mesh Generator, July 2016. URL: http://sourceforge.net/projects/netgen-mesher/.

[10] Pointwise, July 2016. URL: http://www.pointwise.com/.

[11] Pointwise's Quality Meshes Lead to Accurate CFD, July 2016. URL: http://www.pointwise.com/pw/quality.shtml.

[12] Stellar, July 2016. URL: http://www.cs.berkeley.edu/~jrs/stellar/.

[13] Symmetry- and Similarity-Aware Volumetric Meshing Benchmarks, July 2016. URL: https://github.com/FlorianRudolf/SSAVM-Benchmark.

[14] TetGen, July 2016. URL: http://tetgen.org/.

[15] The CUBIT Geometry and Mesh Generation Toolkit, July 2016. URL: https://cubit.sandia.gov/.

[16] The MIT License, July 2016. URL: https://opensource.org/licenses/mit-license.php.

[17] Triangle, July 2016. URL: https://www.cs.cmu.edu/~quake/triangle.html.

[18] VERDICT, July 2016. URL: https://cubit.sandia.gov/public/verdict.html.

[19] ViennaMesh, July 2016. URL: http://viennamesh.sourceforge.net/.

[20] A. Berner, M. Wand, N. J. Mitra, D. Mewes, and H.-P. Seidel. Shape Analysis with Subspace Symmetries. *Computer Graphics Forum*, 30(2), pages 277–286, 2011. doi:10.1111/j.1467-8659.2011.01859.x.

[21] A. Bowyer. Computing Dirichlet Tessellations. *The Computer Journal*, 24(2), pages 162–166, 1981. doi:10.1093/comjnl/24.2.162.

[22] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. unpublished preprint, 2013. URL: https://arxiv.org/abs/1311.3144v3.

[23] A. D. Polyanin and V. E. Nazaikinskii. *Handbook of Linear Partial Differential Equations for Engineers and Scientists (Second Edition)*. Chapman and Hall/CRC Press, 2016. ISBN: 978-1-4665-8145-6.

[24] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the Design of CGAL a Computational Geometry Algorithms Library. *Software: Practice and Experience*, 30(11), pages 1167–1202, 2000. doi:10.1002/1097-024X(200009)30:11<1167::AID-SPE337>3.0.CO;2-B.

[25] A. J. Chorin, J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer-Verlag New York, 1993. ISBN: 978-0-387-97918-2.

[26] A. Martinet, C. Soler, N. Holzschuch, and F. Sillion. Accurate Detection of Symmetries in 3D Shapes. *ACM Transactions on Graphics*, 25(2), pages 439–464, 2006. doi:10.1145/1138450.1138462.

[27] A. P. Singulani, H. Ceric, and S. Selberherr. Thermo-Mechanical Simulations of an Open Tungsten TSV. In *Proceedings of the 14th IEEE Electronics Packaging Technology Conference (EPTC)*, pages 107–111, 2012. doi:10.1109/EPTC.2012.6507061.

[28] A. P. Singulani, H. Ceric, and S. Selberherr. Stress Estimation in Open Tungsten TSV. In *Proceedings of the 18th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 1–4, 2013. doi:10.1109/SISPAD.2013.6650575.

[29] A. Pellé and M. Teillaud. Periodic Meshes for the CGAL Library. In *Proceedings of the 23th International Meshing Roundtable (IMR)*, 2014. Research Note.

[30] B. Brown and S. Rusinkiewicz. Global Non-Rigid Alignment of 3-D Scans. *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2007*, 26(3), pages 21:1–21:10, 2007. doi:10.1145/1276377.1276404.

[31] B. Delaunay. Sur la Sphère Vide. A la Mémoire de Georges Voronoï. *Bulletin de l'Académie des Sciences de l'URSS*, (6), pages 793–800, 1934.

[32] B. M. Klingner and J. R. Shewchuk. Agressive Tetrahedral Mesh Improvement. In *Proceedings of the 16th International Meshing Roundtable (IMR)*, pages 3–23, 2007. doi:10.1007/978-3-540-75103-8_1.

[33] B. Raphael and I. F. C. Smith. *Engineering Informatics: Fundamentals of Computer-Aided Engineering (Second Edition)*. Wiley, 2013. ISBN: 978-1-119-95341-8.

[34] C. Carstensen. An Adaptive Mesh-Refining Algorithm Allowing for an H^1 Stable L^2 Projection onto Courant Finite Element Spaces. *Constructive Approximation*, 20(4), pages 549–564, 2004. doi:10.1007/s00365-003-0550-5.

[35] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D Finite Element Mesh Generator with Built-In Pre- and Post-Processing Facilities. *International Journal for Numerical Methods in Engineering*, 79(11), pages 1309–1331, 2009. doi:10.1002/nme.2579.

[36] C. Jamin, P. Alliez, M. Yvinec, and J.-D. Boissonnat. CGALmesh: A Generic Framework for Delaunay Mesh Generation. *ACM Transactions on Mathematical Software*, 41(4), pages 23:1–23:24, 2015. doi:10.1145/2699463.

[37] C. Kurz, X. Wu, M. Wand, T. Thormählen, P. Kohli, and H.-P. Seidel. Symmetry-Aware Template Deformation and Fitting. *Computer Graphics Forum*, 33(6), pages 205–219, 2014. doi:10.1111/cgf.12344.

[38] C. M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers Inc., 1989. ISBN: 978-1-55860-067-6.

[39] C. O. Frederick, Y. C. Wong, and F. W. Edge. Two-Dimensional Automatic Mesh Generation for Structural Analysis. *International Journal for Numerical Methods in Engineering*, 2(1), pages 133–144, 1970. doi:10.1002/nme.1620020112.

[40] C.K. Lee and S.H. Lo. A New Scheme for the Generation of a Graded Quadrilateral Mesh. *Computers & Structures*, 52(5), pages 847–857, 1994. doi:10.1016/0045-7949(94)90070-1.

[41] D. A. Field. Laplacian Smoothing and Delaunay Triangulations. *Communications in Applied Numerical Methods*, 4(6), pages 709–712, 1988. doi:10.1002/cnm.1630040603.

[42] D. F. Watson. Computing the n-Dimensional Delaunay Tessellation with Application to Voronoi Polytopes. *The Computer Journal*, 24(2), pages 167–172, 1981. doi:10.1093/comjnl/24.2.167.

[43] D. Hisamoto, W.-C. Lee, J.Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T.-J. King, J. Bokor, and C. Hu. FinFET - A Self-Aligned Double-Gate MOSFET Scalable Beyond 20nm. *IEEE Transactions on Electron Devices*, 47(12), pages 2320–2325, 2000. doi:10.1109/16.887014.

[44] D. White, L. Mingwu, S. E. Benzley, and G. D. Sjaardema. Automated Hexahedral Mesh Generation by Virtual Decomposition. In *Proceedings of the 4th International Meshing Roundtable (IMR)*, pages 165–176, 1995.

[45] D.A. May, J. Brown, and L. Le Pourhiet. A Scalable, Matrix-free Multigrid Preconditioner for Finite Element Discretizations of Heterogeneous Stokes Flow. *Computer Methods in Applied Mechanics and Engineering*, 290, pages 496–523, 2015. doi:10.1016/j.cma.2015.03.014.

[46] E. A. Nasr and A. K. Kamrani. *Computer-Based Design and Manufacturing: An Information-Based Approach*. Springer US, 2007. ISBN: 978-0-387-23323-9.

[47] E. Millan and I. Rudomin. Impostors and Pseudo-instancing for GPU Crowd Rendering. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE)*, pages 49–55, 2006. doi:10.1145/1174429.1174436.

[48] F. Rudolf, J. Weinbub, K. Rupp, A. Morhammer, and S. Selberherr. Symmetry-Aware 3D Volumetric Mesh Generation - An Analysis of Performance and Element Quality. In *Proceedings of the 24th International Meshing Roundtable (IMR)*, 2015. Research Note.

[49] F. Rudolf, J. Weinbub, K. Rupp, and S. Selberherr. The Meshing Framework ViennaMesh for Finite Element Applications. *Journal of Computational and Applied Mathematics*, 270, pages 166–177, 2014. doi:10.1016/j.cam.2014.02.005.

[50] F. Santos. Geometric Bistellar Flips. The setting, the Context and a Construction. In *Proceedings of the International Congress of Mathematicians (ICM)*, pages 931–962, 2006. doi:10.4171/022-3/46.

[51] G. Teschl. *Ordinary Differential Equations and Dynamical Systems*. American Mathematical Society, 2012. ISBN: 978-0-8218-8328-0.

[52] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2006. ISBN: 978-0-521-68207-7.

[53] H. Si. TetGen, A Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Transactions on Mathematical Software*, 41(2), pages 11:1–11:36, 2015. doi:10.1145/2629697.

[54] I. Babuška and A. K. Aziz. On the Angle Condition in the Finite Element Method. *SIAM Journal on Numerical Analysis*, 13(2), pages 214–226, 1976. doi:10.1137/0713021.

[55] I. G. Rosenberg and F. Stenger. A Lower Bound on the Angles of Triangles Constructed by Bisecting the Longest Side. *Mathematics of Computation*, 29(130), pages 390–395, 1975. doi:10.2307/2005558.

[56] J. C. Caendish, D. A. Field, and W. H. Frey. An Apporach to Automatic Three-Dimensional Finite Element Mesh Generation. *International Journal for Numerical Methods in Engineering*, 21(2), pages 329–347, 1985. doi:10.1002/nme.1620210210.

[57] J. Chaskalovic. *Finite Element Methods for Engineering Sciences*. Springer-Verlag Berlin Heidelberg, 2008. ISBN: 978-3-540-76342-0.

[58] J. D. Wolter, T. C. Woo, and R. A. Volz. Optimal Algorithms for Symmetry Detection in Two and Three Dimensions. *The Visual Computer*, 1(1), pages 37–48, 1985. doi:10.1007/BF01901268.

[59] J.-E. Wurst. *Hp-Finite Elements for PDE-Constrained Optimization*. Würzburg University Press, 2015. ISBN: 978-3-9582602-4-5.

[60] J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation*. CRC Press, 1998. ISBN: 978-0-8493-2687-5.

[61] J. Kalojanov. Efficient r-Symmetry Detection for Triangle Meshes. Technical Report 2015/01, Saarländische Universitäts- und Landesbibliothek, 2015.

[62] J.-P. Colinge and C.A. Colinge. *Physics of Semiconductor Devices*. Springer US, 2002. ISBN: 978-1-4020-7018-1.

[63] J. Podolak, P. Shilane, A. Golovinskiy, S. Rusinkiewicz, and T. Funkhouser. A Planar-reflective Symmetry Transform for 3D Shapes. *ACM Transactions on Graphics*, 25(3), pages 549–559, 2006. doi:10.1145/1141911.1141923.

[64] J. R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Selected Papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering (WACG)*, pages 203–222, 1996. doi:10.1007/BFb0014497.

[65] J. R. Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3), pages 305–363, 1997. doi:10.1007/PL00009321.

[66] J. R. Shewchuk. What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Proceedings of the 11th International Meshing Roundtable (IMR)*, pages 115–126, 2002.

[67] J. R. Shewchuk. Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations. In *Proceedings of the 19th Annual Symposium on Computational Geometry (SoCG)*, pages 181–190, 2003. doi:10.1145/777792.777821.

[68] J. Rock, T. Gupta, J. Thorsen, J. Gwak, D. Shin, and D. Hoiem. Completing 3D Object Shape From One Depth Image. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2484–2493, 2015. doi:10.1109/CVPR.2015.7298863.

[69] J. Vince. *Geometric Algebra for Computer Graphics*. Springer-Verlag London, 2008. ISBN: 978-1-84628-996-5.

[70] James F. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3), pages 235–256, 1982. doi:10.1145/357306.357310.

[71] John C. Hart. The Object Instancing Paradigm for Linear Fractal Modeling. In *Proceedings of Graphics Interface '92 (GI)*, pages 224–231, 1992.

[72] K. Åhlander. Mesh Generation for Symmetrical Geometries. In *Computational Science and its Applications - ICCSA 2005: International Conference, Singapore, May 9-12, 2005, Proceedings, Part I*, Volume 3480 of *Lecture Notes in Computer Science*, pages 657–668. Springer Berlin Heidelberg, 2005. doi:10.1007/11424758_69.

[73] K. L. Clarkson and P. W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete & Computational Geometry*, 4(5), pages 387–421, 1989. doi:10.1007/BF02187740.

[74] K. Modzelewski, R. Chintala, H. Moolamalla, S. Parke, and D. Hackler. Design of a 32nm Independently-Double-Gated FlexFET SOI Transistor. In *Proceedings of the 17th Biennial University/Government/Industry Micro/Nano Symposium (UGIM)*, pages 64–67, 2008. doi:10.1109/UGIM.2008.24.

[75] L. A. Freitag and C. Ollivier-Gooch. Tetrahedral Mesh Improvement Using Swapping and Smoothing. *International Journal for Numerical Methods in Engineering*, 40(21), pages 3979–4002, 1997. doi:10.1002/(SICI)1097-0207(19971115)40:21<3979::AID-NME251>3.0.CO;2-9.

[76] L. Branets and G. F. Carey. A Local Cell Quality Metric and Variational Grid Smoothing Algorithm. *Engineering with Computers*, 21(1), pages 19–28, 2005. doi:10.1007/s00366-005-0309-7.

[77] L. C. Evans, R. F. Gariepy. *Measure Theory and Fine Properties of Functions, Revised Edition*. Chapman and Hall/CRC Press, 2015. ISBN: 978-1-4822-4238-6.

[78] L. Chen. Mesh Smoothing Schemes Based on Optimal Delaunay Triangulations. In *Proceedings of the 13th International Meshing Roundtable (IMR)*, pages 109–120, 2004.

[79] L. Freitag, M. Jones, and P. Plassmann. An Efficient Parallel Algorithm for Mesh Smoothing. In *Proceedings of the 4th International Meshing Roundtable (IMR)*, pages 47–58, 1995.

[80] L. Kettner, K. Mehlhorn, S. Pion, S. Schirra, and C. Yap. Classroom Examples of Robustness Problems in Geometric Computations. In *Algorithms - ESA 2004: 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004. Proceedings*, Volume 3221 of *Lecture Notes in Computer Science*, pages 702–713. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-30140-0_62.

[81] M. A. Yerry and M. S. Shephard. A Modified Quadtree Approach To Finite Element Mesh Generation. *IEEE Computer Graphics and Applications*, 3(1), pages 39–46, 1983. doi:10.1109/MCG.1983.262997.

[82] M. A. Yerry and M. S. Shephard. Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique. *International Journal for Numerical Methods in Engineering*, 20(11), pages 1965–1990, 1984. doi:10.1002/nme.1620201103.

[83] M. Bern, D. Eppstein, and J. Erickson. Flipping Cubical Meshes. *Engineering with Computers*, 18(3), pages 173–187, 2002. doi:10.1007/s003660200016.

[84] M. Bokeloh, A. Berner, M. Wand, H.-P. Seidel, and A. Schilling. Symmetry Detection Using Feature Lines. *Computer Graphics Forum*, 28(2), pages 697–706, 2009. doi:10.1111/j.1467-8659.2009.01410.x.

[85] M. Brewer, L. F. Diachin, P. Knupp, T. Leurent, and D. Melander. The Mesquite Mesh Quality Improvement Toolkit. In *Proceedings of the 12th International Meshing Roundtable (IMR)*, pages 505–513, 2003.

[86] M. Caroli and M. Teillaud. Computing 3D Periodic Triangulations. In *Algorithms - ESA 2009: 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, Volume 5757 of *Lecture Notes in Computer Science*, pages 59–70. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04128-0_6.

[87] M. J. Pratt. Introduction to ISO 10303 - The STEP Standard for Product Data Exchange. *Journal of Computing and Information Science in Engineering*, 1(1), pages 102–103, 2001. doi:10.1115/1.1354995.

[88] M. L. Staten, J. F. Shepherd, and K. Shimada. Mesh Matching - Creating Conforming Interfaces Between Hexahedral Meshes. In *Proceedings of the 17th International Meshing Roundtable (IMR)*, pages 467–484, 2008. doi:10.1007/978-3-540-87921-3_28.

[89] M. L. Staten, J. F. Shepherd, F. Ledoux, and K. Shimada. Hexahedral Mesh Matching: Converting Non-Conforming Hexahedral-to-Hexahedral Interfaces into Conforming Interfaces. *International Journal for Numerical Methods in Engineering*, 82(12), pages 1475–1509, 2010. doi:10.1002/nme.2800.

[90] M. Murphy, D. Mount, and C. W. Gable. A Point-Placement Strategy for Conforming Delaunay Tetrahedralization. *International Journal of Computational Geometry & Applications*, 11(6), pages 669–682, 2001. doi:10.1142/S0218195901000699.

[91] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. J. Guibas. Discovering Structural Regularity in 3D Geometry. *ACM Transactions on Graphics*, 27(3), pages 43:1–43:11, 2008. doi:10.1145/1360612.1360642.

[92] M. Pharr. *GPU Gems 2: Programming Techniques for High-performance Graphics and General-purpose Computation*. Addison-Wesley Longman, Amsterdam, 2005. ISBN: 978-0-321-33559-3.

[93] M. Szilvi-Nagy and G. Mátyási. Analysis of STL Files. *Mathematical and Computer Modelling*, 38(7-9), pages 945–960, 2003. doi:10.1016/S0895-7177(03)90079-3.

[94] M.J. Atallah. On Symmetry Detection. *IEEE Transactions on Computers*, 34(7), pages 663–666, 1985. doi:10.1109/TC.1985.1676605.

[95] N. Amenta, M. Bern, and D. Eppstein. Optimal Point Placement for Mesh Smoothing. *Journal of Algorithms*, 30(2), pages 302–322, 1999. doi:10.1006/jagm.1998.0984.

[96] N. J. Mitra and M. Pauly. Symmetry for Architectural Design. In *Proceedings of the 2008 Advances in Architectural Geometry, Conference Proceedings (AAG)*, pages 13–16, 2008.

[97] N. J. Mitra, L. Guibas, and M. Pauly. Symmetrization. *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2007*, 26(3), pages 63:1–63:8, 2007. doi:10.1145/1276377.1276456.

[98] N. J. Mitra, L. J. Guibas, and M. Pauly. Partial and Approximate Symmetry Detection for 3D Geometry. *ACM Transactions on Graphics*, 25(3), pages 560–568, 2006. doi:10.1145/1141911.1141924.

[99] N. J. Mitra, M. Pauly, M. Wand, and D. Ceylan. Symmetry in 3D Geometry: Extraction and Applications. *Computer Graphics Forum*, 32(6), pages 1–23, 2013. doi:10.1111/cgf.12010.

[100] N. Kowalski, F. Ledoux, and P. Frey. A PDE Based Approach to Multidomain Partitioning and Quadrilateral Meshing. In *Proceedings of the 21th International Meshing Roundtable (IMR)*, pages 137–154, 2012. doi:10.1007/978-3-642-33573-0_9.

[101] N. Kowalski, F. Ledoux, and P. Frey. Block-Structured Hexahedral Meshes for CAD Models Using 3D Frame Fields. In *Proceedings of the 23th International Meshing Roundtable (IMR)*, pages 59–71, 2014. doi:10.1016/j.proeng.2014.10.373.

[102] N. Kowalski, F. Ledoux, M. L. Staten, and S. J. Owen. Fun Sheet Matching: Towards Automatic Block Decomposition for Hexahedral Meshes. *Engineering with Computers*, 28(3), pages 241–253, 2012. doi:10.1007/s00366-010-0207-5.

[103] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals (Seventh Edition)*. Elsevier, Oxford, 2013. ISBN: 978-1-85617-633-0.

[104] O. Faugeras. *Three-Dimensional Computer Vision - A Geometric Viewpoint*. MIT University Press Group, 1993. ISBN: 978-0-262-06158-2.

[105] P. J. Frey and P.-L. George. *Mesh Generation: Application to Finite Elements (Second Edition)*. Iste, 2008. ISBN: 978-1-84821-029-5.

[106] P. 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 (IMR)*, pages 505–513, 1998.

[107] P.-O. Persson, M. L. Staten, H. Wu, and S. Gao. Automatic Swept Volume Decomposition Based on Sweep Directions Extraction for Hexahedral Meshing. In *Proceedings of the 23th International Meshing Roundtable (IMR)*, pages 136–148, 2014. doi:10.1016/j.proeng.2014.10.379.

[108] P.-O. Persson, M. L. Staten, H. Zhu, J.Chen, H. Wu, and S. Gao. Direct Editing on Hexahedral Mesh Through Dual Operations. In *Proceedings of the 23th International Meshing Roundtable (IMR)*, pages 149–161, 2014. doi:10.1016/j.proeng.2014.10.380.

[109] P. P. Pébay, D. Thompson, J. Shepherd, P. Knupp, C. Lisle, V. A. Magnotta, and N. M. Grosland. New Applications of the Verdict Library for Standardized Mesh Verification Pre, Post, and End-to-End Processing. In *Proceedings of the 16th International Meshing Roundtable (IMR)*, pages 535–552, 2007. doi:10.1007/978-3-540-75103-8_30.

[110] R. D. Cook. *Finite Element Modeling for Stress Analysis*. Wiley, 1995. ISBN: 978-0-471-10774-3.

[111] R. Eymard, T. Gallouët, and R. Herbin. Finite Volume Methods. In *Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3)*, Volume 7 of *Handbook of Numerical Analysis*, pages 713–1018. Elsevier, 2000. doi:10.1016/S1570-8659(00)07005-8.

[112] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007. ISBN: 978-0-89871-629-0.

[113] R. Schneiders. A Grid-Based Algorithm for the Generation of Hexahedral Element Meshes. *Engineering with Computers*, 12(3), pages 168–177, 1996. doi:10.1007/BF01198732.

[114] R. Schneiders. Refining Quadrilateral and Hexahedral Element Meshes. In *Proceedings of the 5th International Conference on Grid Generation in Computational Field Simulations*, pages 679–688, 1996.

[115] R. Schneiders. Algorithms for Quadrilateral and Hexahedral Mesh Generation. In *Proceedings of the VKI Lecture Series on Computational Fluid Dynamic (VKI-LS)*, pages 239–267, 2000.

[116] S. A. Canann, J. R. Tristano, and M. L. Staten. An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral, and Quad-Dominant Meshes. In *Proceedings of the 7th International Meshing Roundtable (IMR)*, pages 479–494, 1998.

[117] S. Bellavia, J. Gondzio, and B. Morini. A Matrix-Free Preconditioner for Sparse Symmetric Positive Definite Systems and Least-Squares Problems. *SIAM Journal on Scientific Computing*, 35(1), pages A192–A211, 2013. doi:10.1137/110840819.

[118] S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers (Seventh Edition)*. Mcgraw-Hill Education - Europe, 2014. ISBN: 978-0-07-339792-4.

[119] S. E. Pav and N. J. Walkington. Robust Three Dimensional Delaunay Refinement. In *Proceedings of the 13th International Meshing Roundtable (IMR)*, pages 145–156, 2004.

[120] S. Fortune and C. J. Van Wyk. Efficient Exact Arithmetic for Computational Geometry. In *Proceedings of the 9th Annual Symposium on Computational Geometry (SoCG)*, pages 163–172, 1993. doi:10.1145/160985.161015.

[121] S. J. Owen. A Survey of Unstructured Mesh Generation Technology. In *Proceedings of the 7th International Meshing Roundtable (IMR)*, pages 239–267, 1998.

[122] S. J. Owen, M. L. Staten, S. A. Canann, and S. Saigal. Q-Morph: An Indirect Approach to Advancing Front Quad Meshing. *International Journal for Numerical Methods in Engineering*, 44(9), pages 1317–1340, 1999. doi:10.1002/(SICI)1097-0207(19990330)44:9<1317::AID-NME532>3.0.CO;2-N.

[123] S.-K. Chang and C. K. Chow. The Reconstruction of Three-Dimensional Objects from Two Orthogonal Projections and Its Application to Cardiac Cineangiography. *IEEE Transactions on Computers*, 22(1), pages 18–28, 1973. doi:10.1109/T-C.1973.223596.

[124] S. K. Khattri. A New Smoothing Algorithm for Quadrilateral and Hexahedral Meshes. In *Computational Science - ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006. Proceedings, Part II*, Volume 3992 of *Lecture Notes in Computer Science*, pages 239–246. Springer Berlin Heidelberg, 2006. doi:10.1007/11758525_32.

[125] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer-Verlag, Wien - New York, 1984. ISBN: 978-3-7091-8754-8.

[126] S. Selberherr. Zum sogenannten Gummel-Stern. Unpublished; Based on Private Communcation with H.K. Gummel in 1987, 2008.

[127] S.-W. Cheng, T. K. Dey, and J. Shewchuk. *Delaunay Mesh Generation*. Chapman & Hall/CRC Press, 2012. ISBN: 978-1-58488-730-0.

[128] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Quality Meshing for Polyhedra with Small Angles. In *Proceedings of the 20th Annual Symposium on Computational Geometry (SoCG)*, pages 290–299, 2004. doi:10.1145/997817.997862.

[129] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Silver Exudation. *Journal of the ACM*, 47(5), pages 883–904, 2000. doi:10.1145/355483.355487.

[130] Schöberl, Joachim. NETGEN - An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules. *Computing and Visualization in Science*, 1(1), pages 41–52, 1997. doi:10.1007/s007910050004.

[131] J. R. Shewchuck. What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures. unpublished preprint, 2002. URL: http://www.cs.berkeley.edu/~jrs/papers/elemj.pdf.

[132] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering (Third Edition)*. CRC Press, 2008. ISBN: 978-1-56881-424-7.

[133] T. Blacker, S. J. Owen, M. L. Staten, R. W. Quadros, B. Hanks, B. Clark, R. J. Meyers, C. Ernst, K. Merkley, R. Morris, C. McBride, C. Stimpson, M. Plooster, and S. Showman. *CUBIT Geometry and Mesh Generation Toolkit 15.1 User Documentation*, 2016. URL: https://cubit.sandia.gov/public/15.1/help_manual/Printed_Documentation/Cubit_15.1_User_Documentation.pdf.

[134] T. D. Blacker and M. B. Stephenson. Paving: A New Approach to Automated Quadrilateral Mesh Generation. *International Journal for Numerical Methods in Engineering*, 32(4), pages 811–847, 1991. doi:10.1002/nme.1620320410.

[135] T. W. Gamelin and R. E. Greene. *Introduction to Topology (Second Edition)*. Dover Publications Inc., 1999. ISBN: 978-0-486-40680-0.

[136] V. T. Rajan. Optimality of the Delaunay Triangulation in $\mathbb{R}^d$. *Discrete & Computational Geometry*, 12(2), pages 189–202, 1994. doi:10.1007/BF02574375.

[137] X. Wu, M. Wand, K. Hildebrandt, P. Kohli, and H.-P. Seidel. Real-Time Symmetry-Preserving Deformation. *Computer Graphics Forum*, 33(7), pages 229–238, 2014. doi:10.1111/cgf.12491.

[138] Y. Ito, A. M. Shih, and B. 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), pages 1809–1833, 2009. doi:10.1002/nme.2470.

[139] Y. Li, W. Wang, R. Ling, and C. Tu. Shape Optimization of Quad Mesh Elements. *Computers & Graphics*, 35(3), pages 444–451, 2011. doi:10.1016/j.cag.2011.03.037.

[140] Y. Lipman, X. Chen, I. Daubechies, and T. Funkhouser. Symmetry Factored Embedding and Distance. *ACM Transactions on Graphics*, 29(4), pages 103:1–103:12, 2010. doi:10.1145/1778765.1778840.

[141] Y. Saad. *Iterative Methods for Sparse Linear Systems (Second Edition)*. Society for Industrial and Applied Mathematics, 2003. ISBN: 978-0-89871-534-7.

[142] Y. Wang, K. Xu, J. Li, H. Zhang, A. Shamir, L. Liu, Z. Cheng, and Y. Xiong. Symmetry Hierarchy of Man-Made Objects. *Computer Graphics Forum*, 30(2), pages 287–296, 2011. doi:10.1111/j.1467-8659.2011.01885.x.

Florian Rudolf

Curriculum Vitae

Erich-Fried-Weg 1/9/905
1220 Wien
+43 650 9014096
flo.rudy@gmail.com
26. August 1984
Exempted from military service

Education

since 07/2012 **Doctoral program in Engineering Sciences**, *TU Wien, AT*.
Applied mathematics in electrical engineering

10/2003–06/2012 **Diploma program in Technical Mathematics**, *TU Wien, AT*.
Concentration in computer science

09/1998–06/2003 **Secondary College for Industrial Management**, *HTL Ungargasse, AT*.
Special training focus business information technology

Employment

since 09/2012 **Freelancer**, *Langenzersdorf, AT*, part-time.
Consulting and services in IT

since 07/2012 **TU Wien**, *Vienna, AT*, part-time.
Project assistant, PhD position

02/2007–10/2010 **Center Communication Systems GmbH**, *Vienna, AT*, part-time.
Software developer for algorithmic, image- and videoprocessing

10/2005–07/2008 **TU Wien**, *Vienna, AT*, part-time.
Tutor for programming, computer graphics, and real-time rendering

Projects

since 2013 **Google Summer of Code program**.
Adminstrator and Mentor for the CSE@TU Wien organization
http://www.iue.tuwien.ac.at/cse/

since 2012 **ViennaMesh**.
Open source mesh generation and adaptation software framework
https://github.com/viennamesh/viennamesh-dev

since 2012 **ViennaGrid**.
Open source mesh data structure library with a focus on volumetric meshes for discretization based simulations
https://github.com/viennagrid/viennagrid-dev

2012 **The Finite Element Method On Massively Parallel Computing Architectures**.
Diploma thesis, TU Wien, Finite Elements on GPU architectures using OpenCL
http://catalogplus.tuwien.ac.at/UTW:UTW:UTW_aleph_acc000556712

2003 **Diploma project, HTL Ungargasse**.
Tomography algorithm for measuring ion thrusters

Own Publications

[1] K. Rupp, J. Weinbub, and F. Rudolf. Automatic Performance Optimization in ViennaCL for GPUs. In *Proceedings of the 9th Workshop on Parallel/High-Performance Object-Oriented Scientific Computing (POOSC)*, pages 6:1 - 6:6, 2010. doi:10.1145/2039312.2039318

[2] K. Rupp, F. Rudolf, and J. Weinbub. ViennaCL - A High Level Linear Algebra Library for GPUs and Multi-Core CPUs. In *Proceedings of the International Workshop on GPUs and Scientific Applications (GPUScA)*, pages 51 - 56, 2010.

[3] K. Rupp, J. Weinbub, and F. Rudolf. Highly Productive Application Development with ViennaCL for Accelerators. *AGU Fall Meeting*, San Francisco, USA, 2012.

[4] K. Rupp, Ph. Tillet, F. Rudolf, and J. Weinbub. ViennaCL - Portable High Performance at High Convenience. In *Proceedings of the 2013 European Numerical Mathematics and Advanced Applications (ENUMATH)*, pages 1 - 2, 2013.

[5] F. Rudolf, K. Rupp, and S. Selberherr. ViennaMesh - a Highly Flexible Meshing Framework. In *Abstracts of the 4th International Congress on Computational Engineering and Sciences (FEMTEC)*, 1 page, 2013.

[6] K. Rupp, F. Rudolf, and J. Weinbub. A Discussion of Selected Vienna-Libraries for Computational Science. In *Proceedings of C++Now (2013)*, 10 pages, 2013.

[7] K. Rupp, Ph. Tillet, F. Rudolf, J. Weinbub, T. Grasser, and A. Jüngel. Performance Portability Study of Linear Algebra Kernels in OpenCL. In *Proceedings of the International Workshop on OpenCL 2013 & 2014 (IWOCL)*, pages 8:1 - 8:11, 2014. doi:10.1145/2664666.2664674.

[8] F. Rudolf, J. Weinbub, K. Rupp, and S. Selberherr. The Meshing Framework ViennaMesh for Finite Element Applications. *Journal of Computational and Applied Mathematics*, 270, pages 166 - 177, 2014. doi:10.1016/j.cam.2014.02.005.

[9] S. E. Tyaginov, M. Bina, J. Franco, Y. Wimmer, F. Rudolf, H. Enichlmair, J.M. Park, B. Kaczer, H. Ceric, and T. Grasser. Dominant Mechanism of Hot-Carrier Degradation in Short- and Long-Channel Transistors. In *2014 IEEE International Integrated Reliability Workshop Final Report (IIRW)*, pages 63 - 68, 2014. doi:10.1109/IIRW.2014.7049512.

[10] Y. Wimmer, S. E. Tyaginov, F. Rudolf, K. Rupp, M. Bina, H. Enichlmair, J.M. Park, R. Minixhofer, H. Ceric, and T. Grasser. Physical Modeling of Hot-Carrier Degradation in nLDMOS Transistors. In *2014 IEEE International Integrated Reliability Workshop Final Report (IIRW)*, pages 58 - 62, 2014. doi:10.1109/IIRW.2014.7049511.

[11] L. Filipovic, F. Rudolf, E. Baer, P. Evanschitzky, J. Lorenz, F. Roger, A. P. Singulani, R. Minixhofer, and S. Selberherr. Three-Dimensional Simulation for the Reliability and Electrical Performance of Through-Silicon Vias. In *Proceedings of the 19th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 341 - 344, 2014. doi:10.1109/SISPAD.2014.6931633.

[12] F. Rudolf, J. Weinbub, K. Rupp, A. Morhammer, and S. Selberherr. Template-Based Mesh Generation for Semiconductor Devices. In *Proceedings of the 19th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 217 - 220, 2014. doi:10.1109/SISPAD.2014.6931602.

[13] F. Rudolf, Y. Wimmer, J. Weinbub, K. Rupp, and S. Selberherr. Mesh Generation Using Dynamic Sizing Functions. In *Proceedings of the 4th European Seminar on Computing (ESCO)*, page 191, 2014.

[14] K. Rupp, F. Rudolf, J. Weinbub, A. Jungel, and T. Grasser. Automatic Finite Volume Discretizations Through Symbolic Computations. In *Proceedings of the 4th European Seminar on Computing (ESCO)*, page 192, 2014.

[15] J. Weinbub, K. Rupp, and F. Rudolf. A Flexible Material Database for Computational Science and Engineering. In *Proceedings of the 4th European Seminar on Computing (ESCO)*, page 226, 2014.

[16] P. Sharma, S. E. Tyaginov, Y. Wimmer, F. Rudolf, K. Rupp, M. Bina, H. Enichlmair, J.M. Park, R. Minixhofer, H. Ceric, and T. Grasser. Modeling of Hot-Carrier Degradation in nLDMOS Devices: Different Approaches to the Solution of the Boltzmann Transport Equation. *IEEE Transactions on Electron Devices*, 62, pages 1811 - 1818, 2015. doi:10.1109/TED.2015.2421282.

[17] F. Rudolf, K. Rupp, J. Weinbub, A. Morhammer, and S. Selberherr. Transformation Invariant Local Element Size Specification. *Applied Mathematics and Computation*, 267, pages 195 - 206, 2015. doi:10.1016/j.amc.2015.04.027.

[18] P. Sharma, S. E. Tyaginov, Y. Wimmer, F. Rudolf, K. Rupp, H. Enichlmair, J.M. Park, H. Ceric, and T. Grasser. Comparison of Analytic Distribution Function Models for Hot-Carrier Degradation in nLDMOSFETs. *Microelectronics Reliability*, 55, pages 1427 - 1432, 2015. doi:10.1016/j.microrel.2015.06.021.

[19] J. Weinbub, M. Wastl, K. Rupp, F. Rudolf, and S. Selberherr. ViennaMaterials - A Dedicated Material Library for Computational Science and Engineering. *Applied Mathematics and Computation*, 267, pages 282 - 293, 2015. doi:10.1016/j.amc.2015.03.094.

[20] F. Rudolf, J. Weinbub, K. Rupp, P. Resutik, A. Morhammer, and S. Selberherr. Free Open Source Mesh Healing for TCAD Device Simulations. In *Large-Scale Scientific Computing*, Volume 9374 of *Lecture Notes in Computer Science*, pages 293 - 300, Springer, 2015. doi:10.1007/978-3-319-26520-9_32.

[21] F. Rudolf, J. Weinbub, K. Rupp, A. Morhammer, and S. Selberherr. Symmetry-Aware 3D Volumetric Mesh Generation - An Analysis of Performance and Element Quality. In *Proceedings of the 24th International Meshing Roundtable (IMR24)*, 2015. Research Note.

[22] P. Sharma, S. E. Tyaginov, Y. Wimmer, F. Rudolf, K. Rupp, H. Enichlmair, J.M. Park, H. Ceric, and T. Grasser. Comparison of Analytic Distribution Function Models for Hot-Carrier Degradation in nLDMOSFETs. In *Abstracts of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis*, page 60, 2015.

[23] P. Sharma, M. Jech, S. E. Tyaginov, F. Rudolf, K. Rupp, H. Enichlmair, J.M. Park, and T. Grasser. Modeling of Hot-Carrier Degradation in LDMOS Devices Using a Drift-Diffusion Based Approach. In *Proceedings of the 20th International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 60 - 63, 2015. doi:10.1109/SISPAD.2015.7292258.

[24] F. Rudolf, J. Weinbub, K. Rupp, P. Resutik, and S. Selberherr. Mesh Healing for TCAD Simulations. In *Abstracts International Conference on Large-Scale Scientific Computations (LSSC)*, page 66, 2015.

[25] P. Sharma, S. E. Tyaginov, Y. Wimmer, F. Rudolf, K. Rupp, M. Bina, H. Enichlmair, J.M. Park, H. Ceric, and T. Grasser. Predictive and Efficient Modeling of Hot-Carrier Degradation in nLDMOS Devices. In *Proceedings of the 2015 IEEE 27th International Symposium on Power Semiconductor Devices & IC's (ISPSD)*, pages 389 - 392, 2015. doi:10.1109/ISPSD.2015.7123471.

[26] P. Sharma, S. E. Tyaginov, Y. Wimmer, F. Rudolf, H. Enichlmair, J.M. Park, H. Ceric, and T. Grasser. A Model for Hot-Carrier Degradation in nLDMOS Transistors Based on the Exact Solution of the Boltzmann Transport Equation Versus the Drift-Diffusion Scheme. In *Proceedings of 2015 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon*, pages 21 - 24, 2015. doi:10.1109/ULIS.2015.7063763.

[27] K. Rupp, Ph. Tillet, T. St Clere Smithe, N. Karovic, J. Weinbub, and F. Rudolf. ViennaCL - Fast Linear Algebra for Multi and Many-Core Architectures. *SIAM Conference on Computational Science and Engineering*, Utah, USA, 2015.

[28] M. Jech, P. Sharma, S. E. Tyaginov, F. Rudolf, and T. Grasser. On the Limits of Applicability of Drift-Diffusion Based Hot Carrier Degradation Modeling. *Japanese Journal of Applied Physics*, 55, pages 1 - 6, 2016. doi:10.7567/JJAP.55.04ED14.

[29] P. Sharma, S. E. Tyaginov, M. Jech, Y. Wimmer, F. Rudolf, H. Enichlmair, J.M. Park, H. Ceric, and T. Grasser. The Role of Cold Carriers and the Multiple-Carrier Process of Si-H Bond Dissociation for Hot-Carrier Degradation in n- and p-channel LDMOS Devices. *Solid-State Electronics*, 115, pages 185 - 191, 2016. doi:10.1016/j.sse.2015.08.014.

[30] A. Morhammer, K. Rupp, F. Rudolf, and J. Weinbub. Optimized Sparse Matrix-Matrix Multilication for Multi-Core CPUs, GPUs and MICs. In *Book of Abstracts of the 2016 Austrian HPC Meeting (AHPC)*, page 23, 2016.